Delft University of Technology
Master of Science Thesis in Embedded Systems

μLightDigit: A TinyML System for Contactless Digit Recognition using Ambient Light

Koen Goedemondt

TU Delft Delft University of Technology

μLightDigit: A TinyML System for Contactless Digit Recognition using Ambient Light

Master of Science Thesis in Embedded Systems

Embedded Systems Group
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Van Mourik Broekmanweg 6, 2628 XE Delft, The Netherlands

Koen Goedemondt
k.s.goedemondt@student.tudelft.nl
kgoedemondt@gmail.com

22^{th} of November 2022

Author
Koen Goedemondt (k.s.goedemondt@student.tudelft.nl)
(kgoedemondt@gmail.com)
Title
μLightDigit: A TinyML System for Contactless Digit Recognition using Ambient Light
MSc Presentation Date
28^{th} of November 2022

Graduation Committee

Prof. dr. Koen Langendoen (Chair)	Delft University of Technology
Dr. Qing Wang	Delft University of Technology
Dr. Jie Yang	Delft University of Technology
Dr. Raj Rajan	Delft University of Technology

Abstract

This thesis describes the design and implementation of μLightDigit, which is the second iteration of the LightDigit project: a contactless air-writing system based on ambient light detection with embedded deep learning using only simple photodiodes. The system is able to classify digits 0–9 written in the air by detecting the dynamic hand shadow using a 3×3 array of simple photodiodes. A novel TinyML system is developed by transforming LightDigit from a Raspberry Pi 4 single-board computer to an STM32H743 microcontroller. This transition is burdened with a significant reduction in computation power while having the goal of creating a more robust and adaptable system. To achieve this, a new photodiode array is designed to overcome the saturation issues of the photodiode array developed for LightDigit. A preliminary analysis is performed to investigate the performance of three deep learning models on the microcontroller: a tiny convolution neural network (CNN), a tiny long short-term memory (LSTM) model, and a tiny Conv-LSTM hybrid. For the final system evaluation, the tiny CNN and ConvLSTM models are chosen which provided the best performance in the preliminary evaluation. In order to test the system's robustness, three different indoor setups and one outdoor setup are created. These setups are created to be reproducible and serve for the verification of the system in different ambient lighting conditions and light intensities. Shadow patterns are recorded for each of the setups and are compared to explain the digit recognition results. The tiny CNN model results in an average accuracy of 84.6% with a maximum of 97.0% and a minimum of 77.8% accuracy across the four different test setups. The tiny ConvLSTM model achieves an average accuracy of 82.4%, with a maximum of 89.3% and a minimum of 72.4%. The tiny ConvLSTM model outperforms the CNN model in the outdoor setup but is generally outclassed by the CNN in all indoor setups. The CNN model is the lightest model with 792 ms inference time and a total size of 138k parameters, which translates to 147 kB. The tiny ConvLSTM model is larger with 446k parameters at 494 kB and 947 ms inference time. For the real-time inference time on the STM32H743 microcontroller, both the tiny CNN and ConvLSTM models remain below 1 second, which is fast enough for real-time applications.

Preface

Before I started my MSc degree in Embedded Systems I studied Electrical Engineering, also at TU Delft. I became interested in machine learning after following a course on computational intelligence. The μLightDigit project piqued my interest because I was looking for a project with elements of machine learning, embedded systems, and electronics. My project was completed over approximately a one-year period at the Embedded Systems Group at TU Delft. During the thesis, I was able to have a unique experience by presenting part of my work at SITB 2022 [1].
I would like to thank my supervisors Qing Wang and Jie Yang. I have to give special thanks to Qing, whose door was always open for questions and who helped me tremendously with the writing and structuring of the thesis. I would also like to thank many other wonderful people which I have met over the past year, many of whom have been a great help to me. It was always a great time to have coffee or lunch together and I can say that I have made new friends this year. Some people I would like to thank in particular. Eric Wang, for useful conversations on various machine learning related topics. Miguel Chavez Tapia, for teaching me about 3D printing. Vito Kortbeek, Girish Vaidya, Talia Xu and Keyarash Ghiasi for helping me with the electronics design. Hao Liu for general help with my thesis. And of course my friends and family for helping me through the busy and stressful periods of deadlines.

Koen Goedemondt

Delft, The Netherlands
22nd November 2022

Contents

Chapter 1

Introduction

When the COVID-19 pandemic started at the end of 2019, nobody had any idea how profound the impact on daily life would be. Lockdowns lasting for weeks or months and obligatory face masks and social distancing became the new normal. Even though the end is in sight almost 3 years later, some regions still have restrictions in place to hamper the spread of the disease. As part of these restrictions, it was advised to wash your hands frequently and avoid touching public surfaces. This made seeing people wearing latex gloves in public a common occurrence. Consequently, the idea was sparked for a system that allows people to enter numbers in a contact-free manner using ambient light. Such a system could be used to replace keypads or touchscreens in public settings, such as in elevators, payment terminals and ATMs.

In existing research, a plethora of hand gesture recognition techniques are used, such as wearable sensors, computer vision, WiFi, Radar, radio-frequency identification (RFID), ultrasound and many more [2]. However, visible light is chosen as it is ubiquitous in both indoor and outdoor environments, and it is already naturally contactless, which is the goal of this research. Additionally, no involved transmitter-receiver setup is required such as is the case with radio-based systems. Virtually all state-of-the-art research used deep learning as the preferred classification approach, as it is well suited to the time-series data which is collected, and when applied correctly can provide excellent results [3].

The system which is proposed in this work uses visible light as the sensing medium and provides natural and intuitive interaction by allowing digits to be written in the air. The core principle of operation is capturing the shadow created by the hand of the user, and inferring the written digit from the movement of the shadow. No camera or additional information is used for classifying the written digit. The first version of this system was created by Hao Liu as his master thesis project which he presented in 2021 [4]. The resulting system was dubbed *LightDigit*, a term that will be used throughout this thesis. The work done by Hao resulted in the LightDigit dataset, containing a total of 26440 instances of air-written digits from 0 to 9, and includes data from a total of 24 participants. This data was collected using a sensor grid of 16 simple photodiodes, which was developed specifically for the project. Using this dataset, two deep learning models were trained and ran on a Raspberry Pi which were able to classify digits in real-time with approximately 90% accuracy. However, challenges still exist which prevent the system from being practically viable. Therefore, the focus of this thesis will be to create a second version of the system, which will attempt to address the problems and limitations of the first version. The focus of this thesis will be two-fold. The first area of focus will be to move from a Raspberry Pi based system to a *TinyML*, i.e., Tiny Machine Learning system, which runs machine learning algorithms on cheap, low-power hardware, both in terms of energy consumption and computing capability. TinyML is

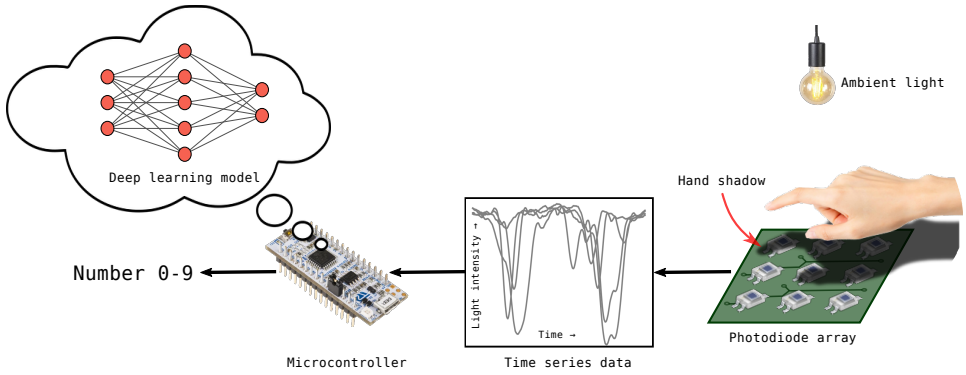


Figure 1.1: **Concept of the μLightDigit system.**

an emerging field of research that has surged in popularity in recent years. Moving the system to a TinyML platform could greatly decrease power consumption, increase efficiency, reduce cost, and also enables the potential for battery-powered operation of the system. Due to this transition to a microcontroller, the system is named **μLightDigit**, with the letter μ meaning *micro* as it represents a smaller, more efficient system compared to the original LightDigit. The second area of focus will be to address the problems and limitations of the first system, predominantly to improve the dynamic range in different ambient light intensities. The following sections will reformulate the areas of focus mentioned above into a research objective, and associated research challenges and provide an overview of the contents of this thesis.

1.1 Research objective

The objective of this thesis is to create a *TinyML* system that is able to correctly classify digits 0–9 written in the air, using only visible ambient light as the sensing medium. This should be done using simple photodiodes while being robust against different lighting conditions. In other words, a tiny machine learning model should be developed for a microcontroller, which can classify time-series light information to digits 0–9.

1.2 Research challenges

For the original LightDigit project, three major research challenges were formulated, which can be summarized as follows: Firstly, the system should work for different people who have different writing styles. Secondly, it should work in many lighting environments. And finally, it should work with limited computational resources on low-power hardware. While these challenges remain relevant, **the focus of μLightDigit** is placed on low-computational resources and system robustness in different lighting conditions, which follows directly from limitations in the old system and to further advance research in the field of TinyML. The research challenges of this thesis are formulated as follows:

1. *Design and implement a TinyML gesture recognition system using a commodity microcontroller.* This challenge presents a paradigm shift compared to the previous LightDigit system, as this further reduces the already limited computational resources of a Raspberry Pi. The amount of RAM on a microcontroller generally ranges from kilobytes to megabytes. The same is true for the amount of storage, which is generally in the megabytes

range. Both of these specifications are orders of magnitude lower than a Raspberry Pi. On top of that, the microcontroller will run bare metal, thus without an operating system.

2. *Improve the system robustness against variation in light conditions.* There are two facets to this challenge, with the first one resulting directly from limitations in the previous system. Due to the old photodiode array having a fixed light sensitivity, it suffers from dynamic range issues in bright lighting conditions. These issues already start at 500 lux, which greatly limits the practical applicability in both indoor and outdoor scenarios. Secondly, there is no clear data on the robustness of the system in environments with multiple light sources, which in turn cause malformed shadow patterns. Such environments are ubiquitous indoors and should be investigated.

1.3 Contributions

The following list summarizes the contributions brought forth by this thesis:

1. An in-depth review of state-of-the-art research into hand gesture recognition techniques and TinyML. For gesture recognition, there will be a focus on methods used for data acquisition through various sensing media, such as computer vision, radar, and infrared. Additionally, the used classification methods and their achieved accuracies. For TinyML, there will be a focus on current application research, available software frameworks, hardware for creating TinyML systems and new developments in the TinyML hardware market. To aid the reader in their possible future TinyML projects, there will also be an overview provided for frequently used TinyML development boards.

2. New research into the development of a TinyML gesture recognition system and performance analysis of a convolutional neural network (CNN), long short-term memory (LSTM) model, and a convolutional-LSTM (ConvLSTM) hybrid model. The models are run on an STM32H743 with ARM Cortex-M7 CPU, and are analyzed in terms of accuracy, inference time, and robustness by evaluating them in various test setups.

3. A novel design of a low-cost photodiode array on a custom PCB using OPT101 photodiodes with adjustable ambient light sensitivity, up to 15k lux. This is paired with automatic calibration and an improved sensing algorithm using an APDS9930 proximity sensor.

4. Analysis of the system in four different reproducible test setups, showing the system performance in different light conditions. Additionally, the shadow patterns are recorded for each different lighting condition and its impact on model performance is discussed.

5. Two different optimized deep learning models, a tiny CNN and ConvLSTM are able to run inference in real-time (under 1 second) on a commodity microcontroller. The CNN model results in an average accuracy of 84.6%, with a maximum of 97.0% and a minimum of 77.8% accuracy across the four different test setups. The ConvLSTM model achieves an average accuracy of 82.4%, with a maximum of 89.3% and a minimum of 72.4%.

1.4 Thesis content

This thesis aims to provide a full overview of the design process, implementation, and evaluation of creating the TinyML-based digit recognition system, μLightDigit. Moreover, a review is performed of the related work in TinyML research and gesture recognition. The contents can be summarized as follows:

1. *Introduction* provided the motivations and main challenges which will be addressed in this work.

2. *Related Work* will be a review of state-of-the-art research in TinyML and gesture recognition.

3. *System Architecture* will provide a high-level overview of the system, which is split into three parts: the photodiode array, microcontroller, and classification algorithm.

4. *System Design* describes the hardware and software used for developing the system, and explains the design process of a new, photodiode array with adaptable light sensitivity.

5. *Tiny Deep Learning Algorithm* details the classification algorithm, which consists of two main parts: data preprocessing and the deep learning model. The data preprocessing steps are explained and three model architectures are provided for testing on the microcontroller: a CNN, LSTM and ConvLSTM hybrid model. Additionally, the photodiode calibration algorithm is explained, and an in-depth analysis of the input data is performed.

6. *Performance Evaluation* will outline the methods used to test and verify the designed hardware and deep learning models. A preliminary evaluation is performed using the LightDigit dataset to establish model performance, and which of the models are suitable for the system, the main restraints being inference time and model size. Additionally, model robustness to sampling distortion and the influence of data input formats are examined. After choosing suitable models, they are optimized and tested in four different reproducible test setups. The final results are analyzed using shadow patterns, input data image patterns, and confusion matrices for each model.

7. *Conclusion* summarizes the results obtained in the thesis and provides future work and recommendations.

Chapter 2

Related Work

2.1 Hand gesture recognition

Hand gesture recognition (HGR) aims to interpret human hand movements by a computer using sensors and algorithms. HGR can be categorized into static and dynamic gestures. Static gestures refer to the hand remaining still during the gesture, and dynamic gestures refer to the hand moving during the gesture. The main application of gesture recognition is contactless human-computer interaction [5]. More specific examples of applications include clinical surgery, robot control, domestic and industrial automation, virtual reality, and gaming [6]. The HGR workflow starts with data acquisition through some sensing medium, after which the data is processed and finally classified [7]. In recent research, classification is usually performed through deep learning with artificial neural networks. However, classical machine learning methods, such as k-nearest neighbors (kNN) and support vector machines (SVMs) are also used. The structure of this section is as follows. First, we will explore current research in gesture recognition, focusing on which types of technologies are used. Secondly, we look at air-writing, which can be considered a subcategory of gesture recognition. Finally, a more in-depth review is performed on gesture recognition and air writing using visible light as the sensing medium, since it is of greater relevance to this thesis.

2.1.1 Hand gesture recognition techniques

To determine the state of the art in gesture recognition research, recent papers and surveys on gesture recognition have been summarized below. The focus lies on which different types of sensing media and the classification methods are used, after which their relevance to this work is briefly discussed.

- *Infrared light* [8] - In this work, the sensor is an array of infrared photodiodes and infrared LEDs are placed in a line. Users move their hand over the sensor array and the infrared sensors capture the infrared light reflected from the hand of the user. They collect data for 27 different gestures ranging from simple swiping motions to more complex circular and shaking motions. This data is passed to a recurrent neural network (RNN) for classification, and they are able to achieve an accuracy of 95% for simple gestures and 93% for complex gestures.
- *Time-of-flight camera* [9] - In this study, a time-of-flight camera is used to collect depth information from the hand of the user. The authors record 9 different gestures indicating

various actions in a car. The gesture estimation algorithm works in two steps. Firstly, a hand pose is inferred using a CNN, after which a long short-term memory (LSTM) model is used for further feature extraction. Combined they can achieve an accuracy of 89.5%.

- *Computer vision* [10] - This study uses a camera to capture video of user hand motions. They then extract only three representative frames from the video sequence which are used for classification. For training, they use the Cambridge gesture recognition dataset which contains nine gesture classes. Using a long convolutional recurrent neural network (CNN-LSTM combination) model, they report an accuracy of 91%.

- *RFID* [11] - This study uses commercially available RFID tags which are arranged in a grid-like array. A single RFID antenna behind the array continuously measures signals emitted from the tags. As the user moves their finger close to the array, the reflection properties of individual tags change due to the multipath effect, allowing the system to determine the position of the finger. They design the system to both track a single finger of the user and allow multitouch gestures. The gestures which are defined are the 26 letters of the alphabet and 4 shapes and the multitouch gestures are rotating, swiping, and pinching motions. For classification, they use a CNN which is able to provide up to 92% classification accuracy.

- *Ultrasound* [12] - This study uses the reflection of high-frequency sound waves to capture hand gestures. They use the microphone and speaker which are already available on devices such as smartphones and smartwatches. A system is developed which estimates the user hand position from the phase shift of the received signal. Using this method they can successfully recognize short words and gestures with an accuracy of 91%.

- *Capacitive sensor* [13] - This study features a wearable bracelet with capacitive plates which is attached to the wrist of the user. As the corresponding hand moves, a change in capacitance between the hand and the capacitive plates can be measured. The authors define seven different static hand poses for classification, and create a TinyML model running on an Arduino Nano 33. By using only a single-layer CNN they are able to achieve a classification accuracy of 97%.

From [14] and [13] it is clear that wearable sensors can be successfully applied, however, we do not consider them relevant to this research. Using this method might be convenient when a user already has a device such as a smartwatch, but our work is aimed at deployment in a public setting. Therefore, wearables would defeat the purpose of a publicly available system. A large amount of gesture detection research is performed on sign language interpretation such as in [15] which is also corroborated by literature reviews such as [7]. Much of the modern research on this topic employs computer vision through both still images and video. For real-time computer vision applications, powerful hardware is required, as it is a computationally intensive task. However, as shown in [15], using smaller models which can run on embedded devices can be successfully deployed.

2.1.2 Air writing

Air writing can be considered a subcategory of gesture recognition, the goal being to classify the symbol which is being written in the air. This can be done through various sensing media such as the ones described above. The focus of this section is to assess the techniques used specifically for air writing data acquisition and relevant classification methods.

- *Time-of-flight camera* [16] - In this study, the authors use a Microsoft Kinect camera to extract the position of the user's fingertips. The Kinect can retrieve depth information

through the use of a regular camera and infrared transmitters and receivers. This information can then be used to determine the position in 3D space of various joints in the human body, including the fingertips. The authors use the position of the fingertip to train a deep neural network and are able to classify both digits and letters with an accuracy of over 90%.

- *Computer vision* [17] - This study uses images captured using a simple webcam to infer hand movements. The hand movements are converted to a trajectory and are projected as a simple 2-dimensional handwritten digit. This enables the use of the well-known MNIST handwritten digit dataset for model training. As a deep-learning based data processing pipeline combined with an LSTM classifier, enabling both single and multi digit numbers, they can classify with 98.7% and 85% accuracy respectively.

- *Radio* [18] - Three impulse radio ultra-wideband radar sensors are placed in triangular geometry. The user moves their hand between the sensors, while the sensors measure the delay of radio pulses that reflect off the hand of the user. The position of the hand is tracked by measuring the signal delay, allowing a trajectory of the hand to be inferred. They collect a dataset of 1000 samples for each digit and using a CNN they can achieve a classification accuracy of 97%.

- *Ultrasound* [19] - This work uses an array of four ultrasonic transceivers, placed 5 cm apart in a square-shaped matrix arrangement. Similar to the radar approach, using sound they are able to infer a trajectory of the hand of the user by measuring the delay of ultrasonic signals. Using a combination model of CNN and LSTM layers, they can achieve 99.5% accuracy.

- *Inertial measurement unit* [20] - The researchers attach a wearable inertial measurement unit (IMU) to the user's hand and test the writing of uppercase letters from the Latin alphabet. They use dynamic time warping to compare the written letters to a template to determine which letter was written. The classification accuracy is heavily person-dependent, but using data collected and tested by the same user is able to achieve 84.6% accuracy.

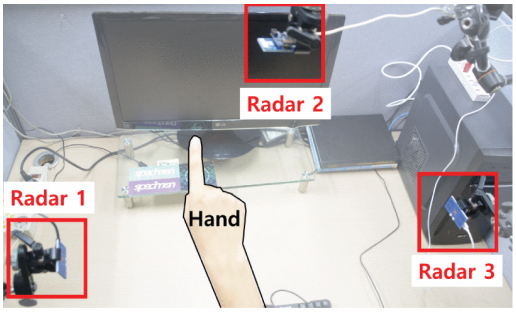
Vision-based approaches provide a large portion of research into gesture recognition. However, the approach proposed in this work aims to use only simple photodiodes, which in turn conserves user privacy. Another approach that has been researched frequently and applied successfully are radar based solutions. However, such solutions are impractical in the use case scenarios we propose, because radar requires several transceivers spaced around the sensing area in a specific orientation, making it more difficult to install as well as it being immobile. The research also shows that a common approach to tackling air-writing is to first extract the trajectory of the hand or finger of the user, such that classification can be performed on a 2d projection of the trajectory. Figure 2.1 shows examples of hand tracking techniques that are used in practice.

2.1.3 Hand gesture recognition using visible light

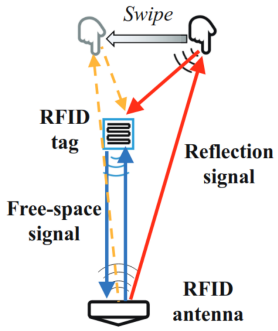
As the primary focus of this thesis is ambient light-based gesture recognition, the following section reviews research performed in this specific area. However, available research on air writing using visible light is limited and only one study was found that specifically targets the air writing of digits.

- In [22], the authors research/study dynamic ambient light-based gesture recognition of air-written digits and sliding gestures. The authors used a 4×3 array of photodiodes,

(a) **Time-of-Flight (TOF) based finger tracking using Microsoft Kinect V2 camera [21]**



(b) **Radar based hand tracking [18]**



(c) **RFID based finger tracking [11]**

Figure 2.1: **Examples of finger tracking techniques.**

which they assume are calibrated to the light condition wherein the system is used. The classification model is a k-Nearest Neighbours with dynamic time warping (DTW) [23] as the distance metric. They classified 8 different sliding gestures with 100% accuracy and digits 0–9 with an accuracy of 82.3%. It should be noted that they assume only a single writing direction and general shape for each of the digits.

- In [24], 7 different static hand gestures were captured using two rows of photodiodes. The top and bottom rows contain 3 and 5 SD5421 photodiodes respectively. The gestures which are considered mimic key presses on a keyboard, with each individual finger pressing down constituting a keypress, as well as an open hand and a fist. The classification was performed using an RNN and resulted in 99.3% accuracy.

- The work done in [25] does not focus on hand gestures specifically, but it is included as they do use ambient light in their proposed system. They use a large 6×6 sensing grid of TSL237 photodiodes that are placed 30 cm apart. Using this setup they measure the influence of the entire body, not just the hands. They define 5 different gestures: punch, hug, clap, step, and jump and collect 15k samples from 22 participants in total. Using an LSTM-based model they report an achieved accuracy of 96.3%.

As can be seen from the summary above, recurrent neural networks (RNN), especially long short-term memory (LSTM), convolutional neural network (CNN), or a combination of the two

architectures are by far the most popular approach to gesture recognition in recent research. This makes sense, as RNNs and CNNs are frequently used with time series data, and dynamic gesture recognition results in time series data. Of course, all except two of the research papers mentioned in this section use TinyML, and the authors use much more powerful hardware to run their models. Therefore, it remains to be seen if recurrent neural networks are a good fit for running on a microcontroller, as they are notorious for being slow, as they contain many complex operations and are not parallelizable. In this respect, convolutional neural networks are more promising as they intrinsically reduce dimensionality, especially when paired with pooling layers.

2.2 TinyML

In recent years, the field of *TinyML* has seen a surge in popularity. TinyML is machine learning on small, low-cost microcontroller units (MCUs), such as IoT devices. Inherently, the devices used for TinyML are much more resource restricted when compared to hardware that is conventionally used to run machine learning models. The power usage typically ranges from a few mW at the low end to several hundred mW at the high end [26]. This stands in stark contrast with hardware commonly used to train large machine learning models, e.g., an Nvidia RTX3090 has a TDP of 350 W. In the past, when machine learning models were required on the edge, the data would be sent to the cloud where powerful hardware runs the computations. TinyML shifts this paradigm to run such models either directly or partially on the device itself [27]. The advantages of this approach are reduced network traffic, latency, and lower cost. Furthermore, it can also improve energy efficiency, privacy, and security [28, 29, 30].

2.2.1 Application research

The field of TinyML is fairly new, however, it has been used successfully in many applications. The following sections will explore recent research focusedd on different applications of TinyML, what kind of models are used and what inference time and accuracy one might expect to attain using such models.

- *Keyword spotting* [31] - This study attempts to classify 12 different keywords from speech signals. They use a convolutional neural network running on the MAX78000, a neural network accelerator specifically designed for CNNs. With this model, they are able to achieve a classification accuracy of 96% with an inference time of only 3.5 ms.
- *Voice recognition* [32] - This study attempts to differentiate adults and children through voice recognition of certain keywords. Using a deep neural network obtained through the Edge Impulse platform and deployed on an Arduino Nano 33 with a Cortex-M4 at 64 MHz, they are able to get an accuracy of 97%.
- *Computer vision* [15] - This study uses a CNN to facilitate the real-time classification of sign language from a video feed. Using an OpenMV H7 development board running ARM Cortex-M7 at 400 MHz they are able to achieve real-time image classification at 20 fps with 75% accuracy.
- *Gesture recognition* [14] - In this study, the authors use a ring worn on the finger of the user which contains an accelerometer. The accelerometer data is fed into an LSTM-based model to classify digits 0-9 written by the user. They design the model to run on an ARM Cortex-M4 at 84 MHz, and are able to achieve accuracy between 75-95% for different gestures.

- *Healthcare* [33] - In this study the authors achieve background noise suppression and speech enhancement in hearing aids by using an LSTM model. They are able to reduce the size of the model by 11.9x through weight quantization and model pruning. Using an ARM Cortex-M7 running at 216 MHz they achieve an inference time of only 2.39ms without a statistical difference in accuracy compared to a larger model.

- *Anomaly detection* [34] - This study focuses on mechanical anomaly detection in household appliances, however, this applies to industrial applications as well. An autoencoder model is designed which is able to classify anomalous vibrations with an accuracy of 92% running on an ARM Cortex M4 at 64 MHz.

Apart from these studies which implement a TinyML system for a specific application, many applications have been proposed in literature, including but not limited to: biometric authentication, object detection, activity recognition, health monitoring and data (pre)processing [26, 35, 36]. The research also proves that it is attainable to run complex models in real-time on a range of low-spec hardware.

2.2.2 Software

Many advances in development tools of TinyML systems have been made over the past few years, improving development flows and enabling the technology to grow [37]. Additional benchmarking and profiling tools have also been developed. [38, 39]. Listed below are several of the most popular libraries and frameworks for developing TinyML systems.

- *Tensorflow Lite Micro* [40] - The most popular open-source framework for neural network models [35], of which the book was released at the end of 2019. This framework is actively maintained by developers at Google and has an active community. It supports many of the most common neural network architectures and layer types and provides examples to get started.

- *GLOW* [41] - A compiler made by NXP to convert models from Open Neural Network Exchange (ONNX) format to low memory footprint format. It supports models generated by the most popular frameworks such as TensorFlow and PyTorch and has a fairly extensive tutorial and documentation.

- *Embedded Learning Library* [42] - A framework created by Microsoft and offers cross-compiling pre-trained models to optimized C++ code which can be deployed on several MCU platforms, such as Arduino and micro:bit.

- *EmbML* [43] - can convert trained models from WEKA or scikit-learn to C++. It has support for MLPs, logistic regression, decision trees and support vector machines (SVM) and attempts to minimize RAM usage and coverts operations to fixed point. However, there are limited documentation and examples available.

- *emlearn* [44] - A software library that allows models to be written in Python using Keras or Scikit-learn and converted to C code. The library supports Decision trees, Random forest, MLPs and Naive Bayes. However, it has limited documentation and examples.

- *CMSIS-NN* [45]: ARM Cortex-M specific highly optimized kernels for Neural Networks and is integrated with TensorFlow Lite Micro. The library makes use of the DSP acceleration capabilities of Cortex M4 and M7 microcontrollers, which can also be used to speed up neural network computations.

- *X-CUBE-AI* [46] - The only proprietary software on this list. It is an AI expansion package specifically for STM32 microcontrollers which can generate optimized code from pre-trained neural network and classical machine learning models. It supports models exported from TensorFlow and other frameworks which can export to the standard ONNX format.

2.2.3 Hardware

A large variety of MCUs exist, and many types have been used in research. However, almost all the chips used are ARM Cortex-M based devices. The ARM Cortex-A line is also mentioned frequently in literature, however these are not considered. These chips are present in the Raspberry Pi and mobile phones (among others), and are deemed to powerful for the scope of this project. Table 2.1 provides an overview of common hardware used for convenience of the reader which has been collected from surveys on TinyML research. This is a general overview, and most of the devices which are listed are general-purpose MCUs, based on ARM Cortex-M architectures. Boards which are similar to each other have been left out. However, the hardware listed in this table may be interchanged with similar from other manufacturers to better match system requirements such as the amount of I/O or peripherals. The exception to this is the ESP32, which uses a Tensilica Xtensa LX6 CPU, the advantage of which is that it includes on-board Wi-Fi and Bluetooth connectivity. The most specialized chip is the MAX78000 by Maxim Integrated (see Figure 2.3b), which contains a 64-core neural network accelerator. It is paired with an ARM Cortex M4 for general computing and interfacing with the neural network chip. This results in high power efficiency and inference which is several orders of magnitude faster than a general-purpose microcontroller [47].

The following sources were used in the compilation of this Table: [27, 35, 48, 49, 50, 47, 51]. As the hardware listed in Table 2.1 is commonly used in TinyML, it is easy to find performance analysis for these chips with various model architectures and datasets. Note that some specialized neural-network accelerator chips which are mentioned in Section 2.2.4 are not included, resulting from either lack of available research or them not being commercially available. Figures 2.2 and 2.3 show examples of commodity and specialized hardware platforms, respectively. Experimental hardware, hardware currently under development, and lesser-known hardware are discussed in Section 2.2.4.

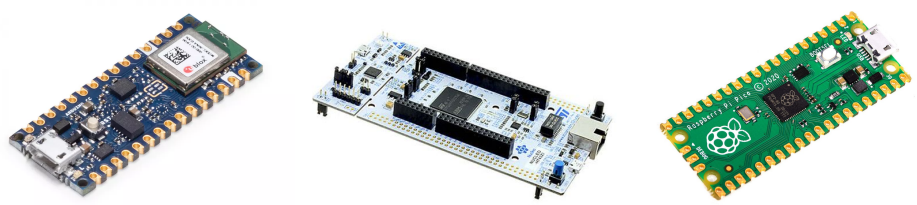
2.2.4 Hardware developments

There are several hardware platforms specially designed for TinyML applications currently under development. These platforms aim to make machine learning more energy efficient and inference time faster since many edge devices are powered by batteries. Unsurprisingly, it shows a trend towards neural network accelerators with a large number of low-power cores, as deep learning models benefit greatly from heavy parallelization. It should be noted that at the time of writing some of these products are not yet commercially available, which is in part due to the recovering chip market after the large shortage during the COVID-19 pandemic.

- *XCORE.AI* [52] - Developed by XMOS, it is a new chip design heavily focused on IoT connectivity in conjunction with machine learning hardware acceleration (see Figure 2.3c).

- *PULP* [53] - Stand for Parallel Ultra-Low-Power and is an open-source CPU design for neural network acceleration developed at ETH Zurich and the University of Bologna. A hardware implementation of this design is the GAP8 by Greenwaves Technologies, an 8-core ultra low-power processor, which is commercially available (see Figure 2.3a).

- *ARM Cortex-M55* [54] - a new ultra-low-power microcontroller featuring new vector extensions (MVE) specifically designed to improve digital signal processing and machine learning capabilities.
- *ARM Ethos U-55* [54] - a neural processing unit designed to work in conjunction with Cortex-M chips, designed to speed up CNN and RNN operations.



(a) **Arduino Nano 33** (b) **STM32H743** (c) **Raspberry Pi Pico**

Figure 2.2: **Examples of general-purpose microcontrollers used for TinyML.**

(a) **GAP8** (b) **MAX 78000** (c) **X-CORE-AI**

Figure 2.3: **Examples of specialized microcontrollers containing neural network accelerators used for TinyML.**

Table 2.1: **Comparison of different hardware used for TinyML.**

Board	Chip	CPU	Clock speed	Flash	SRAM	Hardware acceleration	Board cost	Chip cost	Comments
Arduino Nano 33	nRF52840	ARM Cortex M4	64 MHz	1 MB	256 KB	DSP accelerator including 4×8-bit fixed point SIMD	€25.00	€8.00	On-board WiFi/Bluetooth
Arduino Due	Atmel SAM3x8E	ARM Cortex M3	84 MHz	512 KB	96 KB	No	€40.00	€11.00	No floating point hardware
Sparkfun Edge	Ambiq Apollo3	ARM Cortex M4	96 MHz	1 MB	384 KB	DSP accelerator including 4×8-bit fixed point SIMD	€25.00	€3.50	Highly optimized for low-power
Raspberry Pi Pico	RP 2040	ARM Cortex M0+	133 MHz	16 MB	264 KB	No	€5.00	€1.00	No floating point hardware
MAX78000FTHR	MAX78000	ARM Cortex M4	140 MHz	1 MB	512 KB	Neural network accelerator \w 64 cores and 432KB weight storage	€35.00	€20.00	Highly optimized for CNNs
Espressif ESP32	Tensilica Xtensa LX6	Tensilica Xtensa LX6	240 MHz	4 MB	520 KB	No	€5.00	€2.00	On-board WiFi/Bluetooth
STM32 NUCLEO-H743	STM32H7	ARM Cortex M7	480 MHz	2 MB	1 MB	DSP accelerator including 4×8-bit fixed point SIMD	€30.00	€12.00	6-stage pipeline and 64-bit FPU compared to M4
Teensy 4.1	IMXRT1062DVJ6	ARM Cortex M7	600 MHz	8 MB	1 MB	DSP accelerator including 4×8-bit fixed point SIMD	€35.00	€15.00	6-stage pipeline and 64-bit FPU compared to M4

Chapter 3

System Architecture

This section will provide a high-level rundown of the system μLightDigit, and in the following sections, the design details will be presented. An overview of μLightDigit can be seen in Figure 3.1, with a snapshot of the prototyped system presented in Figure 3.2. The system consists of three main components: the photodiode sensing array, the microcontroller for computing and the tiny deep learning based classification algorithm. An infrared proximity sensor is attached to the photodiode array and is used for the detection of the user's hand. The photodiode array measures the light intensity at different points, converts it to voltage signals and sends them to the microcontroller. The microcontroller samples the voltages using the internal ADC, provides control of all peripherals, and is the computational backbone of μLightDigit. The classification algorithm performs data preprocessing and passes it to the tiny deep learning model, which classifies the air-written digit. μLightDigit can be placed in environments with different ambient light conditions because the system can perform calibration to automatically adjust its light sensitivity level. When placed in a new environment or when the light conditions change, calibration should be performed again for finding the optimal dynamic range for sensing the ambient light.

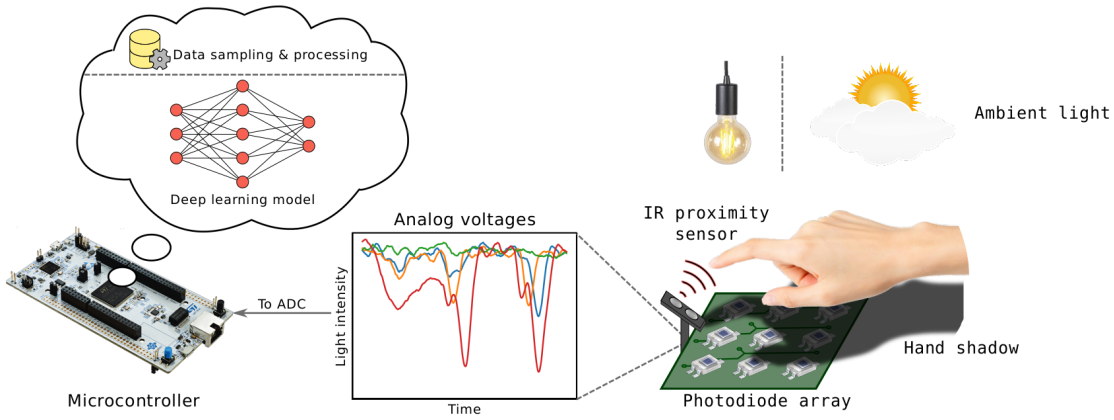
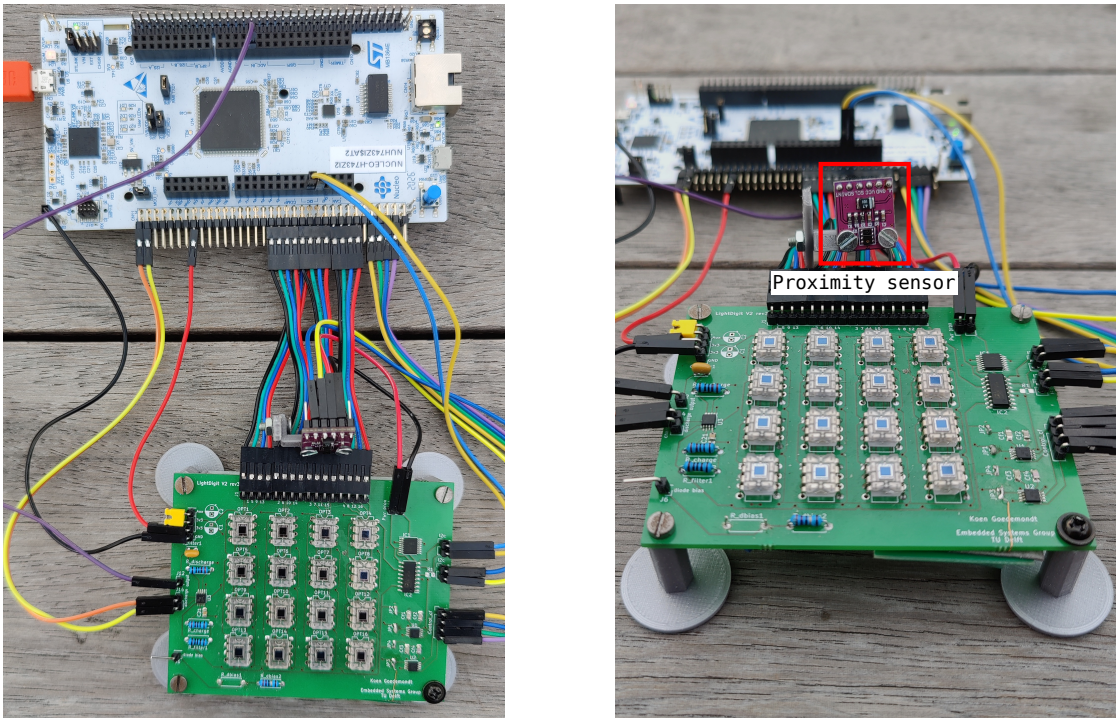


Figure 3.1: **μLightDigit system architecture.**



(a) **Top view of the system.**

(b) **Front view of the system, showing the proximity sensor (purple board).**

Figure 3.2: **The prototype of μLightDigit.**

3.1 Photodiode sensing array

The photodiode array is a 4×4 grid of Texas Instruments OPT101s, which is a photodiode and amplifier combination package. For each classification of the air-written digits, only 9 out of the 16 photodiodes are used, which corresponds to a 3×3 grid in one of the 4 corners of the array. This subset is used to conform to the training data, as it consists of only 9 channels, and gives some flexibility with regard to the position of the hand and surrounding light sources. By measuring which corner of the photodiode array receives the most shadow, this can be chosen dynamically to achieve the best accuracy. This can be helpful, especially in lighting conditions where the light comes in at an angle, causing a shift in the hand shadow. The dimensional layout of the photodiodes on the photodiode array can be seen in Figure 3.3. As the hand of the user moves over the photodiodes, the shadow projected onto them will change. As each photodiode measures the light intensity at a particular location, a particular shadow pattern is captured in every sample. The light intensity is converted to an analog voltage, which is captured and digitized by the microcontroller. The photodiode array from LightDigit is replaced by a new design for μLightDigit, which will be explained in detail in Section 4.3. The primary difference is that in μLightDigit the light sensitivity of the photodiodes is adaptive to a wide range of ambient lighting conditions, making the system be able to work outdoors with sunlight.

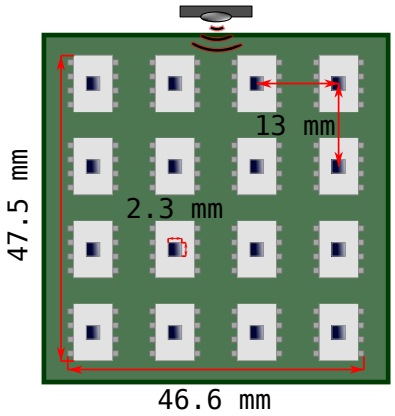


Figure 3.3: **Photodiode array dimensions and spacing.**

3.2 Computing unit

Because of the objective of designing μLightDigit as an embedded system, we choose an STM32 microcontroller as the computing unit. The microcontroller handles all the control of the system such as signal sampling and processing and runs the inference of the tiny deep learning algorithm in real-time . It is responsible for converting the analog signal received from the photodiode array using its internal ADC, preprocessing this data, and feeding it into the tiny deep learning model. Additionally, it responds to interrupts from the proximity sensor –which tells the system when the user starts/stops the air-writing– such that it knows when to stop and start sampling the photodiodes. Finally, it performs calibration to tune the sensitivity of the photodiodes, in order to obtain maximum dynamic range in different ambient lighting intensities.

3.3 Tiny deep learning algorithms

To design a reliable gesture recognition system, one of the most crucial parts is the classification algorithm. In μLightDigit, the designed classification algorithm consists of data preprocessing, photodiode sensitivity calibration and the deep learning model which facilitates the recognition of the air-written digits. Convolutional neural networks (CNN), long short-term memory (LSTM) and a ConvLSTM hybrid architecture are used for the tiny embedded deep learning model. All of these have been successfully employed in related research, and are well-known to be suitable for time-series data processing. As it needs to be able to run in real time on a microcontroller, Chapter 4 will provide the rundown on how to reduce the size of these learning models to match the limited computational resources of the microcontroller.

Chapter 4

System Design

This chapter outlines the hardware and software designs of μLightDigit. The main focus of the hardware design is the processing unit and the design of the photodiode array. The latter has been redesigned compared to the original LightDigit [4] to improve the system's robustness. In the software design, the software framework which is used for designing the deep learning models will be highlighted, and a complete description of the model deployment flow will be given.

4.1 Microcontroller selection

LightDigit used a Raspberry Pi 4 for running the model, while the goal of the TinyML system is to port it to a low-power microcontroller Several considerations were made when choosing the right microcontroller for this project. From preliminary research, it had already become clear that some kind of on-chip hardware acceleration would help realize real-time inference time below 1 second. Additionally, dedicated floating point hardware would be convenient for data preprocessing. This narrows down the search to an ARM Cortex-M4 or Cortex-M7, with the latter being the most powerful chip of the Cortex-M line. Both chips contain ARM DSP extensions which provide single instruction multiple data (SIMD) instructions for the acceleration of certain operations which are useful in neural network computations.

The Arduino Nano 33 was considered, which has an ARM Cortex-M4 based CPU. However, it was dismissed in favor of a Cortex-M7 based device, as it has a flexible clock with a higher possible clock speed. This in turn makes the development flow more forgiving by allowing more inefficiencies and allows for easier experimenting with larger model sizes and different architectures. Eventually, the choice was settled on the STM32NUCLEO-H743ZI2 development board, which features an ARM Cortex-M7 running at a maximum of 480 MHz, and has a generous amount of I/O because of the 144-pin package. It is at the high end of microcontrollers because of its relatively high clock speed and contains a dedicated floating point unit (FPU). However, it is by no means a match for a Raspberry Pi 4 which has an order of magnitude higher computing ability. Table 4.1 can be used as a reference for a rough hardware comparison between the LightDigit and μLightDigit systems.

It was decided to use a general-purpose microcontroller, as opposed to a more specialized chip that includes dedicated neural network accelerators. This allows for inter-device compatibility if the model should be ported to a different device. Furthermore, it does not require the use of proprietary software or specialized libraries which can onlfy be used on one specific chip.

Table 4.1: **Comparison of specs between the Raspberry Pi 4 and STM32H743.**

Specification	**Raspberry Pi 4**	**STM32H743**
CPU	ARM Cortex-A72 (Quad-core)	ARM Cortex-M7 (Single-core)
Clock speed	1.5 GHz	480 MHz
RAM	4 GB	1 MB
Storage	>GBs	2 MB
Peak power	6.4 W	0.45 W

4.2 Software development flow

To implement deep learning models, a well-supported and rich software framework is required. This avoids the time-consuming, error-prone and tedious process of implementing neural networks by hand. The following considerations are made before choosing a framework:

- The framework must be compatible with ARM Cortex-M devices.
- The most essential neural network layers should be supported, such as dense layers, convolutional layers, LSTM layers, and pooling layers.
- There should be ample documentation and educational material for learning the framework.
- For development speed and convenience, network architectures should be allowed to be written in a high-level programming language such as Python.
- The framework should be focusedd on microcontrollers and have included support for memory and run-time optimizations such as automatic quantization.

4.2.1 TensorFlow Lite for Microcontrollers

The only framework which fulfills these requirements is TensorFlow Lite for Microcontrollers [55], which often gets abbreviated to TFLM, TFLu, or TFLμ (in this thesis TFLM is used to remain consistent with the developers). The project is based on the already popular and widely used TensorFlow, which is already a mature, popular and very well documented deep learning framework. It allows models to be written in Python using TensorFlow, which in turn is converted to an optimized format using the TFLite converter, creating a TFLite model. The TFLite model in combination with the TFLM runtime and operation kernels allows the model to run efficiently on a microcontroller. Operation kernels provide the most low-level computations which are chained together to execute all operations in a neural network layer.

In contrast to some frameworks, it does not attempt to transpile source code to run on a microcontroller, but instead, it has ported TensorFlow operation kernels to C++ targeted at microcontrollers. This means there are no C/C++ standard libraries, no dynamic memory allocation and the core runtime fits in 16 kB RAM. It also includes model optimization techniques out of the box and it is relatively easy to port models between different devices, as many popular 32-bit microcontrollers are supported. An added benefit of TFLM is that the model can be run and verified on the host computer, which already includes automatic parameter quantization. This eliminates the need to transfer each model individually to the microcontroller before being able to test the accuracy. For model hyperparameter tuning, keras-tuner [56] is used in conjunction with the HyperBand algorithm [57]. A complete overview of the creation and deployment of a model can be seen in Figure 4.1.

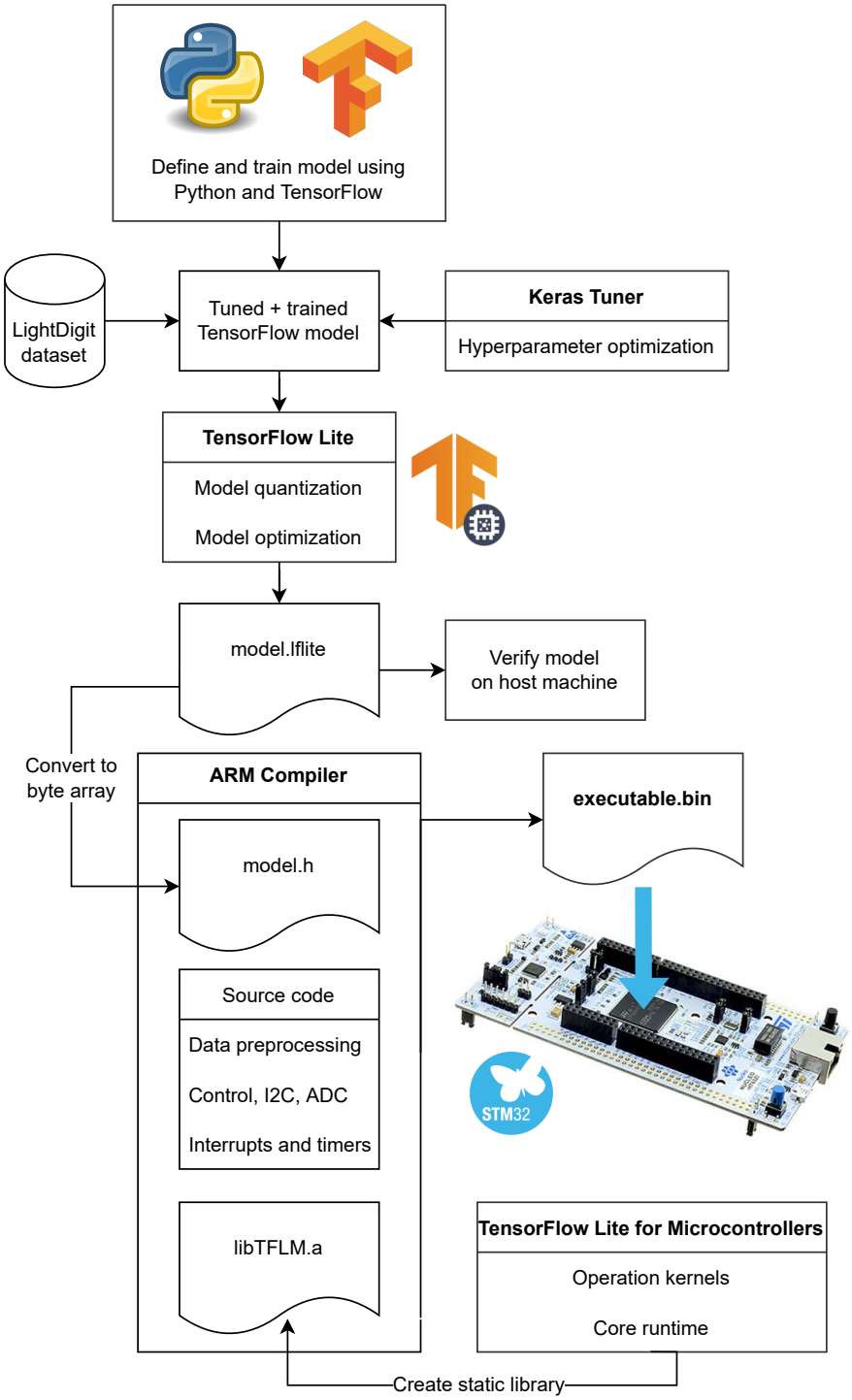


Figure 4.1: **Model development flow using TensorFlow Lite for Microcontrollers.**

4.2.2 Model Quantization

TFLM provides automatic post-training quantization of model weights to 8-bit integers. This can be achieved by providing the TFLite converter with a generator function that provides a representative sample from the training data. The data samples are used to scale the weights internally, such that the dynamic range is optimized.

4.2.3 Limitations

On-device training is not supported, which means all models must be trained on a separate host computer. An important step to take is to fix the input and output size of the model which is created. This needs to be done to avoid the TFLite converter from inserting flexible tensor operations, which are (at the time of writing this thesis) not supported in TFLM. STM32H743 is not directly supported with a makefile, therefore, this needs to be developed as well. Cmake is used to generate the makefile.

4.2.4 CMSIS-NN

As briefly discussed in Section 4.1, the Cortex-M7 has special instructions to speed up certain computations which are useful for digital signal processing. It drastically speeds up 8-bit integer arithmetic by using SIMD operations. Besides, they can also be used for neural network acceleration. CMSIS-NN is the library developed by Arm aiming at neural network acceleration, and is directly integrated into TFLM. It can be turned on by simply adding a compiler flag. As will be seen in later chapters, it works especially well for convolutional neural network acceleration.

4.3 Photodiode array redesign

While testing using the first version of the photodiode array, it was observed that the output of the OPT101 would saturate when the ambient light intensity exceeded approximately 500 lux. Practically, this means that in bright conditions the photodiodes quickly saturate. This has a great impact on shadow detection capability in many indoor and almost all outdoor environments, since they are often much brighter than this. As an example, in the measured light intensity in the office where many of these experiments were performed, the brightness is already 500 lux due to the overhead lighting, without considering sunlight entering through the window. Comparatively, on sunny days this can easily reach 1k lux indoors and upwards of 50k lux outdoors. As a result, using the photodiode array in such environments could result in inaccurate data, because some shadows are not correctly detected. To better illustrate the problem, let us consider the scenario in Figure 4.2. The light sensor is partially covered, therefore, one would expect to see a drop in voltage at the output. Saturation happens when the average light intensity on the sensor would cause a voltage at the output of OPT101 which it is unable to provide. This causes the output to be stuck at the maximum output voltage, which is determined by the supply voltage. Thus, even though the sensor should detect a shadow, it is not able to do so as the output is saturated.

4.3.1 Improving the dynamic range of photodiodes

In order to understand the details of exactly why saturation occurs, the inner workings of the OPT101 must be considered. Figure 4.3 shows the schematic of the internal architecture which was taken from the datasheet [58]. As can be seen from Figure 4.3, the OPT101 package combines a photodiode, a transimpedance amplifier and a feedback network. Because photodiodes are

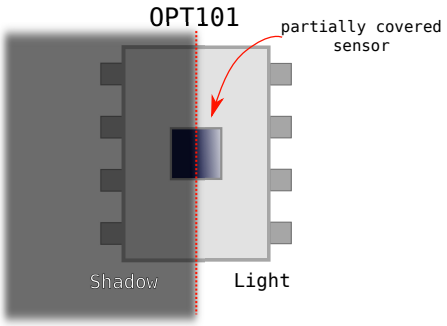


Figure 4.2: **Top view of an OPT101 with a shadow partially covering its light sensor. The dashed red line indicates the shadow boundary.**

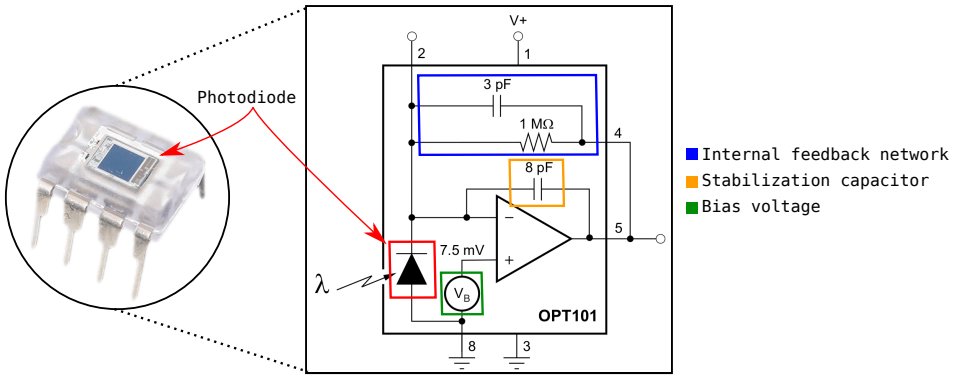


Figure 4.3: **OPT101 with internal schematic and notable parts highlighted. Notice that the internal feedback network (pin 4) is connected to the output (pin 5).**

current sources, the transimpedance amplifier converts the photodiode current into a usable voltage, which in turn can be sampled by an ADC. The output voltage is linear with respect to the illuminance of the photodiode, until it is limited by the positive power supply rail. With the specific opamp used in the OPT101, the maximum output that can be reached is the positive power supply - 0.8 V. When this happens, the output is considered saturated as it can no longer increase. The supply voltage can range from 2.7 V to 36 V, thus opting for a higher supply voltage is one way of improving the dynamic range of the output.

Another way of improving the dynamic range is by tuning the gain of the amplifier. In a transimpedance amplifier, the gain is equivalent to the feedback resistor value. Therefore, the feedback resistor determines the sensitivity of the output. The OPT101 includes an internal feedback network with a 1 MΩ resistor and 3 pF capacitor. The capacitor is required to maintain the stability of the internal opamp due to the capacitance of the photodiode [59], in turn also limiting the bandwidth of the amplifier. The combination of a low supply voltage and high sensitivity results in the internal opamp quickly reaching the positive power supply rail and saturating. Therefore, the dynamic range can be improved either through increasing the supply voltage and/or changing the feedback resistor value. To accommodate this, the internal feedback network can be disconnected and a custom feedback network can be used.

4.3.2 Previous design

The photodiode array designed for LightDigit uses a 4×4 grid of OPT101 photodiodes, with the centers of the sensors placed exactly 13 mm apart. Each of the OPT101 uses the internal 1 MΩ feedback network, resulting in a fixed light sensitivity. The outputs are sampled using 2 external 10-bit, 8-channel MCP3008 ADCs which are addressed through an SPI interface. The supply voltage is 5 V, resulting in the output of the OPT101s reaching approximately 4.2 V.

4.3.3 Requirements

The biggest challenge will be to improve upon the old design by adding adaptability to different ambient light intensities and at the same time keeping the new design compatible with the already existing dataset. In addition, more requirements are added for practical considerations and to keep costs down. The design requirements are formulated as follows:

- The photodiodes' position and the photosensitive area should be the same as the old design.
- The system should be able to operate at least from 100 lux to 10k lux, covering practically all indoor scenarios.
- The array should be powered from the same supply as the microcontroller.
- The cost of the design must be as low as possible.
- The photodiodes should be sampled at a high enough frequency such that there is at most 20% distortion in each retrieved sample and model accuracy is maintained.

4.3.4 Design

When coming up with the new design, it was first considered to use a new type of photodiode with an external amplifier circuit. This would be able to relieve some spacing constraints which the OPT101 package size brings. The OPT101 uses a DIP8 package, which is quite large relative to the photosensitive area. This leaves a space of only 3 mm between sensors, which is not large enough for placing most component packages. However, in order to avoid the need to investigate the impact of using differently spaced photodiodes or using photodiodes with a different size photosensitive area, it was decided to keep using the OPT101 as the photodetector. As a result, the main goal of the new design became to replace the internal feedback network with a variable resistor, which can be changed depending on the ambient light intensity. The most flexible way to achieve this is through using a digital potentiometer. It has a wide range of programmable resistor values and can be rapidly interacted with through some serial interface. However, digital potentiometers with values in the mega ohms are fairly uncommon and relatively expensive at around €5 each. Due to this restriction, combined with the limited spacing between the photodiodes, it was investigated if it would be possible to use a single feedback network for all OPT101s. The obvious advantage would be that only a single digital potentiometer and stabilization capacitor would be necessary for all photodiodes. The main question following this was whether the response time of the OPT101s would be fast enough to switch between them while keeping the distortion at an acceptable level. In the datasheet the settling time to reach 1% of the final output is said to be 70 μs. For the entire array, this would be x16, resulting in 1.120 ms for the total array. At first glance this seems like a sufficiently short sampling time given the user does not move their hand extremely quickly. From this initial evidence of feasibility, the design as shown in Figure 4.4 was created.

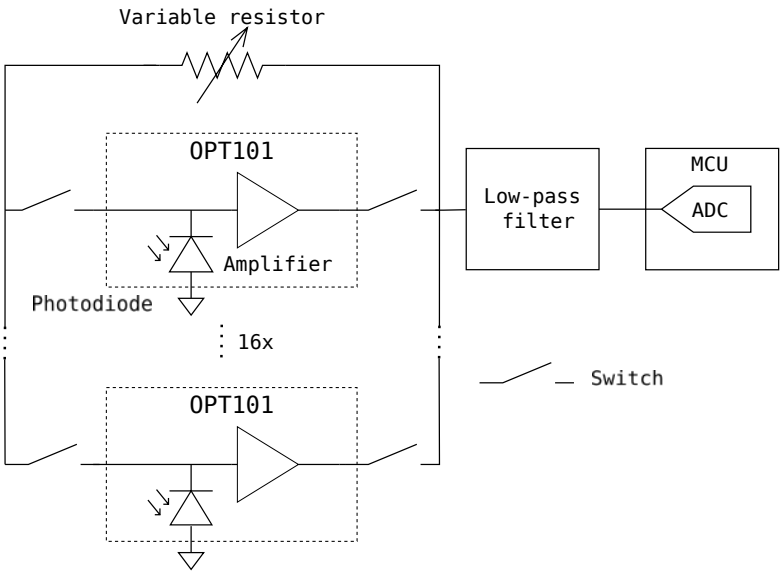


Figure 4.4: **Basic schematic showing the design principle of the new photodiode array.**

The final design is the result of several iterations, using simulations and creating several prototypes. It was made to be modular and robust, allowing the ability to leave out certain parts of the design while maintaining a functioning system. Compared to the old design, the feedback network was redesigned, an output low-pass filter was added and the external ADCs were removed in favor of the MCUs internal ADC. An external proximity sensor was added to detect the presence of the hand of the user. Moreover, supporting circuitry was required to make the design work. First, an overview of the system can be seen in Figure 4.5 and a component list in Table 4.2. The sections after that will discuss each different part of the design in more detail. Additionally, the simulating and testing of the circuit will be explained.

Components

The components used are listed in Table 4.2.

Table 4.2: **Components used in the new photodiode array design**

Component	Manufacturer	Function	Amount	Price	Total price
OPT101	Texas Instruments	Light detection	16	€2,00	€32,00
AD5242BRZ1M	Analog devices	Digital potentiometer	1	€5,00	€5,00
AD5242BRZ100	Analog devices	Digital potentiometer	1	€5,00	€5,00
TMUX1121	Texas Instruments	Precision analog switch	3	€1,75	€5,25
TS3A24157	Texas Instruments	SPDT analog switch	16	€0,90	€14,40
RB531VM-40FH	ROHM	Schottky diode	16	€0,05	€0,80
Generic resistors	-	-	5	€0,01	€0,05
Generic capacitors	-	-	6	€0,01	€0,06
				Total	€62,56

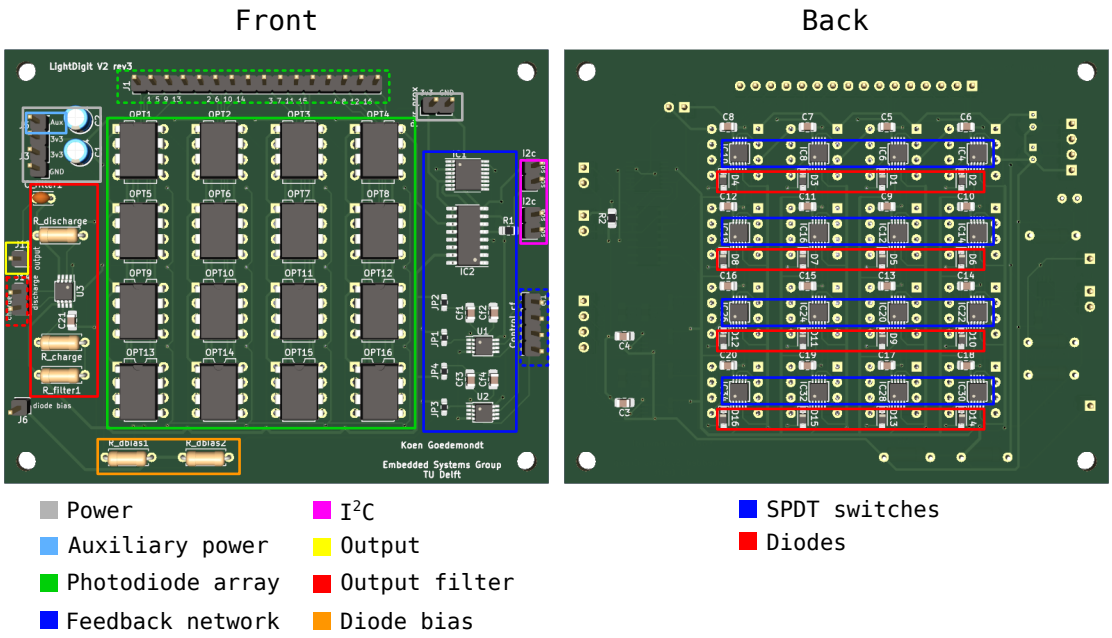


Figure 4.5: **Annotated PCB front and back.**

Overview

An overview of the final designed PCB can be seen in Figure 4.5. All the design files and schematic can be found in Appendix A. Each of the separate modules is explained in further detail in the following sections.

Power supply

The power supply for the general electronics is 3.3 V. The OPT101s are connected to a secondary auxiliary power supply which can either be connected to 3.3 V, or connected through a separate power supply which can be increased to 4.1 V. This allows for a slightly higher output voltage range, thus increasing the dynamic range further. Increasing beyond this point is not possible with the internal ADC of the microcontroller, as the reference voltage for the ADC can be at most 3.3 V. However, it could be possible to add circuitry to the output which scales the voltage back to 3.3 V range, which would allow for a supply voltage of around 5 V. Increasing it further then this could potentially damage the switching electronics, as the maximum allowed voltage at their inputs is Vdd + 0.5 V.

Switches and diodes

Each of the OPT101s is controlled by a TS3A24157, a 2-channel single-pole dual-throw switch. The switch isolates both the internal photodiode and the OPT101 output from the feedback network. The switches are controlled directly through GPIO pins on the microcontroller, and allow the OPT101s to be individually activated and deactivated. When an OPT101 is activated, it is connected to the output and the feedback network. In a deactivated state, a Schottky diode is connected to the internal photodiode and keeps the voltage at the inverting input at a low (around 50 mV) positive voltage. At first glance, this might seem unnecessary, however

the CMOS switches inside the TS3A24157 start conducting when a negative voltage is applied. Secondly, this keeps the output of each of the deactivated OPT101s low, avoiding leakage current through the switches. The diode is biased through the diode bias network, which is either tunable with a potentiometer or can be applied using an external power supply.

Feedback network

The feedback network consists of two digital potentiometers combined with 4 activatable feedback capacitors. Each of the digipots have 255 step RDACs for resistance adjustment and can be programmed using I^2C. The first one is the AD5242BRZ1M - a 2-channel 1 MΩ resistor, the 2-channels are connected which allows for course feedback resistance adjustments up to 2 MΩ. Similarly, the second one is the AD5242BRZ100 - a 2-channel 100 kΩ resistor, the 2-channels are connected which allows for fine feedback resistance adjustments up to 200 kΩ. The activatable feedback capacitors can be switched on or off using a TMUX1121 precision analog switch. The capacitance values are 10, 22, 30 and 47 pF and have been chosen such that the feedback resistor value that can be used lies between 20 kΩ and 2 MΩ. Additionally, the fact that they are switchable ensures the system bandwidth is high enough such that switching delays do not exceed the desired time.

Output filter

A low-pass filter is added to the output to filter small output perturbations which were observed on the output of the OPT101 during testing. To decrease the possible cutoff frequency to the desired value, circuitry was added which is able to more quickly drain and charge the filter capacitor. This essentially provides the output filter with 2 separate cutoff frequencies, one to facilitate faster switching and the other to facilitate noise filtering. The cutoff frequency can be calculated using Equation 4.1 and is directly related to the RC (τ) time. The 5 τ is considered, which is the time it takes to reach 1% of the final voltage. The output low pass filter uses a resistance of 1 kΩ and capacitance of 22 nF, resulting in $f_{cutoff} = 7.2kHz$ and $5\tau = 110\mu s$. On the other hand, the secondary resistors have a resistance of 200 Ω, reducing 5 τ to 22 μs and allowing for a faster response time at the output. This allows flexibility in choosing the primary filter cutoff frequency, as the output response time is not limited by the filter cutoff frequency. For example, if the cutoff frequency of 7.2 kHz is found to be too high, it can be further reduced by placing a higher resistor and/or lower capacitor value. A secondary resistance value of 200 Ω was chosen to limit the maximum possible output current, as the opamp can provide a maximum of 15 mA.

$$f_c = \frac{1}{2\pi R_f C_f} \tag{4.1}$$

Proximity sensor

To facilitate the detection of the hand of the user, support for adding an APDS-9930 infrared proximity sensor was added to the design. It can be attached to the PCB using a 3D-printed mounting bracket and can be adjusted in 3 dimensions and inclination angle. It can be programmed using I^2C and is connected to the same bus as the feedback network. The sensor will generate an interrupt when it detects the hand of the user, which in turn triggers sampling to start. When to user removed the hand, a second interrupt is generated to stop the sampling. For convenience, dedicated power pins and I^2C pins are added to the PCB for connecting the proximity sensor. It is connected to the same I^2C bus as the feedback resistors.

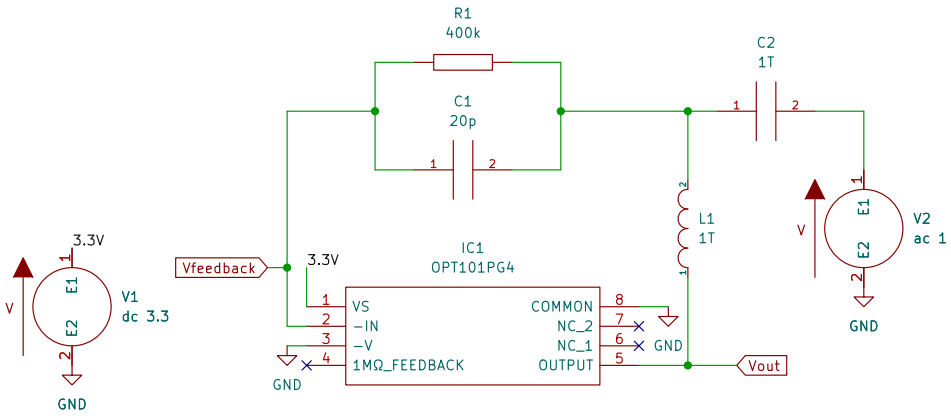


Figure 4.6: **OPT101 stability simulation setup. The phase margin is determined by examining the phase shift caused by the loop gain.**

4.3.5 Stability analysis

Using an inappropriate feedback network or introducing stray capacitance in an opamp design might cause stability issues. This can cause them to oscillate, ring or result in other kinds of unwanted behavior such as lockup (i.e., the output gets stuck on the positive or negative power rails). This is relevant since the OPT101 contains an internal opamp which might require additional compensation when using an external feedback network. In Section 4.3.1, it was already briefly stated that the capacitor in the feedback network is needed to ensure stable operation. To elaborate on this further, the capacitor is necessary to compensate for the phase shift caused by the capacitance of the photodiode at the inverting input. At 180° phase shift between the inverting and non-inverting inputs, negative feedback becomes positive feedback and the circuit is unstable. Phase margin indicates the amount of phase shift required to reach this critical point of 180° phase shift and can be seen as a safety margin before the circuit becomes unstable. In industry, the rule of thumb is to have at least 45° of phase margin to ensure the stability of an opamp circuit [60]. Having a lower phase margin will result in ringing and overshoot. As detailed in [59], Equation 4.2 is used to calculate the required feedback capacitor value to ensure a phase margin of 45°.

$$C_f = \sqrt{\frac{C_{in} + C_{pd}}{2\pi R_f F_{GBW}}} \tag{4.2}$$

C_{in} is the stray parasitic capacitance from surrounding components and opamp inputs. The most significant contributor to this capacitance is the switches, which have up to 140 pF stray input capacitance in on-state and 50 pF stray input capacitance in off-state. Both the capacitance at the input and output can be calculated as $15 \times 50nF + 140nF = 890nF$, which is modeled as 2 nF at the input, and 2 nF at the output. This gives a safety margin for the PCB traces and for other components which will be placed. C_{pd} is the photodiode capacitance, which is caused by the depletion region in the PN-junction of the photodiode. R_f is the feedback resistance and F_{GBW} is the gain-bandwidth product of the opamp, which is 2 MHz. In order to verify the stability of the final circuit, SPICE simulations were performed using Ngspice [61] and the SPICE model provided by Texas Instruments, which can be found in Appendix B.2. The phase margin is determined by breaking the feedback loop of the opamp and inserting a very large capacitor and inductor as described in [62] and shown in Figure 4.6. This allows SPICE to find

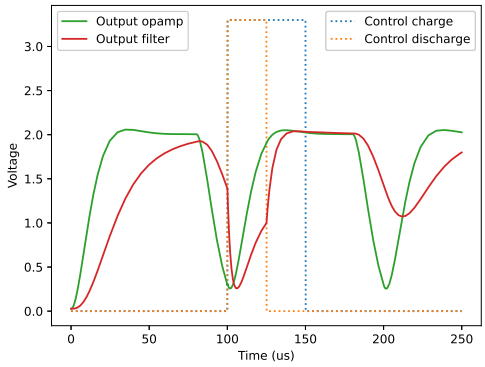


Figure 4.7: **Switching and output filter simulation using R_f=400 kΩ and C_f=20 pF.**

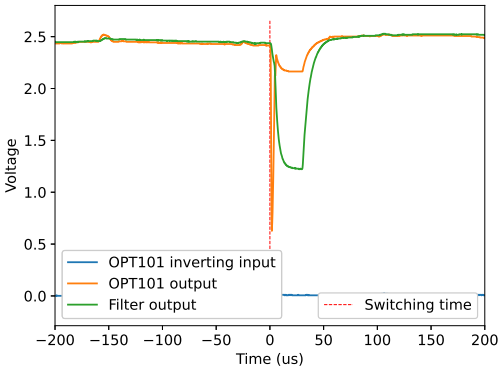


Figure 4.8: **Measured output with R_f=480 kΩ and C_f=20 pF.**

the correct DC operating points for performing the AC analysis. Using this circuit, and the values obtained through Equation 4.2, the stability of the circuit was verified.

4.3.6 Simulations

To further examine switching behavior and to ensure the proper functioning of the output filter, transient SPICE simulations are performed using the OPT101 SPICE model. The photodiode current is modeled using the current source `I_photodiode`, and the switching behavior is modeled as a pulsed current source. The setup for the OPT101 simulation with feedback network and the output filter can be found in Appendix B. Double push buttons are used to model the analog switches, which are controlled using control voltages `v_control_charge` and `v_control_discharge`. Figure 4.7 shows 3 simulated pulses, with the output filter being activated on the second pulse. After activation, both switches are closed for 25 μs, moving the output of the filter to V(output opamp)/2. Then, the discharge switch is closed and the filter output moves to a stable voltage after approximately 15 μs. For comparison Figure 4.8 shows actual measured switching behavior in roughly the same conditions.

4.3.7 Prototype and PCB

To verify the simulation results, a breadboard and soldered prototype were created to test the design concept, and can be seen in Figure 4.9. The parasitic capacitances from surrounding electronics are modeled as 2.2 nF capacitors, which are placed at the inverting input and output to ground. Additionally, the diode to compensate for the negative voltage at the inverting input was verified in the prototype. The diode was biased using an external power supply and kept the voltage of the inverting input at approximately 30 mV. This is low enough for the output of the opamp to remain at 0, limiting leakage currents and giving a fast response time. From measurement on the prototype, using a potentiometer as a feedback resistor, it was found that resistor values below 15 kΩ resulted in oscillations on the output signal. Therefore, the internal opamp can be considered unstable at low gains, which should be taken into account when using the system. For reference, the final PCB design files and schematics can be found in Appendix A. The electrical schematics and PCB were designed using KiCad 6 and manufactured by Aisler.net.

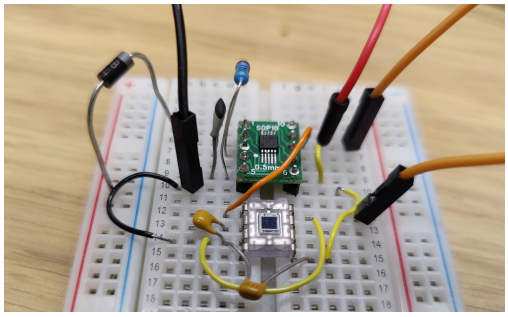
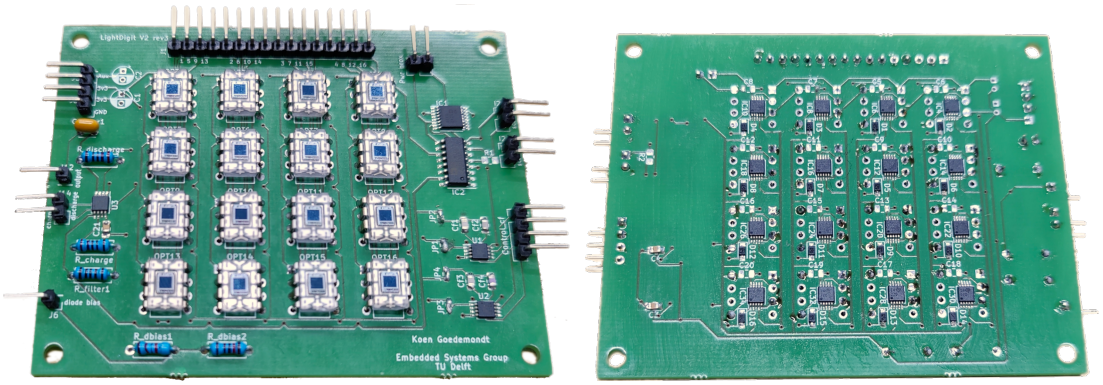


Figure 4.9: **Prototype on a breadboard with a single OPT101, switches and diode. Parasitic capacitances are modeled as 2.2 nF capacitors at the inverting input and the output.**



(a) **Front of the assembled PCB.** (b) **Back of the assembled PCB.**

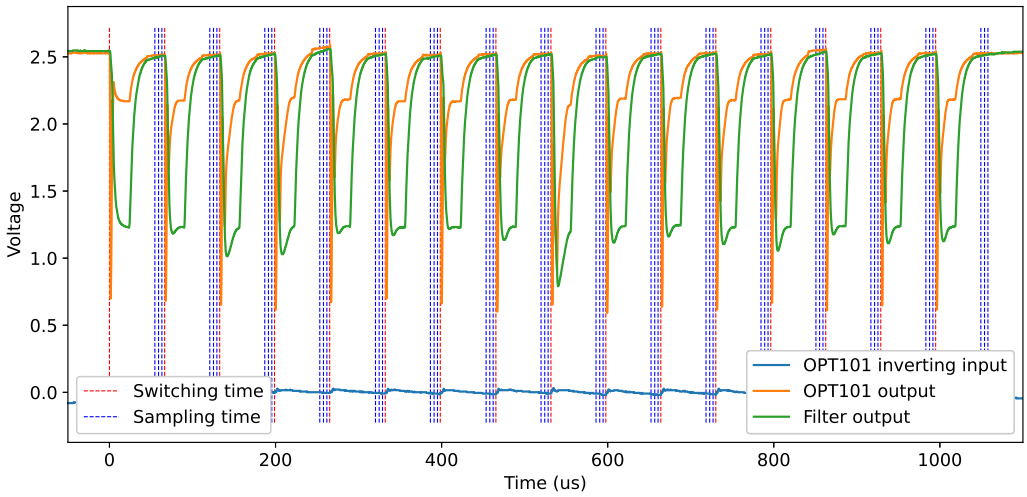
Figure 4.10: **Assembled PCB.**

4.3.8 Results

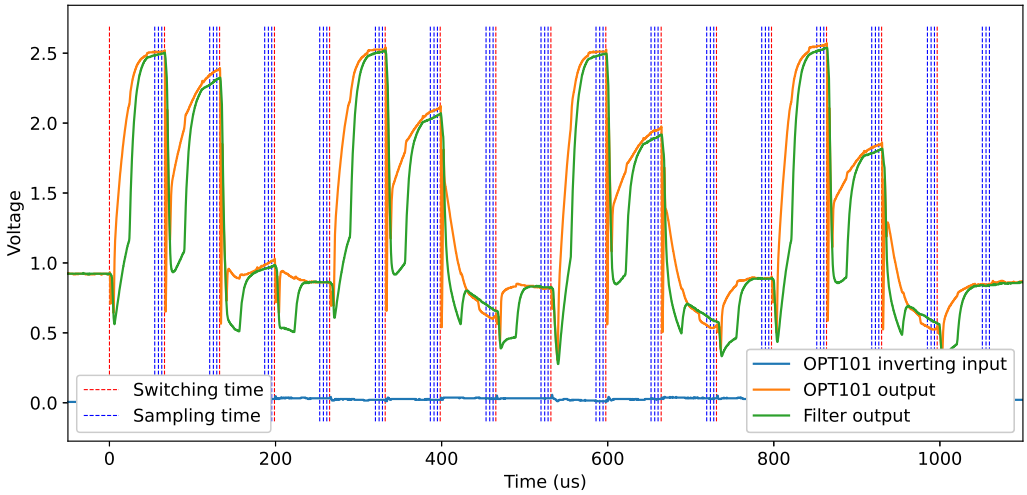
The final assembled PCB can be seen in Figure 4.10. The resulting circuit is highly adaptable to ambient light intensity but has some drawbacks. Figure 4.11 shows 2 full samples of the photodiode array, sampling all 16 photodiodes within 1.05 ms and a subset of 9 photometries within 0.6 ms. The array can handle light intensities up to 15k lux, but beyond that saturation becomes a problem, even with the lowest possible feedback resistance. As the opamp is not unity-gain stable, a minimum feedback resistance of 15 kΩ is required for stability. High-to-low transitioning is occasionally paired with a dip in the signal from the output opamp. It is not quite clear what causes this, but it was observed that the voltage eventually settles close to the average voltage resulting from the ADC samples. Therefore, it is not considered a problem as it does not cause signification variations in the output.

Response time issues

During testing, it was observed that a combination of a positive voltage at the inverting input and the input capacitance could cause a delay in response time. Figure 4.12a shows a measurement that illustrates this problem. It can be seen that it takes some time for the inverting input voltage to reach a low enough value for the opamp to respond and the output to rise. This is due



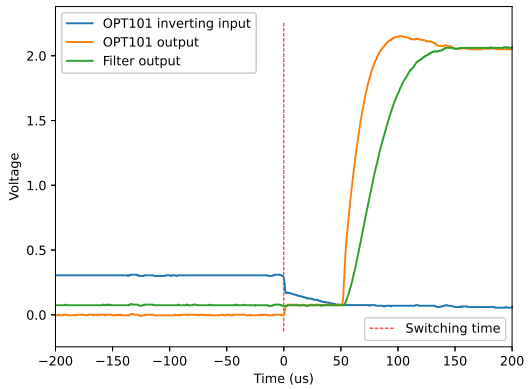
(a) **Full sample with the photodiode array uncovered.**



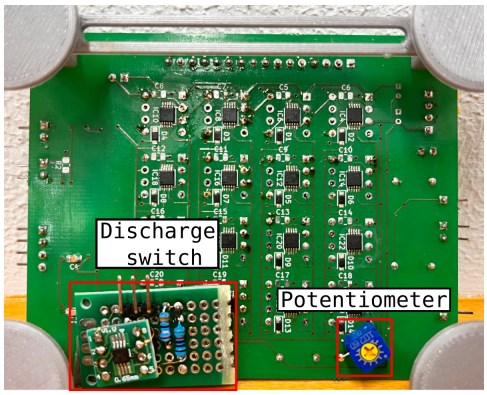
(b) **Full sample with the right side of the photodiode array covered.**

Figure 4.11: **Measured output of the photodiode array over one full sample. In this case, the output is sampled 3 times by the ADC, which is indicated by the vertical blue lines. Switching between photodiodes is indicated by the vertical red lines.**

to the input capacitance needing some time to discharge. As the only discharge path is through the generally high-valued feedback resistor, this time can be significant compared to the total sampling time. In order to mitigate this problem, some additional circuitry was added to the bottom of the PCB. First, a switch controlled through GPIO pins is connected to a relatively small resistor value of 1 kΩ, which can be opened to drain the opamp input capacitance more quickly. Secondly, a potentiometer is placed instead of a static resistor in the diode bias network and can be used to manually tweak the diode bias voltage to keep it at a sufficiently low value.



(a) **Response time delay caused by positive voltage at the inverting terminal of the OPT101.**



(b) **Adhoc additions to PCB for response time improvement**

Figure 4.12: **Response time issues caused by positive voltage at the inverting input.**

Chapter 5

Tiny Deep Learning Algorithm

The classification algorithm is made up of two main components: data preprocessing and the embedded deep learning model. Data preprocessing is required to transform the raw ADC values into data that the deep learning model can use. First, a deep analysis of the data is performed, which explains how the data can be interpreted visually and how movement is encoded in the input data. Additionally, the concept of sampling delay distortion is explained. Secondly, three different deep learning models are designed: a convolutional neural network (CNN), a long short-term memory (LSTM) model, and a ConvLSTM hybrid model, which uses elements from the CNN and LSTM. The architecture of each of the models is explained and visualized. Finally, the photodiode calibration algorithm is discussed

5.1 Data interpretation

The data is received as a 1×9 array of numbers, one number for each channel which is used for sensing. This data can be visualized as a 3×3 grid, analogous to the layout of the physical photodiodes. Figure 5.1 shows an example of how data can be interpreted visually. Numbers are normalized from 0 to 1, representing a low light intensity and a high light intensity respectively. The dark pixels roughly indicate the location of the fingertip, which traces a trajectory over time. To represent an entire sample in a single 2D array, the raw 1×9 array of measurements can be

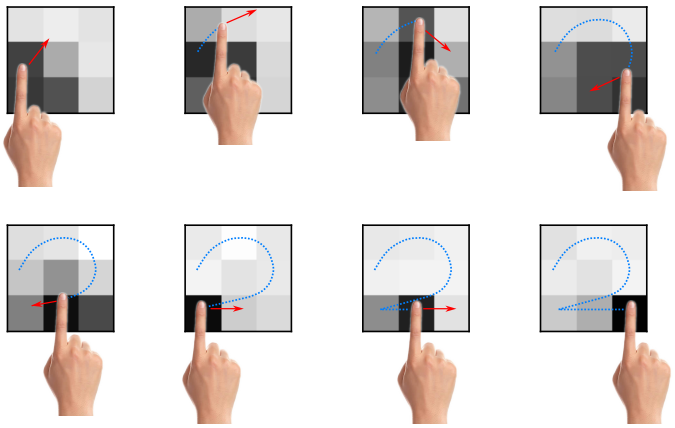


Figure 5.1: **Example drawing of the number 2.**

stacked vertically. This results in particular patterns, which can be seen in Figure 5.2. These patterns contain movement information of the users' hand, encoded in the particular pattern which is formed. This is explained in detail in Section 5.2.4.

5.2 Data processing

The data processing is performed on the microcontroller and consists of 3 main steps. Figure 5.3 shows the different steps performed in the data processing. Before being sampled and converted into 16-bit values, the analog low-pass filter on the photodiode array removes high-frequency noise and data spikes. Doing this in hardware saves processing time and reduces the complexity of the data preprocessing algorithm.

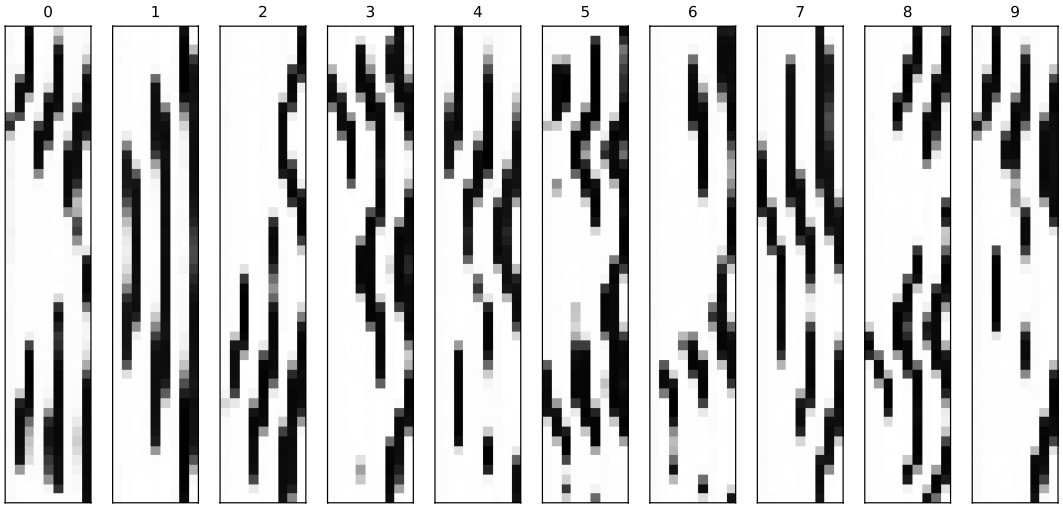


Figure 5.2: **Image patterns created by stacking samples.**

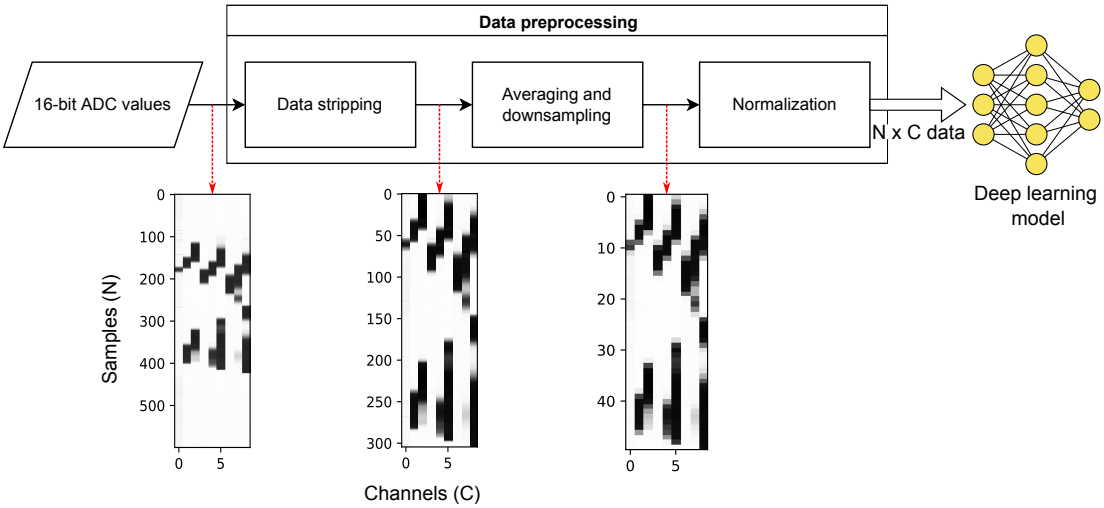


Figure 5.3: **Data processing pipeline.**

5.2.1 Data collection

The input data to the microcontroller are analog voltages and are digitized using the internal 16-bit ADC with a 3.3 V reference voltage. A single-channel internal ADC is used, and each photodiode is sampled in quick succession to get a single sample. Depending on the input dimensions of the model, the photodiodes which are sampled are adjusted. For example, given a model which requires only 9 input channels, not all 16 photodiodes need to be sampled, but only the corresponding 9. Data collection is activated through the proximity sensor, or manually by pressing a button. When manual data collection is activated, the default sampling time is 3 seconds at 200 Hz, collecting a total of 600 samples. On the other hand, when data collection is activated through the proximity sensor the sampling time is dynamic and depends on how long the proximity sensor remains active. The maximum dynamic sampling time is 10 seconds. If this is exceeded the system will output an error and the sampling should be restarted.

5.2.2 Data stripping

The first step of the actual data preprocessing is stripping "Empty" data from the beginning and end of the collected signals. This data is introduced when sampling of the photodiodes starts too early or ends too late. As can be seen in Figure 5.3, this data does not capture any shadow and therefore contains no useful information. Stripping is done by calculating the variance of each sample, normalizing the variance to a value between 0 and 1, and removing samples that lie below a certain threshold. The threshold is determined by a method similar to the elbow method used in clustering [63]. A large amount of samples is taken from the training and test set, and data is stripped using thresholds ranging from 0 to 1. The average amount of samples left over after stripping is plotted against the threshold, and the inflection point is found in this data. Figure 5.4 shows this plot and the inflection point at a threshold of 0.20. A second, more sophisticated data stripping method was tried which also considered the variance across several samples by splitting the data into windows. However, this did not result in additional gains; so the simpler method was used.

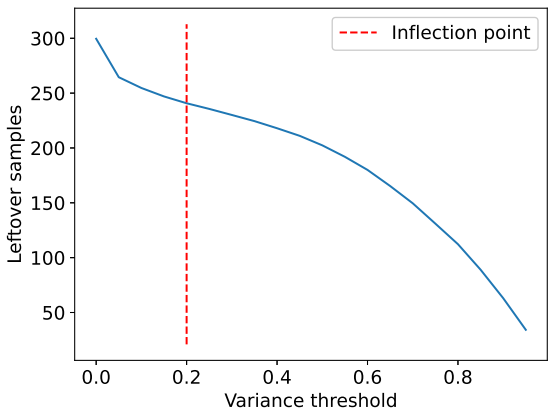


Figure 5.4: **Leftover samples after stripping vs. variance threshold with a highlighted inflection point.**

5.2.3 Data downsampling and normalization

The model expects input data with fixed dimensions. However, during the data preprocessing dimensions can vary, because data is either not sampled for a fixed amount of time due to the proximity sensor or a certain amount of samples is stripped from the data. Assume the input dimensions of the model are $N \times C$, with C the number of channels and N the number of samples. The original data contains an arbitrary amount of samples X, so to conform to the model input dimensions the data is downsampled to a fixed length N. To achieve this, all samples are split into groups of size floor(len(X)/N). If the remainder X mod $N \neq 0$, the excess samples are split randomly across groups, i.e. some randomly chosen groups will increase in size. Then, all samples in each group are averaged, resulting in data of dimensions $N \times C$. The impact of different data lengths is investigated in more detail, while also considering model accuracy loss in Section 6.1.3.

5.2.4 Channel ordering and information encoding

As the user moves their hand over the photodiodes and casts a shadow, a pattern is formed when this data is shown as a 2D image. From this pattern, movements can be extracted by examining how the shadow transitions over the different channels. Different patterns are created when the channels are placed in particular orders and encode movement information differently. To demonstrate the encoding of movement in the image patterns, three channel orderings are shown in Figure 5.5. Horizontal channel ordering, meaning all horizontal channels side by side. Vertical channel ordering, meaning all vertical channels are stacked side by side. Diagonal channel ordering, meaning the diagonal channels are stacked side by side. The numbers next to the photodiodes indicate the index in the image of where the channel is placed. The horizontal and vertical orderings are annotated to show which patterns match specific hand movements. The blue and red lines with markers indicate the channels which are related, i.e. they give information about the direction of movement. Red channels indicate side-to-side movement, with left or right indicated by the arrows in the patterns. Similarly, the blue channels indicate the up and down movement of the hand of the user. For additional explanation, let us consider the horizontal image and the first blue line as an example. Channels 1, 4, and 7 are connected with a blue line, with channel 7 getting dark first, then 4, and then 1. This indicated the shadow moving from channel 7 to 4 and then to 1, meaning the hand of the user is moving up, i.e. the upward movement if encoded in this specific channel transition. The diagonal case is not annotated, as it is not intuitively obvious which channels are related. However, a deep learning model might be able to extract this information if it is there. In Section 6.1.2, the influence of channel order is tested on two different models, and the resulting accuracy is measured.

5.2.5 Sampling delay distortion

In an ideal scenario, all the photodiodes are sampled at the same time instance. In that case, the output voltage from each of the photodiodes is measured at the same time. However, in practice, this is often not efficient as this requires a lot of extra hardware, and it might also be unnecessary as long as the system can tolerate some amount of distortion. The photodiodes are sampled one by one, from top left to bottom right. This can be compared to how digital cameras sample their sensors, scanning line by line. In the context of the digital camera, distortion can be caused by movement in either the camera or the image, which can result in blurry images. For the photodiode array, distortion can be defined as the relative change in illuminance of the photodiodes between times t_{start_sample} and t_{end_sample}, i.e., the time it takes to sample the whole array from photodiode 1 to 16. The first sampled photodiode will experience no distortion, and the last sampled photodiode will experience the highest distortion. The factors

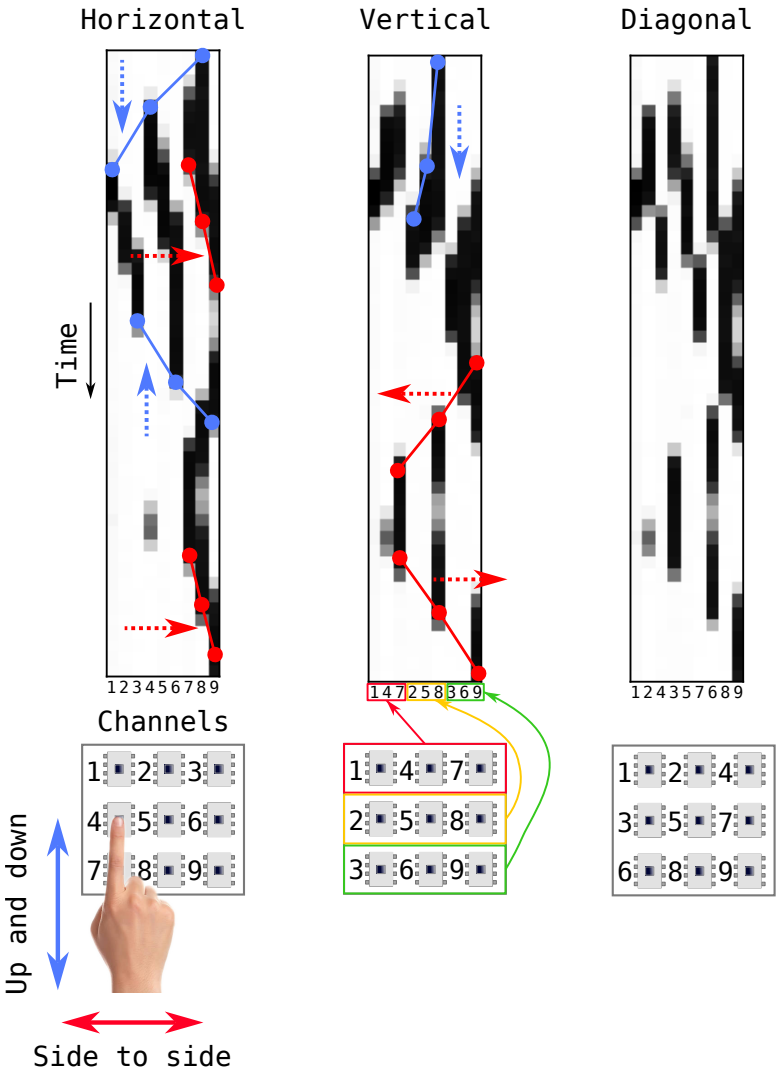


Figure 5.5: **Different channel ordering and how movement is related to shadow shifts.**

that influence the amount of distortion are the users' hand movement speed, sampling speed, and gradient of the shadow. An upper bound for the amount of distortion can be derived, based on a given hand movement speed and sampling speed and assuming no gradient. The distortion is directly proportional to the change in illuminance of the photodiode, as the current through the photodiode is an approximately linear function of illuminance (if it is within the proper operating region). The maximum change in illuminance is obtained when a shadow with an edge gradient $\ll$ d is moving at a 45° angle over the photodiode. Figure 5.6 shows how the upper bound can be derived, with the relative change in shadow area marked in red. An estimate of 0.1 m/s for hand movement speed was determined by moving a user's hand over a measuring tape, taking a top-view video of it, and extracting the speed using the video timestamp. The change relative to the sensor area is then obtained by dividing by the area of the sensor L^2, resulting in Equation 5.1.

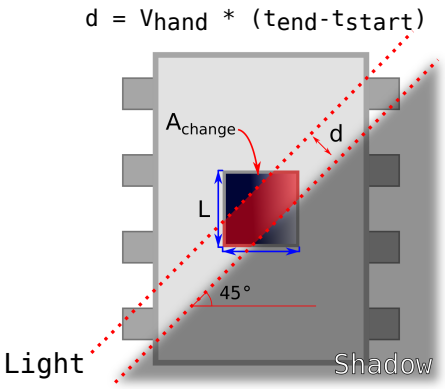


Figure 5.6: **Example of distortion caused by the delay in sampling the first photodiode and the last photodiode in the array.**

$$A_{relative\ change} = \frac{Ld\sqrt{2} - \frac{1}{2}d^2}{L^2} \tag{5.1}$$

This equation shows the maximum change in the shadow area covering the photosensitive area between the sampling start and end times. Filling in L=2.3e-3 m as the photodiode side length and d=0.1*1.05e-3 m results in 0.075 or 7.5%. However, when using only a subset of the photodiodes this is reduced further. As the LightDigit dataset contains only 9 channels, the sampling time becomes 0.59 ms, resulting in 0.036 or 3.6%. To evaluate the impact of sampling delay distortion on classification accuracy, it is tested on 3 different deep learning models in Section 6.1.4.

5.3 Expanding the LightDigit dataset

For model training, the LightDigit dataset is used. It is a dataset constructed by Hao Liu in 2021 for tackling the problem of digit recognition. The dataset consists of 20880 samples of air-written digits collected by one person, who simulated different writing styles. Additionally, it contains 5500 samples from 24 different participants with different nationalities. The dataset contains 9 channels collected from a 3×3 grid out of the total 16 photodiodes on the photodiode array.

In μLightDigit, we expand the original dataset by mirroring all training data across the time axis. These "new" samples are exactly the same digits as before, except for the written in the reverse direction. This is highly beneficial since this data is very easy to obtain and results in higher accuracy, and makes the models generalize better.

5.4 Tiny deep learning algorithm

Three deep learning models are considered for facilitating digit classification: a convolutional neural network (CNN), long short-term memory (LSTM) model, and a convolutional-LSTM hybrid model. This section provides background information on the model architectures and provides the designs of each of the models which will be tested. Model distortion resilience and the photodiode calibration algorithm will also be discussed. In Chapter 6, the models' architectures outlined here will undergo a preliminary evaluation to determine the feasibility

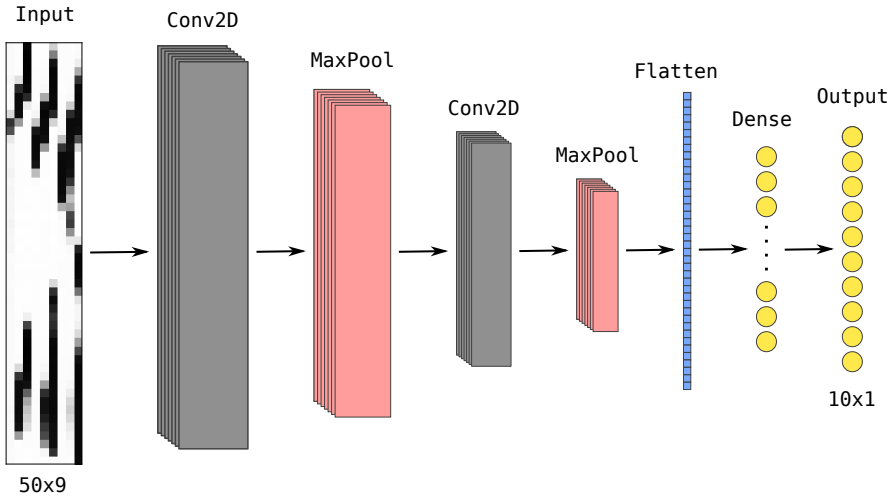


Figure 5.7: **CNN model architecture.**

of running them on the microcontroller and to choose which models will be used in the final evaluation. The final evaluation will test model performance and robustness in 3 indoor and 1 outdoor scenario.

5.4.1 Convolutional neural network model

Convolutional neural networks (CNNs) are most widely used for image recognition tasks. They can extract spatial patterns from images by performing convolutions over the original image with small, trained images called kernels. However, in research, they are also frequently used to classify time series data, such as finding specific segments in audio (keyword spotting). Considering our specific use case, the patterns which are created from drawing digits can be considered images, as different digits provide a unique spatial pattern. The architecture which was used for the evaluation can be seen in Figure 5.7, and is based on LeNet-5 [64], an old but still widely used architecture. CNNs provide the added benefit of dimensionality reduction through convolutional and pooling layers, reducing the number of operations required. From previous tests using keras-tuner it was found that kernels of 9×3 and 5×3 are beneficial for the data which is provided. This makes sense, as the input data is not square but rectangular and most changes in the patterns happen along the vertical axis. The amount of filters is varied between $N = 16 \dots 64$ and is kept the same in both layers.

5.4.2 Long short-term Memory model

A long short-term memory model (LSTM) [65] is considered. It is frequently used in related research, as it is suitable for classifying time series data. This model has a large number of parameters, as the intermediate hidden state needs to be returned (`return_sequences=True` in TensorFlow) to get accuracy that is comparable to the CNN. Without this setting, the TFLite model greatly suffers in accuracy when the model is converted to TFLite and quantized. The architecture can be seen in Figure 5.8. The model uses a single LSTM layer and a dense layer, with the number of units in the LSTM layer varied between $N = 16, \dots, 128$.

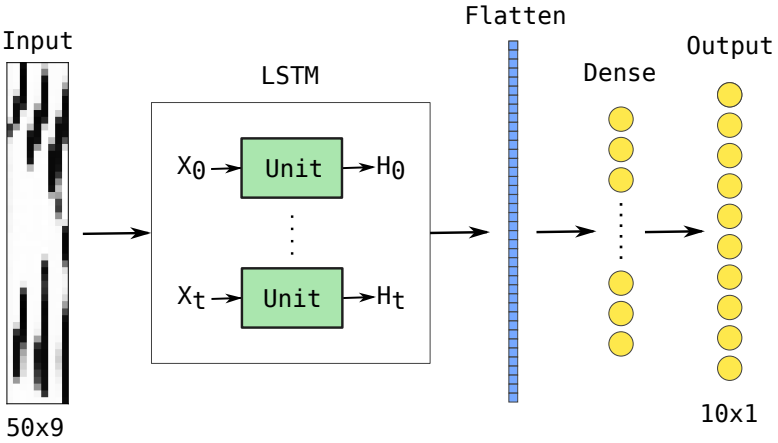


Figure 5.8: **LSTM model architecture.**

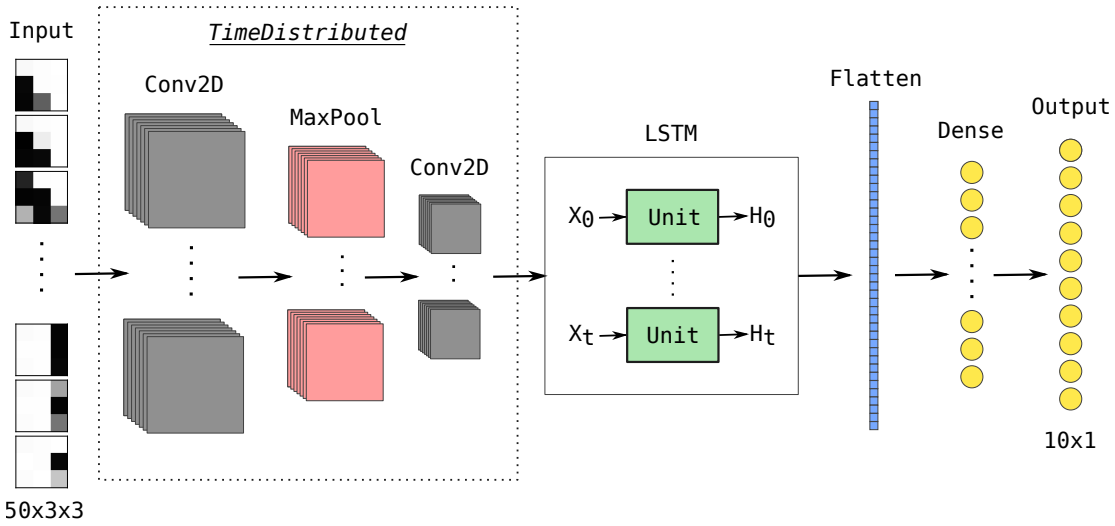
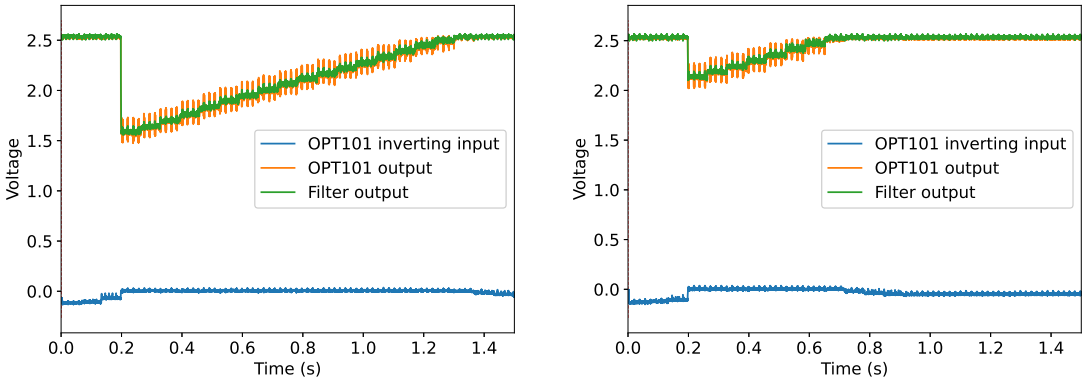


Figure 5.9: **ConvLSTM model architecture.**

5.4.3 ConvLSTM model

A hybrid model using both convolutional layers and an LSTM layer [66]. The idea is to combine the strengths of both convolutional layers and LSTM layers. The first part consists of convolutional layers which are meant to extract spatial patterns from each time step and reduce the dimensionality. The second part consists of an LSTM layer that extracts temporal patterns, i.e., patterns along the time dimension. To realize this, `TimeDistributed` layers are used to apply the convolutional and pooling layers to every time step, which is input as a 3×3 grid. The amount of filters for both convolutional layers is kept constant at 32, with kernel sizes 3×3 and 2×2, while varying the amount of LSTM units between $N = 16, \ldots, 128$. The architecture can be seen in Figure 5.9.



(a) **Calibration in 700 lux ambient light intensity.**

(b) **Calibration in 1000 lux ambient light intensity.**

Figure 5.10: **Output measurements of the photodiode array during calibration.**

5.4.4 Photodiode calibration

An algorithm was developed for automatically calibrating the photodiode array to the ambient light intensity. The goal of the calibration is to provide the output of the photodiode's maximum dynamic range, i.e. the highest possible output resolution. This is done through a series of measurements with different feedback resistance values. The measurements are started with the maximum possible resistance value, i.e. the highest sensitivity. In almost all except dark environments, this will cause the photodiodes to reach saturation. Then the resistance value is reduced until a significant drop in output voltage is measured, after which the resistance is gradually increased. If consecutive measurements then fall within a predefined range the measurements are considered the same and calibration is complete. During calibration, the light intensity is averaged over the entire photodiode array and measured several times per iteration. The light intensity should remain constant across the photodiode array during calibration, which can take up to 2 seconds to complete. Figure 5.10 show the output of the photodiode array in 2 different ambient light intensities. Note how the signal approaches the maximum possible output as the calibration converges. Also, at the start of the calibration, the photodiodes are saturated, which can be seen by the negative voltage at the inverting input. This happens because the opamp cannot provide the required voltage to compensate for the photodiode current.

Chapter 6

Performance Evaluation

This chapter describes the evaluation of the proposed deep learning models for μLightDigit. It consists of two main parts: the preliminary evaluation and the final evaluation.

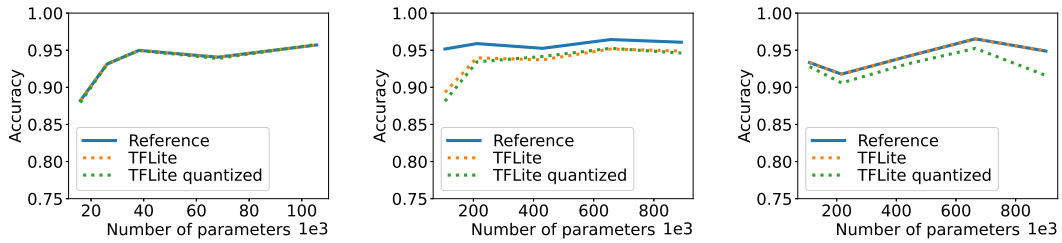
Preliminary evaluation. All three model architectures which were described in Chapter 5: a CNN, LSTM, and ConvLSTM are tested if running them on the microcontroller is feasible. Two practically applicable evaluation scenarios from the LightDigit dataset are used to determine model accuracy. These scenarios are also used to perform three tests using transformed input data: several input data length dimensions, three different channel orderings, and the influence of sampling delay distortion are investigated and their impact on accuracy is measured. Through post-training quantization, model weights and biases are quantized from regular 32-bit floating point numbers to 8-bit integers. The models are again evaluated to see the impact the quantization process had on the accuracy. Then, the models are evaluated on the microcontroller to determine inference time and model size. Two of the best models in terms of accuracy in both testing scenarios are chosen to continue to the final evaluation.

Final evaluation. The robustness is investigated on the two best-performing models from the preliminary evaluation. This is done by measuring the accuracy of the CNN and the ConvLSTM models in different lighting conditions. Simultaneously, the new photodiode array is tested if it is capable of functioning in a wide range of light intensities. A set of four different reproducible test setups have been created for model testing, covering one outdoor and three indoor scenarios, in a wide range of light intensities. Using two people, test data is collected in the scenarios, each providing 20 samples for every digit. Besides, for each test setup, the corresponding shadow pattern is collected. For the evaluation, the accuracy of each model is measured, and the results are scrutinized with the help of collected shadow patterns, confusion matrices, and image patterns.

6.1 Preliminary evaluation with LightDigit dataset

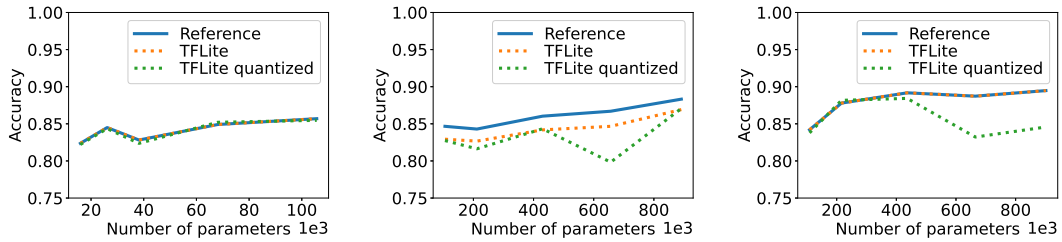
For the preliminary evaluation, only data from the LightDigit dataset is used. Two reference evaluation scenarios are constructed specifically for the preliminary evaluation of the models. The dataset is split into two sets, 80% of the data is dedicated to the training set and 20% to the test set. Each model is trained using only the training set and its performance is evaluated by the accuracy it is able to achieve on the test set. The way this split is made is different for both evaluation scenarios, and is detailed below:

1. *Intra subjects*: The test set contains data from people who are also in the training set.
2. *Inter subjects*: The test set does not contain data from people who are in the training set.



(a) **CNN** (b) **LSTM** (c) **ConvLSTM**

Figure 6.1: **Intra subjects accuracy comparison.**



(a) **CNN** (b) **LSTM** (c) **ConvLSTM**

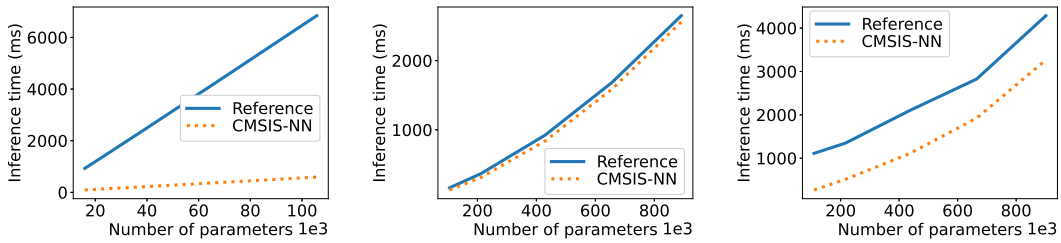
Figure 6.2: **Inter subjects accuracy comparison.**

Both scenarios are considered realistic in a practical sense. For example, if the system would is placed in a new environment with unknown users the inter subjects scenario applies. However, as the system is used and the model is updated with new data it would transition to intra subjects. Inter subjects represents the most challenging scenario, as some writing styles might not be in the training set. Therefore, this scenario results in lower accuracy compared to the intra subjects scenario. In order to eliminate randomness in the training process, the results of each scenario are averaged over 5 runs.

The main goal of the preliminary evaluation is to determine which kind of model architecture would be suitable for running on the microcontroller in terms of inference time and model size. Both evaluation scenarios detailed in Section 5.3 are tested on all three model architectures discussed in Chapter 5. Figures 6.1 and 6.2 show accuracy performance of the reference model, unquantized TFLite model and 8-bit quantized TFLite model. The accuracy received from the TensorFlow model is considered the baseline, and the TFLite and quantized TFLite are compared to the baseline model to establish the effects of model conversion and quantization on accuracy. To make a reasonable comparison, each model type is provided with roughly the same layout, with the output layers being the same for all the models. The output layers consist of a flatten layer, dense layer and the output layer. Before varying parameters specific to the different model architectures, an evaluation of the amount of dense nodes in the dense layer was performed. After the evaluation the amount of dens nodes is fixed at 128, as adding extra nodes made only a very slight difference in accuracy, while adding significantly to model size.

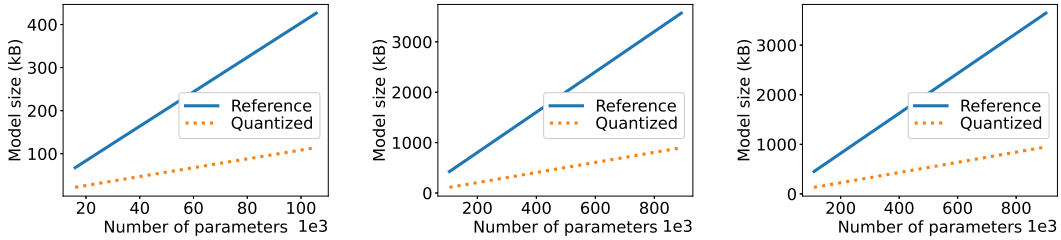
6.1.1 Model comparison

The CNN experiences no decrease in accuracy from the TFLite conversion and in some cases, the quantized model even outperforms the reference model. This is contrary to both the LSTM and



(a) **CNN** (b) **LSTM** (c) **ConvLSTM**

Figure 6.3: **Inference time comparison for models run on the MCU.**



(a) **CNN** (b) **LSTM** (c) **ConvLSTM**

Figure 6.4: **Model size comparison.**

ConvLSTM models, which do suffer an accuracy loss of up to 10% for some model sizes. This is most likely due to the inference reliability LSTM being much more sensitive to quantization than the CNN, and as the ConvLSTM also uses an LSTM layer it is also negatively affected by this. Besides accuracy, the other important factors are inference time and model size, which are compared in Figures 6.3 and 6.4 respectively. The CNN has the slowest inference time using the reference kernels but benefits greatly from the optimized kernels, providing a 10-fold decrease in inference time. On the other hand, the LSTM sees almost no benefit from the optimized kernels, and the ConvLSTM has a moderate constant benefit, as the convolutional layer is kept at the same size. The CNN is also the smallest of the models in terms of size. However, with quantization, even the largest of both the pure LSTM and ConvLSTM models still remain below 1 MB in size, which is acceptable for the microcontroller. For the final evaluation, two model architectures are picked which perform best in both scenarios of the preliminary evaluation. The pure LSTM is disregarded because of its slow inference time and poor quantized performance. Due to excellent post-quantization performance, a high-parameter CNN is the first choice, as it performs with an accuracy of 95% and 85% in both testing scenarios. For the second model, a medium-parameter ConvLSTM model is chosen, as it has better performance than the CNN in inter subjects, and inference time remains below 1 second.

6.1.2 Influence of channel order

As mentioned in Section 5.2.4, the input data can be altered by changing the order in which channels appear in the image patterns. Doing so changes the way movement information is encoded in the input data. This might change the way the models perceive the input data, and influence their accuracy. Especially the CNN, as this is a model which extracts its information from spatial patterns. Horizontal, vertical, and diagonal channel orderings are tested for the

Table 6.1: **Different channel orders with accuracy from LSTM and CNN models for the intra subjects scenario.**

	Channel order	LSTM	CNN
Horizontal	1 2 3 4 5 6 7 8 9	0.953	0.941
Vertical	1 4 7 2 5 8 3 6 9	0.948	0.939
Diagonal	1 2 4 3 5 7 6 8 9	0.952	0.924

Table 6.2: **Impact of different data lengths on model accuracy for the inter subjects scenario.**

Data length (N)	LSTM	CNN
100	0.958	0.941
75	0.963	0.942
50	0.956	0.945
25	0.953	0.933

LSTM and CNN models. The ConvLSTM is left out, as it is a combination of the two other models. The results can be found in Table 6.1. It is observed that changing channel order has only a small influence on model accuracy, staying virtually equal for the LSTM and only dropping 1.7% in the diagonal case for the CNN.

6.1.3 Influence of data length

Several data input lengths have been considered ($N = 100 \dots 25$), with Table 6.2 showing different dimensions of N with resulting accuracies tested for the intra subjects evaluation scenario. It was chosen to use a length of $N = 50$ samples for the models, as at this point it begins to drop in accuracy for the convolutional model, while still being small enough to not cause an excessive number of parameters in the models.

6.1.4 Robustness to distortion

To ensure the sampling delay does not cause the models to significantly drop in accuracy, the impact of distortion is investigated on the most significant models. An estimation of distortion was calculated for the test set of the model, and the model was evaluated using this test data. The maximum distortion is the maximum attainable distortion on the channel (i.e. photodiode) which is sampled last, as illustrated in Figure 5.6. As distortion increases approximately linearly for subsequently sampled channels, the value is linearly interpolated between each channel. Then this value is either subtracted from or added to the values of the test set, depending on the sign of the time derivative of the samples. If the time derivative is positive, meaning the channel is getting brighter, the distortion value is added to the corresponding channel. Vice versa for when the time derivative is negative. Figure 6.5 shows the distortion robustness for the medium-sized

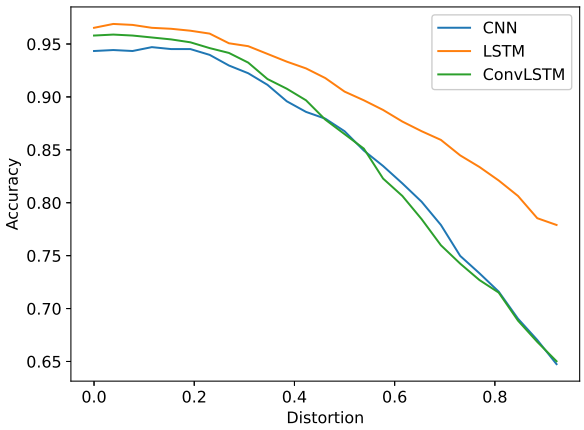


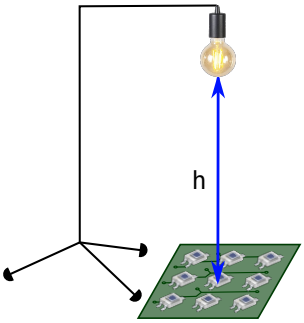
Figure 6.5: **Distortion robustness of the different reference models using the intra subjects scenario.**

model for each of the model types. The accuracy remains stable until 20% distortion, after which it starts to decline. This is well below the 7.5% maximum distortion calculated in Section 5.2.5, so this has no influence on model performance.

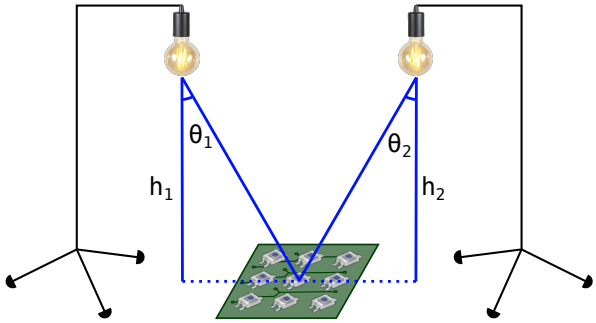
6.2 Final evaluation setups

This section details the test setups which were used to collect verification data, which is used for the final evaluation. All data was collected using the new photodiode array design. The data will be used to test the robustness of the models in different lighting environments, which causes different shadow patterns. Setups 1, 2, and 3 used a tripod with Sencys 2700k 10 W light bulbs attached to the handle. The distance between the hand of the user and the photodiode array was attempted to be kept between 5 and 10 mm. The exact parameters of the test setup are listed in table 6.3. The data is collected by two people, one of which is in the training data and one is not. The results are averaged to represent a scenario somewhere between inter and intra subjects, to provide some sense of the average performance of the system.

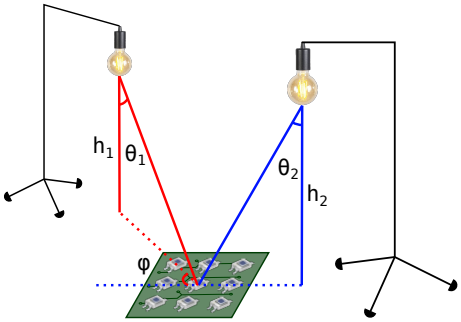
- **Setup 1:** Single light source directly overhead, with light at different heights to create different light intensities. This setup is mainly to show performance in different light intensities.
- **Setup 2:** Symmetrically placed light sources on the center line of the photodiode array. Height h remains the same but angle of incidence θ is varied. This setup is to show the influence of wider shadow patterns on model accuracy.
- **Setup 3:** Asymmetrically placed light sources, with the left light source rotated φ°around the mid-point of the photodiode array. This setup is to show the impact of a non-symmetric shadow pattern.
- **Setup 4:** Sunlight, however since photodiode saturation remains a problem at high light intensities (above 15k lux), the data collection is performed during overcast conditions. This setup is to show performance in natural light, which causes a vastly different shadow pattern as light is incident from all sides.



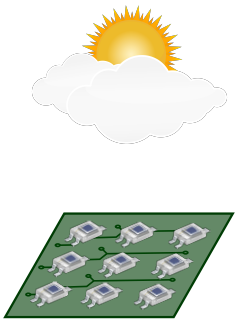
(a) **Test setup 1: single overhead light source.**



(b) **Test setup 2: symmetric light sources.**



(c) **Test setup 3: 2 asymmetric light sources.**



(d) **Test setup 4: sunlight.**

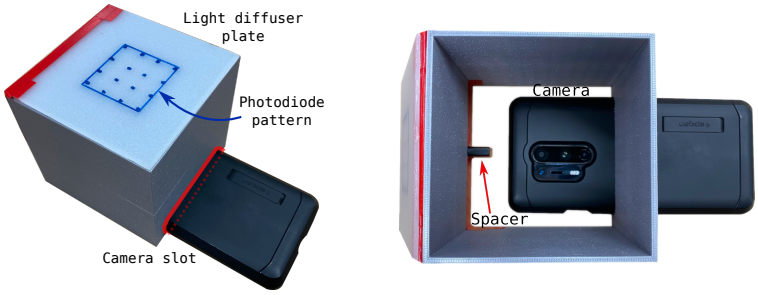
Figure 6.6: **Evaluation setups.**

Table 6.3: **Test setup parameters**

Test setup	$\mathbf{h}_{1,2}$	$\boldsymbol{\theta}_{\mathbf{1,2}}$	$\boldsymbol{\varphi}$	**Illuminance**	$\mathbf{R}_f$
1a	0.17 m	-	-	5000 lux	63 kΩ
1b	0.5 m	-	-	1000 lux	226 kΩ
1c	0.6 m	-	-	500 lux	525 kΩ
2a	0.4 m	15 °	-	1350 lux	195 kΩ
2b	0.4 m	15 °	-	900 lux	280 kΩ
3	0.6, 0.4 m	30 °, 15 °	45 °	900 lux	280 kΩ
4	-	-	-	10k lux	15 kΩ

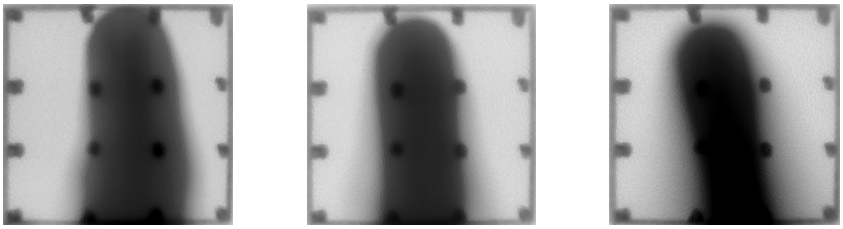
6.3 Shadow patterns

To better inspect the shape and dimensions of each shadow created in different testing scenarios the shadow patterns have been recorded. To obtain the shadow pattern, a 3D printed box with a light diffuser lid was created and is shown in Figure 6.7. The lid is a light diffuser plate with a drawing of the approximate positions of the photodiodes and provides a reference for shadow dimensions. A slot at the bottom of the box was created such that it fits a smartphone for video recording. Each shadow pattern image contains the approximate dimensions of the photodiode array. The shadow patterns were edited to enhance the visibility of the shadow, by turning the image to greyscale and increasing the contrast. It is very obvious that the shadow pattern in setup 4 is much weaker compared to the other scenarios.



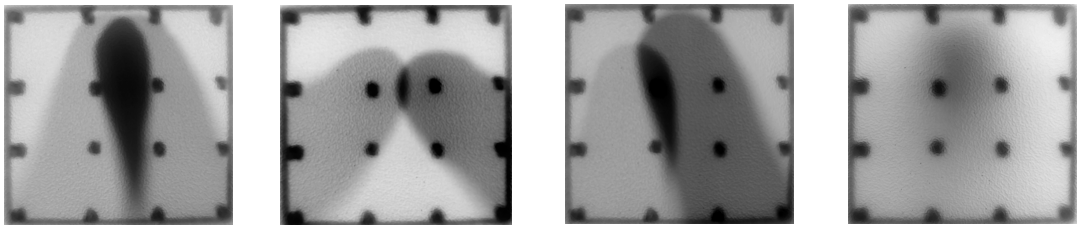
(a) **Side view with closed lid** (b) **Top view with open lid**

Figure 6.7: **Setup for capturing shadow patterns**



(a) **500 lux** (b) **1000 lux** (c) **5000 lux**

Figure 6.8: **Shadow patterns for test setup 1**



(a) θ**=15°** (b) θ**=30°**

Figure 6.9: **Shadow patterns in setup 2**

Figure 6.10: **Shadow pattern in setup 3**

Figure 6.11: **Shadow pattern in setup 4**

6.4 Final evaluation

For the final evaluation, two model architectures that performed the best in both the intra and inter subject scenarios are chosen from the preliminary evaluation. The first model is a relatively small CNN model with 64 filters and 100k parameters. The second model is a larger ConvLSTM model with 32 LSTM units and 400k parameters. These models will be tested in the final evaluation.

Hyperparameter optimization. The model hyperparameters are further optimized using keras-tuner through the hyperband algorithm to find possible better combinations of first and second layers filters, kernel sizes, and amount dense nodes, resulting in the models which can be found in Table 6.4. The model parameters are varied around the existing parameters in order to find the optimal combination of hyperparameters for the two chosen models. For the ConvLSTM model, a 1d convolution approach, as well as a 2d convolution approach, is tried as well as trying variations in the number of filters, kernel size, and amount of LSTM and dense nodes. However,

Table 6.4: **Scenario accuracy and statistics for the quantized tuned models**

Scenario	**CNN**	**ConvLSTM**
Intra subjects	0.957	0.954
Inter subjects	0.870	0.857
Statistics		
Parameters	138k	446k
Inference time	792 ms	947 ms
Model size	147 kB	494 kB

Table 6.5: **Prediction accuracy for both models in all scenarios**

Test setup	**CNN**	**ConvLSTM**
1a (one light source)	0.807	0.787
1b (one light source)	0.818	0.836
1c (one light source)	0.897	0.892
2a (two light sources, *symmetric*)	0.868	0.743
2b (two light sources, *symmetric*)	0.783	0.724
3 (two light sources, *asymmetric*)	0.970	0.893
4 (outdoor with sunlight)	0.778	0.892
Average	**0.846**	**0.824**

to keep both the inference time below 1 second and the quantized performance as high as possible, the maximum LSTM nodes of the ConvLSTM model are set at 40. For the pure CNN models, the kernel sizes and amount of filters are varied in the convolutional layers. For the pooling layers, the pool sizes, and strides are varied. For both models, there is also experimented with a dropout layer, but this is disregarded as it does not provide any extra accuracy improvements. Additionally, the output dense layer of both models has been tuned between 128 and 256 nodes.

Final models. For an initial comparison, the final tuned models are quantized and tested again using the LightDigit dataset reference scenarios. By tuning the number of filters, LSTM nodes, and dense layer size the models increased in size compared to the baseline models. To ensure the inference time and model size are still at an acceptable level, this was tested on the microcontroller. Both the reference accuracy and model statistics can be found in Table 6.4. The accuracy of both models is close, however, the CNN performs 1.5% better in the inter subjects scenario. This is likely due to the quantization causing accuracy degradation in the ConvLSTM. The inference time is also still within 1 second and the model size is within 1 MB.

6.4.1 Accuracy in different setups

The accuracy results for each setup are given in Table 6.5. In order to investigate the model accuracy differences across the test setups, we use the shadow patterns as described in Section 6.3 and examples of image patterns from each test scenario, which can be seen in Figure 6.12. In the following section, the accuracy of each setup is discussed. When averaging the CNN and ConvLSTM accuracy across all test setups, it can be concluded that both model types have comparable accuracy. However, it can be observed that the ConvLSTM performs slightly worse than the CNN, lagging behind 2% in accuracy. When considering specific setups, setup

1 accuracies are comparable, while the CNN performs best on setups 2 and 3 and ConvLSTM performs best on setup 4. It should be noted that the dataset for performing these tests is not large, as it only contains 40 samples per digit per setup. This can give an idea of the model performance in the real world, but could also lead to variation in the results. To minimize the influence of variation due to the training process, the models have been trained and tested 5 times, and the results are averaged.

Setup 1 (one light source)

As can be seen from the shadow patterns for setup 1 in Figure 6.8, they all look very similar to each other. All of them have a single connected shadow with strong contrast. This is reflected in Figure 6.12 by very clear lines separating shadow from light areas. Both models are also performing on par with each other, differing at most 2%. Results for setup 1c are better than for 1a and 1b, showing an accuracy of close to 90% for both models, compared to 78.7–83.6%. It may be performing slightly better because the shadow pattern is somewhat narrower at the base of the finger, with a larger gradient compared to 1a and 1b.

Setup 2 (two light sources, symmetric)

Setup 2 yields the overall worst performance of all setups and the CNN greatly outperforms the ConvLSTM. The CNN is still able to achieve 86.8% and 78.3% accuracy on setups 2a and 2b respectively, while the ConvLSTM only reaches 74.3% and 72.4%. While the lines in the image pattern in 2a are already fading compared to setup1, setup 2b proves the most challenging of all. Associated with this setup is a wide, relatively homogeneous shadow with only a small central dark spot which can be seen in Figure 6.9b. By having a poorly defined shadow of the fingertip, the shadow captured creates weak, low-contrast lines in the corresponding image pattern. It is also highly sensitive to changes in finger height, as raising the finger higher will cause the secondary shadows to diverge. This can be observed in Figure 6.12, as the top part is darker than the bottom part. Having such poor separation between channels makes it harder for the models to identify channel shifts.

Setup 3 (two light sources, asymmetric)

Setup 3 is the best setup for both models, with the CNN reaching 97.0% while the ConvLSTM is able to achieve 89.3%. The shadow pattern can be seen in Figure 6.10, and is similar to setup 2a, showing a large concentrated shadow area below the fingertip. The fact that the secondary shadows are asymmetric does not seem to influence the model performance. The general trend seems to be that shadow patterns with a large, concentrated shadow below the fingertip and a secondary shadow with lower contrast surrounding it performs the best. This may be explained by the fact that the larger, surrounding shadow can bridge the relatively large gaps between the photodiodes more easily, allowing the capture of more sublet hand movements. While at the same time, the exact fingertip position is very clear because of the high-contrast center shadow.

Setup 4 (outdoor with sunlight)

Setup 4 is the only one where the CNN gets heavily outperformed by the ConvLSTM, with 77.8% accuracy against 89.2%. The associated shadow pattern shows a low-contrast point shadow directly below the fingertip, which results in line fading in the corresponding image pattern. This explains why the CNN is having trouble in this case, as the lines have low contrast and get blurry between channels. However, the true reason for the performance difference between the

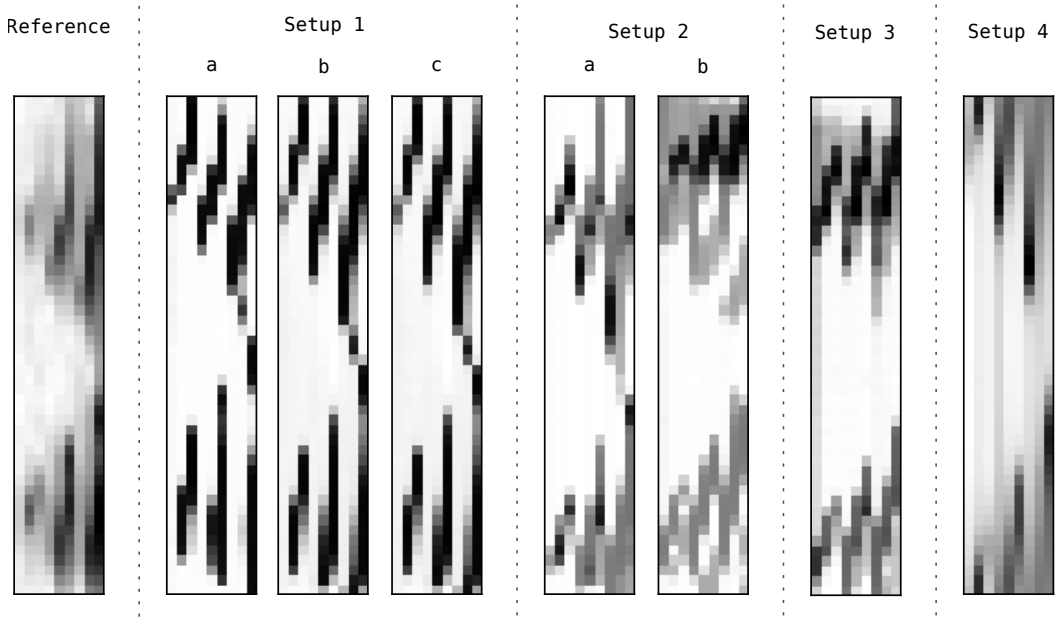
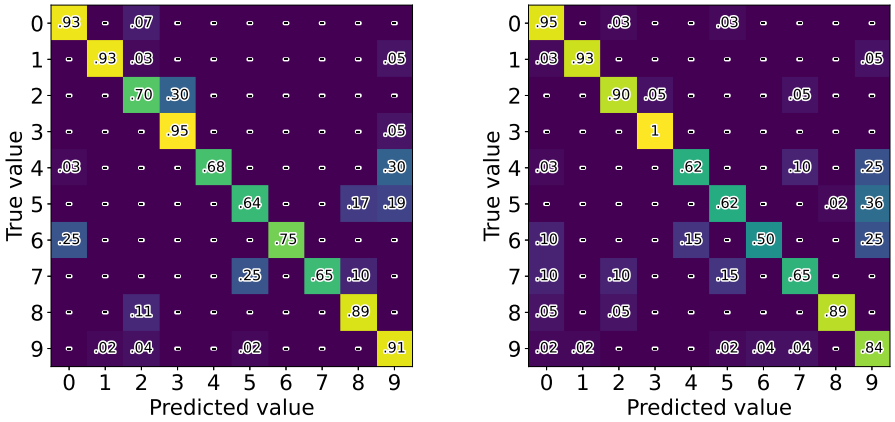


Figure 6.12: **Comparison of image patterns for every setup for the number 0 with the reference from the training set.**

two models in the same setup is quite difficult to explain, as it is not straightforward to show what the models "know" and what information they extract from the input data. However, as the CNN is solely focused on spatial patterns, the results from this model are more interpretable by humans as opposed to the ConvLSTM. As lines in the image patterns fade, it becomes more difficult for the CNN model to extract as many distinct and detailed characteristics, such as is the case in 2b and 4. On the other hand, the ConvLSTM is able to extract spatial information from individual samples and combine it with time information. In other words, the pure spatial features lose information as the lines become blurry, but the ConvLSTM is able to compensate for the loss of information with the addition of the temporal features. This is why, in the case of setup 4, the ConvLSTM has superior performance to the CNN. To conclude, both models have their strengths, but overall the CNN is the preferred model. It is smaller, has a lower inference time, no quantization error, and is faster to train, which outweighs the fact that the ConvLSTM is able to outperform it in some scenarios.

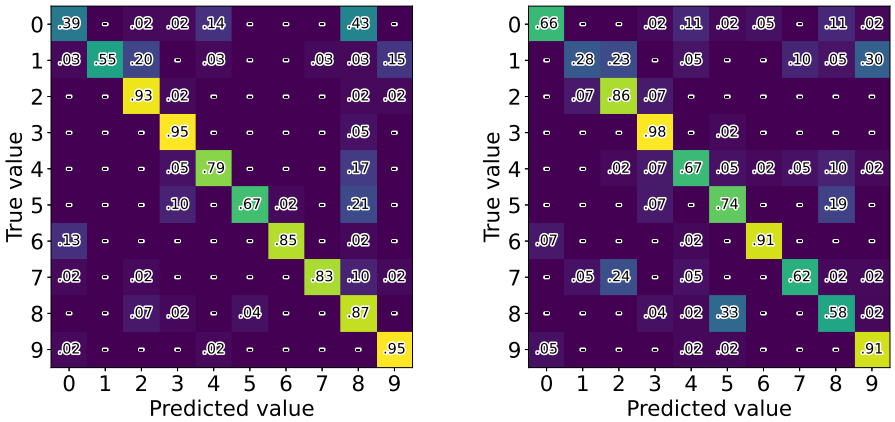
6.4.2 Digit misclassifications

To provide a deeper performance analysis of μLightDigit, a subset of all confusion matrices is used for explaining differences in accuracy (setups 1a, 2b, and 4), which can be seen in Figures 6.13, 6.14 and 6.15 respectively. The entire collection of confusion matrices can be found in Appendix C. By comparing confusion matrices in Figure 6.13, similarities in misclassifications can be observed by the different models. This could indicate problems with the data causing the misclassifications. By combining this information with the observed shadow patterns and image patterns, several explanations can be given which may lead to misclassification. It should be noted that these problem do not occur in isolation, but often in conjunction.



(a) **CNN 500 lux** (b) **ConvLSTM 500 lux**

Figure 6.13: **Confusion matrices for Setup 1a (one light source).**



(a) **CNN θ=30°** (b) **ConvLSTM θ=30°**

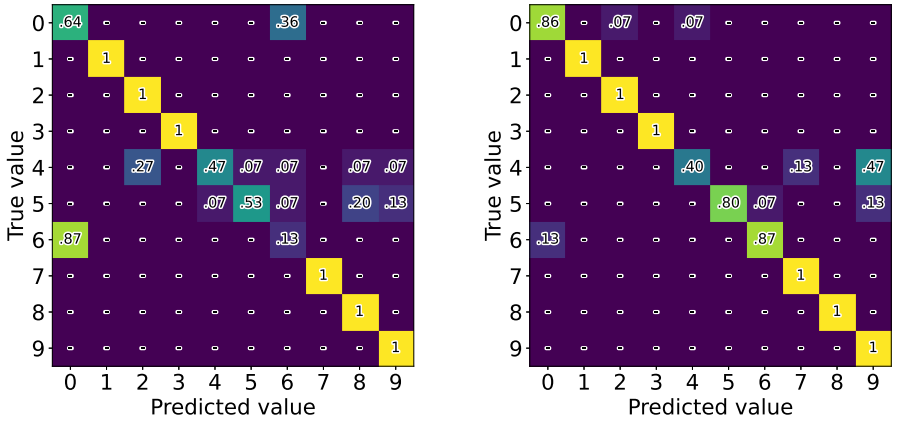
Figure 6.14: **Confusion matrices for Setup 2b (two light sources, symmetric).**

Motion similarities between digits

The writing motion of specific digits is very similar. An example of such cases happening can be observed in setup 1 in particular, where the 4 is frequently mistaken as being a 9. Depending on the handwriting, someone might write a 4 with exactly the same motion as writing a 4, but without closing the top circle. However, the similarity between digits on its own is often not enough to cause misclassification. This happens due to added distortions that are introduced through other ways which are discussed below.

Fly-in and fly-out distortion

Before someone starts writing, the hand is moved in position to where a person starts writing a certain number. For different handwriting, this starting location might vary. This problem is called *fly-in* and refers to the hand of the user "flying into" the sensing area, causing shadows to be detected and sampling to start. An example of where this might cause a problem is between the digits 4 and 9. Moving the hand towards the starting position of the 4 might exactly mimic a portion of writing the 9. Similarly, *fly-out* is the opposite case, where the hand "flies out" of the



(a) **CNN 10k lux** (b) **ConvLSTM 10k lux**

Figure 6.15: **Confusion matrices for Setup 4 (outdoor with sunlight).**

sensing area after writing is complete. This can cause numbers such as 5 and 8 to be confused.

Low resolution aliasing

The third cause for classification problems that can be observed is the photodiodes not capturing any shadow for some period of time. This means the combination of finger location and shadow pattern causes the shadow to fall between photodiodes or outside the sensing area. This results from both the number of photodiodes and the photodiode area being low and the distance between the photosensitive area being relatively large. Some parts of shadow patterns might be missed if they are not directly moved over a photodiode, such as is the case for setups 2b and 4. Visual evidence of cases of low-resolution aliasing can be observed in 6.12 setups 2b, 3, and 4. The lack of grey in the middle part of these image patterns indicates that no shadow is captured. This could be solved by creating a more densely packed photodiode array with more photodiodes.

6.4.3 Photodiode array limitations

During testing in the different verification setups, it became clear that the photodiode array is only able to handle illumination of up to 15k lux. Beyond this point, the output gets locked at the positive power supply rail and does not decrease even when the photodiode is fully covered. It is not quite clear why this happens, but light entering the side of the package has some influence. There might also be internal effects in the photodiode package, such as excessive leakage, which lock the output in saturation. However, this could not be confirmed through simulation or testing. Even though this forms a problem outside in sunny conditions, virtually all indoor use cases for the system are covered. Therefore, it does not cause problems in most practical scenarios.

Chapter 7

Conclusion

The μLightDigit project is the second iteration of the LightDigit project, which has developed a contactless air-writing system based on ambient light detection using embedded deep learning on a Raspberry Pi 4, using only a 4×4 grid of simple photodiodes. The main challenges of this project were to develop a novel TinyML system using a commodity microcontroller, which is robust in a large variety of lighting conditions. This was achieved by transitioning LightDigit to an STM32H743, an ARM Cortex-M7 based microcontroller. To facilitate classification, a deep learning model was created which can run in real-time on this hardware. Additionally, a new photodiode array was designed to overcome the saturation issues of the photodiode array initially developed for LightDigit, which greatly increased system robustness. The photodiode array was made to have an adjustable light sensitivity, which is made possible through a programmable potentiometer in the feedback network. The newly designed photodiode array has excellent adaptability to light intensities up to 15k lux. Through a calibration algorithm, the photodiode array can automatically be calibrated to the surrounding light intensity to provide the optimal dynamic range.

To investigate the right kind of deep learning model, an analysis was performed that tested multiple types of deep learning architectures. Three different kinds of embedded deep learning models were tested: a convolutional neural network (CNN), long short-term memory (LSTM), and a CNN-LSTM hybrid model. The models were developed using TensorFlow, converted to an optimized and quantized format using TensorFlow Lite and finally were run on the microcontroller using TensorFlow Lite for microcontrollers. Using the LightDigit dataset, the models were trained using the inter and intra subject scenarios. A preliminary evaluation of the three different models was performed, where each of the models was tested on the microcontroller, reporting on inference time and model size. The influence of data length, channel order, and sampling delay distortion was also investigated. Data length was set at 50 samples, the channel ordering horizontal and the amount of distortion that can be tolerated is up to 20%, which is well below the calculated maximum of 7.5%. Through comparison of accuracies achieved in the evaluation scenarios, it was found that the CNN and ConvLSTM (with a low number of LSTM units) are able to maintain accuracy after post-training quantization, while also maintaining satisfactory inference time. Through highly optimized operation kernels provided by the CMSIS-NN library integrated into TFLM, the inference time of the models can be greatly reduced, especially the CNN. The pure LSTM was found to suffer from accuracy loss after quantization and was not able to greatly benefit from inference time speedup through CMSIS-NN.

For the final evaluation, a CNN and ConvLSTM were chosen which provided the best performance in the preliminary evaluation. In order to test system robustness, 4 different, reproducible

verification setups were created to test different ambient lighting conditions and light intensities. Shadow patterns were recorded and compared to explain the results. The CNN resulted in an average accuracy of 84.6% and the ConvLSTM an average accuracy of 82.4%. The top result for both was achieved in setup 3, with the CNN achieving 97% accuracy, while the ConvLSTM achieved 89.3%. The CNN was the lightest model with 792 ms inference time and a total size of 138k parameters, which translated to 147 kB. ConvLSTM was larger with 446k parameters at 494 kB and 947 ms inference time. Both models remain below 1 second inference time, fast enough for real-time application. The ConvLSTM performed best in the outdoor testing scenario with 89.2% compared to 77.8% of the CNN. However, the CNN was generally superior in all other indoor test setups, which will be the primary deployment environment of the system. Given the extra advantages that the CNN provides, such as excellent quantized performance, low inference time, low model size, and fast training it is preferred over the ConvLSTM. At the inference time is deemed too long, this can be shortened by reducing the number of filters in the convolutional layers, and the number of dense nodes in the dense layer. It will be at the cost of a few percentage points of accuracy, however good results above 90% accuracy are still attainable.

7.1 Future work

During the development of μLightDigit, new ideas have come up for improvements that could be implemented in future iterations. Even though the photodiode array has been improved by having adaptable sensitivity, it is not able to handle bright sunlight. There also exist unresolved problems which can be addressed in future revisions, such as dealing with modulated led light.

7.1.1 Fly-in and fly-out distortion

An investigation can be performed to determine the impact of fly-in and fly-out distortion during the air-writing phase. An algorithm can be developed that strips off some part of the data, or by using some other method, minimizes the influence of fly-in and fly-out distortion.

7.1.2 Modulated LED light

In recent years, dimmable LED lights are more frequently used in indoor lighting scenarios. These lights are pulse-width modulated (PWM) with high-frequency signals to facilitate the dimming of the LEDs. The human eye is not able to pick up these signals, but the photodiodes do see this signal. It is present as high-frequency noise in light measurements, which can be detrimental to the accuracy of the models. An algorithm can be devised which can detect the modulation frequency and adapts the photodiode sampling in such a way that the photodiode is only sampled on the high pulse. This would make the system even more robust in different lighting conditions.

7.1.3 Data collection during operation

As it is a large effort to collect more data, this can be performed while the device is being used in practice. New data samples can be saved to the internal flash of the microcontroller and offloaded to another device that can train a new model. The model can then be swapped out with a new model which should be able to achieve higher accuracy.

7.1.4 Finger trajectory estimation

First extracting a trajectory and then performing digit classification on a simpler, 2D image has been successfully performed in existing research. This eliminates the time aspect of the data and greatly simplifies the classification problem. It would also allow the use of the MNIST dataset for handwritten digits, and can perhaps also be relatively easily expanded to other handwritten symbols. Estimating the trajectory of the finger was attempted using optical flow techniques. Experiments were performed using the Lucas-Kanade and Horn-Schunck methods to extract the optical flow vectors from individual samples. However, it was found that the optical flow vectors were moving with heavy biases in certain directions and the results were not good enough to be usable. A likely cause for this might be the small size of the individual samples (3×3). Another method could be developed to estimate a finger trajectory from the data.

7.1.5 Improved photodiode array design

The existing photodiode array can be further improved to avoid low-resolution aliasing and bright light saturation. The OPT101 is already a very old design IC, and modern photodiodes have advantages. Using a different photodiode with lower junction capacitance, such as VBP104S (48 pF compared to 1.2 nF), avoids needing large stabilization capacitors which result in faster response time and higher system bandwidth. Another advantage would be a smaller package size, which means more photodiodes can fit closer together for higher resolution.

Recommendations for photodiode array design

When building a new photodiode array with an external op amp, it is recommended to use a high bandwidth rail-to-rail operational amplifier suitable for low supply voltage. Also, note that transimpedance amplifiers used with photodiodes should take DC parameters into account. The most important is the input bias current, which should be much smaller than the photodiode current. A good choice would be a precision rail-to-rail opamp such as the OPA320. A challenge would be to investigate how well the existing dataset translates to the use of different photodiodes. Modern, digital photodiodes can also be considered. This saves the trouble of finding a proper photodiode and opamp combination, stabilizing the circuit, and converting the output signal. All values can immediately be read digitally, however one should take note of the conversion time of these digital sensors. These can be relatively long, e.g. in the range of several hundreds of milliseconds, which is not fast enough to capture dynamic hand shadows as it would cause too much distortion.

Powering the system through the photodiodes

As photodiodes are simply small solar cells, they generate current which can be used to power devices. If μLightDigit can be fitted with a battery and larger photodiode array, it might be possible to run the device using only the power provided by the photodiode array. This would require substantial design changes to the current design, so decent knowledge of electronics design would be recommended.

Bibliography

[1] KS Goedemondt, J Yang, and Q Wang. Embedded AI Enabled Air-Writing for a Post-COVID World. In *42nd WIC Symposium on Information Theory and Signal Processing in the Benelux (SITB 2022)*, 2022.

[2] Lin Guo, Zongxing Lu, and Ligang Yao. Human-machine interaction sensing technology based on hand gesture recognition: A review. *IEEE Transactions on Human-Machine Systems*, 51(4):300–309, 2021.

[3] SHI Yuanyuan, LI Yunan, FU Xiaolong, MIAO Kaibin, and MIAO Qiguang. Review of dynamic gesture recognition. *Virtual Reality & Intelligent Hardware*, 3(3):183–206, 2021.

[4] Hao Liu. Lightdigit: Embedded deep learning empowered fingertip air-writing with ambient light. Master's thesis, Delft University of Technology, 2021.

[5] Rafiqul Zaman Khan and Noor Ibraheem. Hand gesture recognition methods and applications: A literature survey. *International Journal of Artificial Intelligence and Applications (IJAIA)*, 3:161–174, 08 2012.

[6] Munir Oudah, Ali Al-Naji, and Javaan Chahl. Hand gesture recognition based on computer vision: A review of techniques. *Journal of Imaging*, 6(8), 2020.

[7] Mais Yasen and Shaidah Jusoh. A systematic review on hand gesture recognition techniques, challenges and applications. *PeerJ Computer Science*, 5:e218, 2019.

[8] Krzysztof Czuszyński, Jacek Rumiński, and Alicja Kwaśniewska. Gesture recognition with the linear optical sensor and recurrent neural networks. *IEEE Sensors Journal*, 18(13):5429–5438, 2018.

[9] Aditya Tewari, Bertram Taetz, Frédéric Grandidier, and Didier Stricker. A probabilistic combination of cnn and rnn estimates for hand gesture based interaction in car. 10 2017.

[10] Vijay John, Ali Boyali, Seiichi Mita, Masayuki Imanishi, and Norio Sanma. Deep learning-based fast hand gesture recognition using representative frames. In *2016 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, pages 1–8, 2016.

[11] Chuyu Wang, Jian Liu, Yingying Chen, Hongbo Liu, Lei Xie, Wei Wang, Bingbing He, and Sanglu Lu. Multi - touch in the air: Device-free finger tracking and gesture recognition via cots rfid. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, page 1691–1699. IEEE Press, 2018.

[12] Wei Wang, Alex X. Liu, and Ke Sun. Device-free gesture tracking using acoustic signals. MobiCom '16, page 82–94, New York, NY, USA, 2016. Association for Computing Machinery.

[13] Sizhen Bian and Paul Lukowicz. Capacitive sensing based on-board hand gesture recognition with tinyml. In *Adjunct Proceedings of the 2021 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2021 ACM International Symposium on Wearable Computers*, pages 4–5, 2021.

[14] Brian Coffen and Md.Shaad Mahmud. Tinydl: Edge computing and deep learning based real-time hand gesture recognition using wearable sensor. In *2020 IEEE International Conference on E-health Networking, Application and Services (HEALTHCOM)*, pages 1–6, March 2021.

[15] Aditya Jyoti Paul, Puranjay Mohan, and Stuti Sehgal. Rethinking generalization in american sign language prediction for edge devices with extremely low memory footprint. In *2020 IEEE Recent Advances in Intelligent Computational Systems (RAICS)*. IEEE, dec 2020.

[16] Fenglin Liu, Wei Zeng, Chengzhi Yuan, Qinghui Wang, and Ying Wang. Kinect-based hand gesture recognition using trajectory information, hand motion dynamics and neural networks. *Artificial Intelligence Review*, 52(1):563–583, 2019.

[17] Adil Rahman, Prasun Roy, and Umapada Pal. Air writing: Recognizing multi-digit numeral string traced in air using rnn-lstm architecture. *SN Computer Science*, 2, 02 2021.

[18] Seong Kyu Leem, Faheem Khan, and Sung Ho Cho. Detecting mid-air gestures for digit writing with radio sensors and a cnn. *IEEE Transactions on Instrumentation and Measurement*, 69(4):1066–1081, 2020.

[19] Borja Saez-Mingorance, Javier Mendez-Gomez, Gianfranco Mauro, Encarnacion Castillo-Morales, Manuel Pegalajar-Cuellar, and Diego P. Morales-Santos. Air-writing character recognition with ultrasonic transceivers. *Sensors*, 21(20), 2021.

[20] Yuqi Luo, Jiang Liu, and Shigeru Shimamoto. Wearable air-writing recognition system employing dynamic time warping. In *2021 IEEE 18th Annual Consumer Communications and Networking Conference (CCNC)*, pages 1–6, 2021.

[21] Vangos Pterneas. Finger tracking using kinect v2. `https://pterneas.com/2016/01/24/kinect-finger-tracking/`, 2016.

[22] Qianhong Hu, Zhiwen Yu, Zhu Wang, Bin Guo, and Chao Chen. Vihand: Gesture recognition with ambient light. In *2019 IEEE SmartWorld, Ubiquitous Intelligence and Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pages 468–474, 2019.

[23] Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370. Seattle, WA, USA:, 1994.

[24] Haihan Duan, Miao Huang, Yanbing Yang, Jie Hao, and Liangyin Chen. Ambient light based hand gesture recognition enabled by recurrent neural network. *IEEE Access*, 8:7303–7312, 2020.

[25] Raghav H. Venkatnarayan and Muhammad Shahzad. Gesture recognition using ambient light. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 2(1), mar 2018.

[26] Taiwo Ajani, Agbotiname Imoize, and Prof. Aderemi Atayero. An overview of machine learning within embedded and mobile devices-optimizations and applications. *Sensors*, 21:1–44, 06 2021.

[27] Partha Pratim Ray. A review on tinyml: State-of-the-art and prospects. *Journal of King Saud University - Computer and Information Sciences*, 34(4):1595–1623, 2022.

[28] Stanislava Soro. TinyML for Ubiquitous Edge AI, 2021.

[29] Dr. Lachit Dutta and Swapna Bharali. TinyML Meets IoT: A Comprehensive Survey. *Internet of Things*, 16:100461, 2021.

[30] Ramon Sanchez-Iborra and Antonio F. Skarmeta. Tinyml-enabled frugal smart objects: Challenges and opportunities. *IEEE Circuits and Systems Magazine*, 20(3):4–18, 2020.

[31] Mehmet Görkem Ulkar and Osman Erman Okman. Ultra-low power keyword spotting at the edge. *CoRR*, abs/2111.04988, 2021.

[32] Ahmad Dziaul Islam Abdul Kadir, Ahmed Al-Haiqi, and Norashidah Md Din. A dataset and tinyml model for coarse age classification based on voice commands. In *2021 IEEE 15th Malaysia International Conference on Communication (MICC)*, pages 75–80, 2021.

[33] Igor Fedorov, Marko Stamenovic, Carl Jensen, Li-Chia Yang, Ari Mandell, Yiming Gan, Matthew Mattina, and Paul N. Whatmough. TinyLSTMs: Efficient neural speech enhancement for hearing aids. In *Interspeech 2020*. ISCA, oct 2020.

[34] Mansoureh Lord and Adam Kaplan. Mechanical anomaly detection on an embedded microcontroller. In *2021 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 562–568. IEEE, 2021.

[35] Hui Han and Julien Siebert. Tinyml: A systematic review and synthesis of existing research. In *2022 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pages 269–274, 2022.

[36] Swapnil Sayan Saha, Sandeep Singh Sandha, and Mani Srivastava. Machine Learning for Microcontroller-Class Hardware – A Review, 2022.

[37] Vijay Janapa Reddi, Brian Plancher, Susan Kennedy, Laurence Moroney, Pete Warden, Anant Agarwal, Colby Banbury, Massimo Banzi, Matthew Bennett, Benjamin Brown, Sharad Chitlangia, Radhika Ghosal, Sarah Grafman, Rupert Jaeger, Srivatsan Krishnan, Maximilian Lam, Daniel Leiker, Cara Mann, Mark Mazumder, Dominic Pajak, Dhilan Ramaprasad, J. Evan Smith, Matthew Stewart, and Dustin Tingley. Widening access to applied machine learning with tinyml, 2021.

[38] Colby R. Banbury, Vijay Janapa Reddi, Max Lam, William Fu, Amin Fazel, Jeremy Holleman, Xinyuan Huang, Robert Hurtado, David Kanter, Anton Lokhmotov, David Patterson, Danilo Pau, Jae-sun Seo, Jeff Sieracki, Urmish Thakker, Marian Verhelst, and Poonam Yadav. Benchmarking tinyml systems: Challenges and direction, 2020.

[39] Lennart Heim, Andreas Biri, Zhongnan Qu, and Lothar Thiele. Measuring what really matters: Optimizing neural networks for tinyml, 2021.

[40] P. Warden and D. Situnayake. *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-low-power Microcontrollers*. O'Reilly, 2020.

[41] NXP Semiconductor. GLOW. `https://www.nxp.com/design/software/development-software/eiq-ml-development-environment/eiq-inference-with-glow-nn:eIQ-Glow`, July 2020.

[42] Microsoft Coorperation. Embedded Learning Library. `https://microsoft.github.io/ELL/`, December 2018.

[43] Lucas Tsutsui da Silva, Vinicius MA Souza, and Gustavo EAPA Batista. Embml tool: supporting the use of supervised learning algorithms in low-cost embedded systems. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1633–1637. IEEE, 2019.

[44] Jon Nordby. emlearn: Machine Learning inference engine for Microcontrollers and Embedded Devices, March 2019.

[45] Liangzhen Lai, Naveen Suda, and Vikas Chandra. Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus, 2018.

[46] STMicroelectronics. X-CUBE-AI. `https://www.st.com/en/embedded-software/x-cube-ai.html`, July 2022.

[47] Marco Giordano, Luigi Piccinelli, and Michele Magno. Survey and comparison of milliwatts micro controllers for tiny machine learning at the edge. In *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 94–97, 2022.

[48] Atis Elsts and Ryan McConville. Are microcontrollers ready for deep learning-based human activity recognition? *Electronics*, 10(21), 2021.

[49] Bharath Sudharsan, Simone Salerno, Duc-Duy Nguyen, Muhammad Yahya, Abdul Wahid, Piyush Yadav, John G. Breslin, and Muhammad Intizar Ali. Tinyml benchmark: Executing fully connected neural networks on commodity microcontrollers. In *2021 IEEE 7th World Forum on Internet of Things (WF-IoT)*, pages 883–884, 2021.

[50] Fouad Sakr, Francesco Bellotti, Riccardo Berta, and Alessandro De Gloria. Machine learning on mainstream microcontrollers. *Sensors*, 20(9), 2020.

[51] Norah N. Alajlan and Dina M. Ibrahim. TinyML: Enabling of Inference Deep Learning Models on Ultra-Low-Power IoT Edge Devices for AI Applications. *Micromachines*, 13(6), 2022.

[52] XMOS. *XU316-1024-FB265 Datasheet*, 7 2021.

[53] Antonio Pullini, Davide Rossi, Igor Loi, Giuseppe Tagliavini, and Luca Benini. Mr.wolf: An energy-precision scalable parallel ultra low power soc for iot edge processing. *IEEE Journal of Solid-State Circuits*, 54(7):1970–1981, 2019.

[54] Allan Skillman and Tomas Edsö. A technical overview of cortex-m55 and ethos-u55: Arm's most capable processors for endpoint ai. In *2020 IEEE Hot Chips 32 Symposium (HCS)*, pages 1–20, 2020.

[55] Advait Jain et. al. TensorFlow Lite for Microcontrollers (TFLM). `https://github.com/tensorflow/tflite-micro`, 2022.

[56] Tom. O'Malley, etc. Kerastuner. `https://github.com/keras-team/keras-tuner`, 2019.

[57] Lisha Li and et. al. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 2018.

[58] Texas Instruments. *OPT101 Monolithic Photodiode and Single-Supply Transimpedance Amplifier*, 1994. Rev. B.

[59] Texas Instruments. *AN-1803 Design Considerations for a Transimpedance Amplifier*, 2008.

[60] Application Report: Stability Analysis of Voltage Feedback Op Amps. Technical Report SLOA020A, 2001.

[61] Holger Vogt, Marcel Hendrix, Paolo Nenzi, and Dietmar Warning. ngspice - the open source spice simulator for electric and electronic circuits.

[62] Art Kay and Tim Green. Analog Engineer's Pocket Reference. Technical report, Texas instruments, 2019.

[63] Yuxi (Hayden) Liu. *Python Machine Learning by Example - Third Edition*. 10 2020.

[64] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[65] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[66] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *CoRR*, abs/1506.04214, 2015.

Appendix A

PCB design files

A.1 Electrical schematics

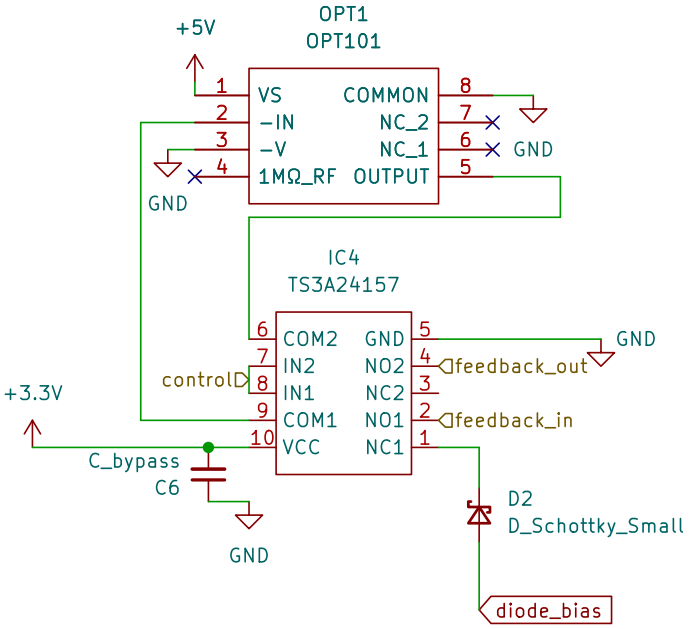


Figure A.1: **OPT101 with switch.**

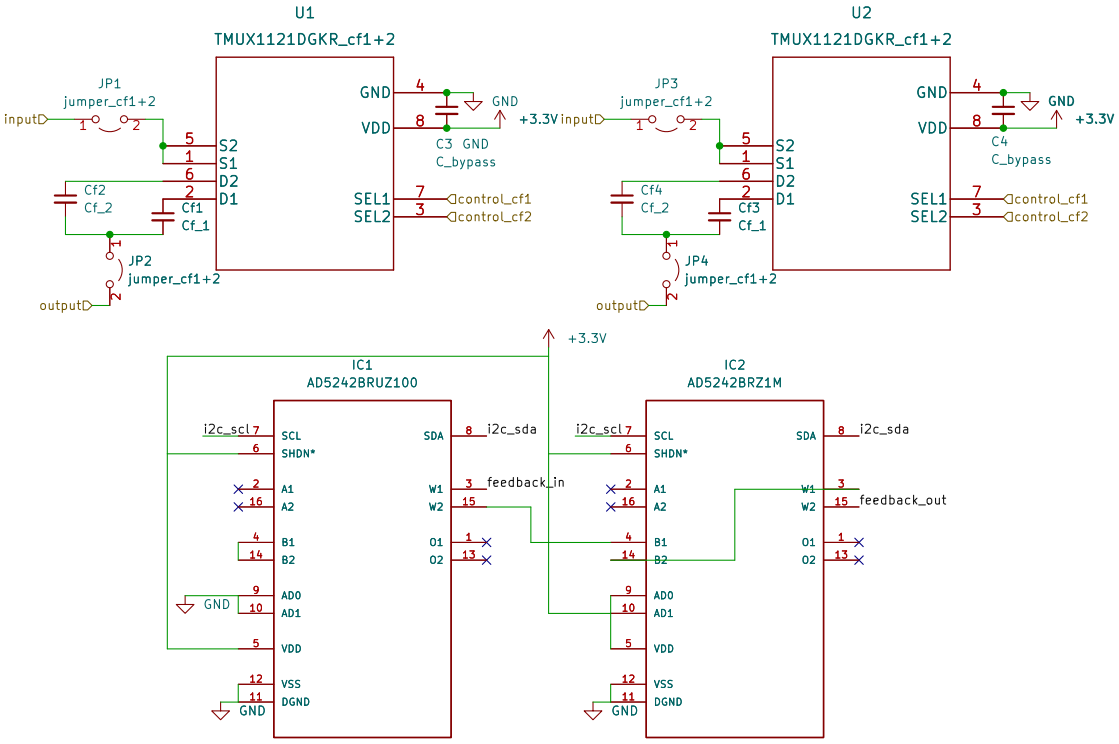


Figure A.2: **Feedback network.**

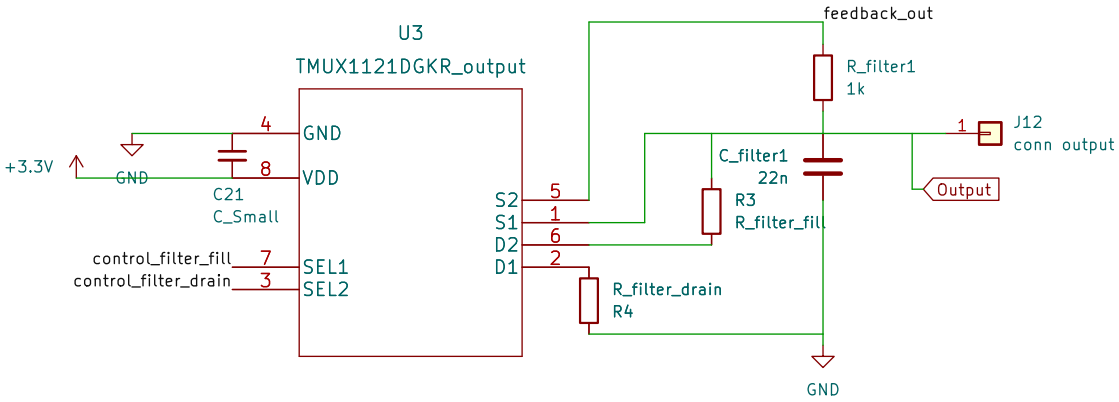


Figure A.3: **Output filter.**

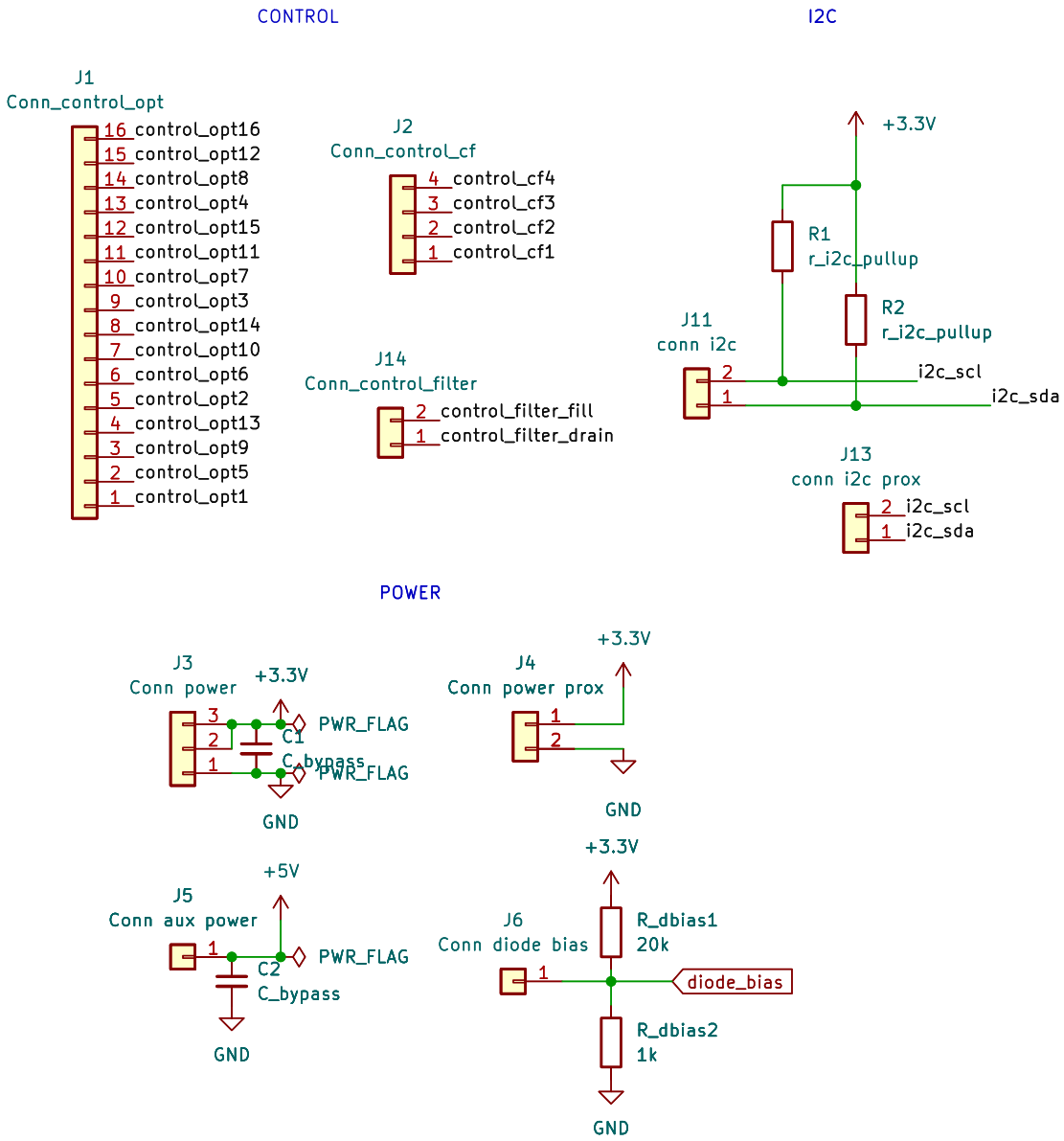


Figure A.4: **Connectors.**

A.2 PCB layout

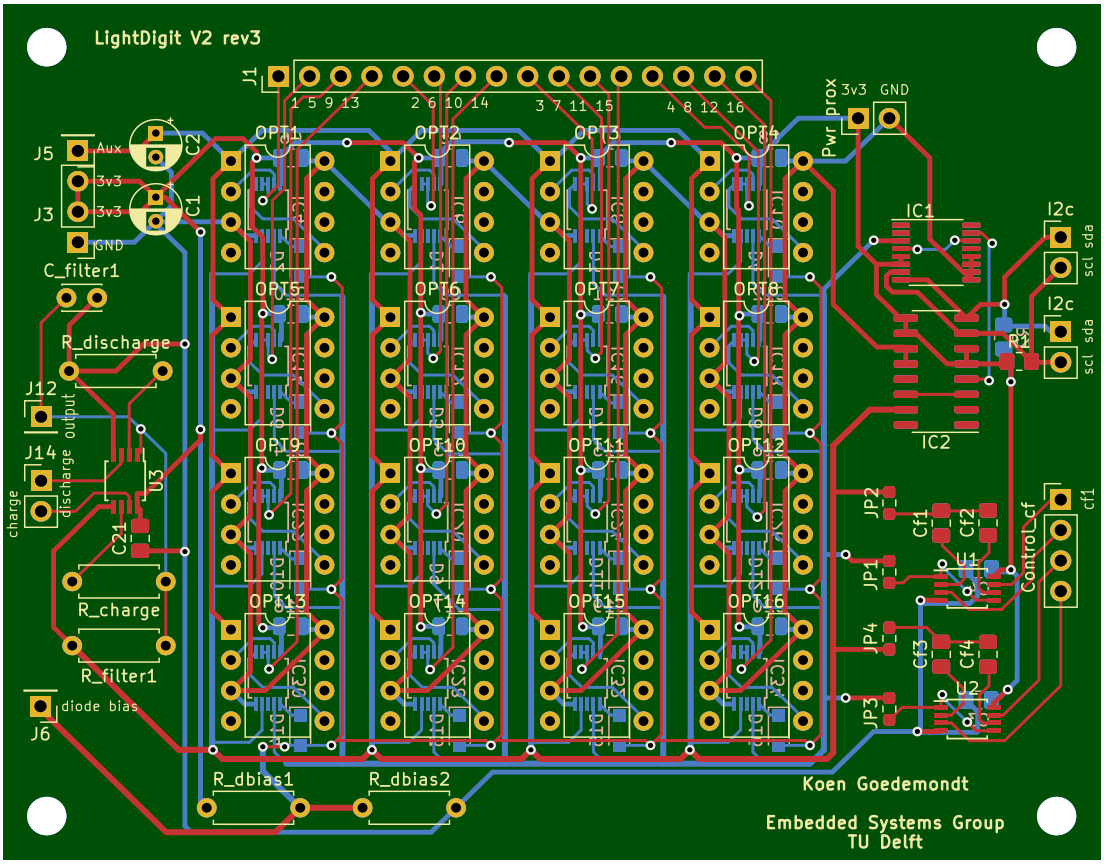


Figure A.5: **PCB track layout.**

Appendix B

SPICE simulations

B.1 Simulation layouts

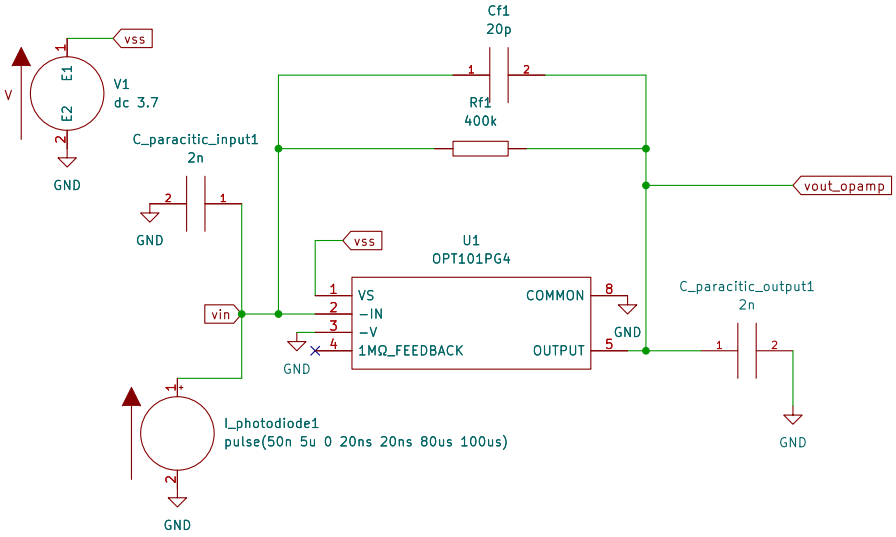


Figure B.1: **Simulation layout OPT101.**

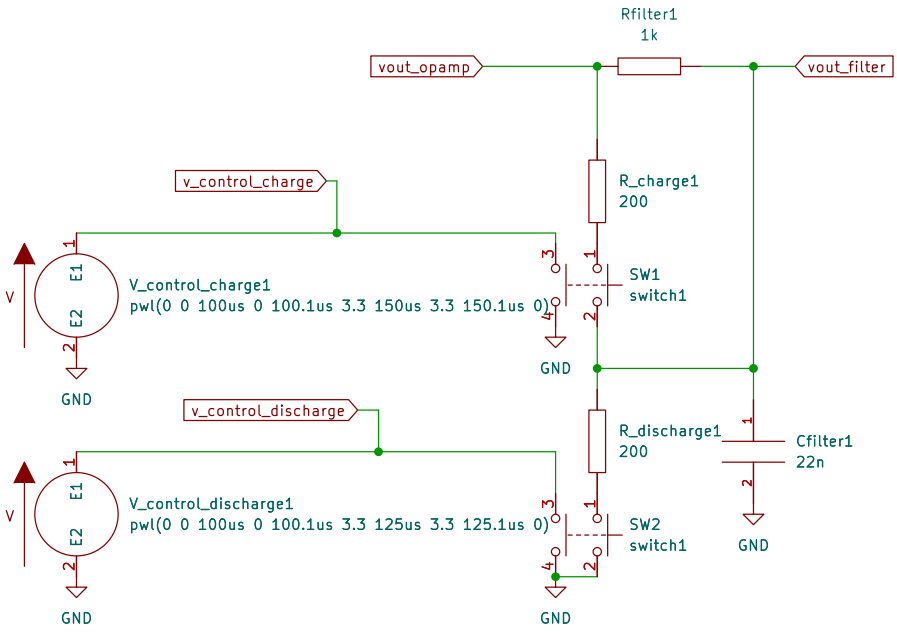


Figure B.2: **Simulation layout output filter.**

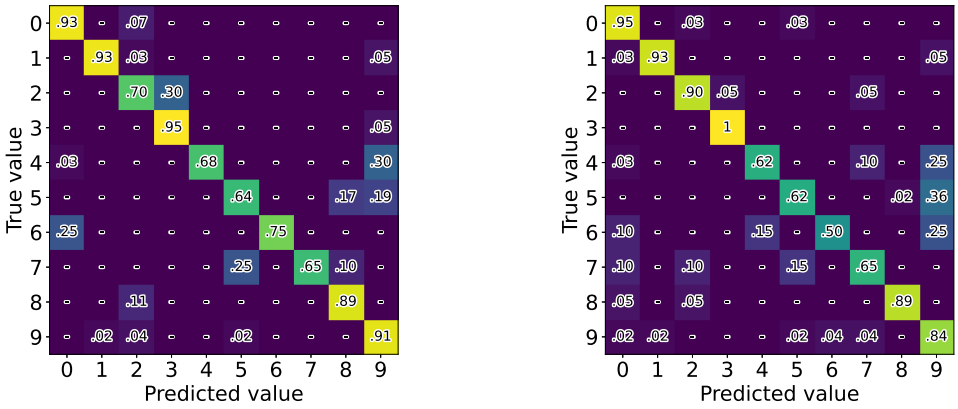
B.2 OPT101 SPICE model

```
*  OPT101P,W SPICE MACROMODEL-- Copyright 1994 Burr-Brown Corp.
*  Bandwidth: 14kHz,  Iq = 0.12mA, Single-Supply: +2.7 - +36V
*  REV.A   12-13-94  EM
*
* Connections:   vs
*                |  inverting input
*                |  |  vpin3
*                |  |   |  feedback
*                |  |   |  |    vout
*                |  |   |  |     |  common
 *                |  |   |  |     |    |
.subckt OPT101   vs in vpd feed vout com
*
  c1   11 12 1E-12
  c2    6  7 1E-12
  cee  10 99 9E-12
  cpar in feed 5.5e-12
  cf   in vout 9e-12
  cphoto in com 1200e-12 ; photodiode capacitance
  dc   vout 53 dx
  de   54  vout dx
  dlp  90 91 dx
  dln  92 90 dx
  dp    4 vs dx
  egnd 99  0 poly(2) (vs,0) (4,0) 0 .5 .5
  fb    7 99 poly(5) vb vc ve vlp vln 0 1.061E9 -1E9 1E9 1E9 -1E9
  ga    6  0 11 12 22.5E-6
  gcm   0  6 10 99 314.2E-12
  iee  vs 10 dc 10.00E-6
  hlim 90  0 vlim 1K
  q1   11  in 13 qx
  q2   12  1 14 qx
  jpd  ed vout vout jp ; JFET pulldown
  r2    6  9 100e3
  rc1   44 11 79.58E3
  rc2   44 12 79.58E3
  re1  13 10 74.40E3
  re2  14 10 74.40E3
  ree  10 99 20.00E6
  ro1   8 vout 3
  ro2   7 99 3
  rp   vs 4 2.8e6
  rf   in feed 1e6
  vb    9  0 dc 0
  vc   vs 53 dc 1.815
  ve   54  4 dc 0
  vlim  7  8 dc 0
```

```
  vlp  91  0 dc 15
  vln   0 92 dc 15
  vls  4 44 dc 0.7
  vped 1 com dc 7.5e-3
  vbulk 4 0 dc 0
  vdrain ed vpd dc 0
  iq   vs 4 dc 100u
  fdrain vs 4 vdrain 1
.model dx D(Is=800.0E-18)
.model qx PNP(Is=800.0E-18 Bf=25.00E3)
.model jp PJF(VTO=-1.2 BETA=72U LAMBDA=7M)
.ends
```

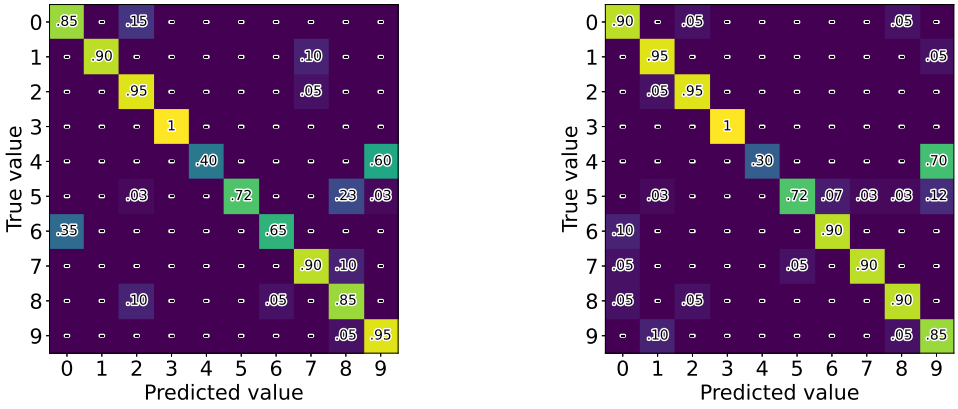
Appendix C

Test setup confusion matrices



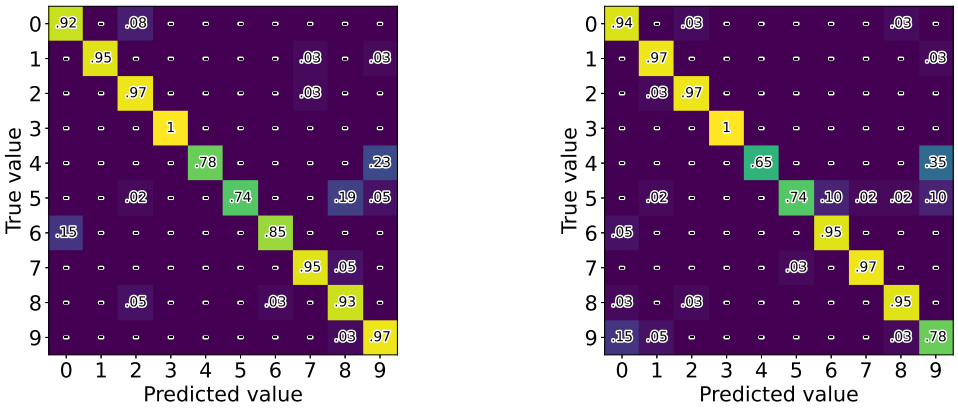
(a) **CNN 500 lux** (b) **ConvLSTM 500 lux**

Figure C.1: **Setup 1 confusion matrices.**



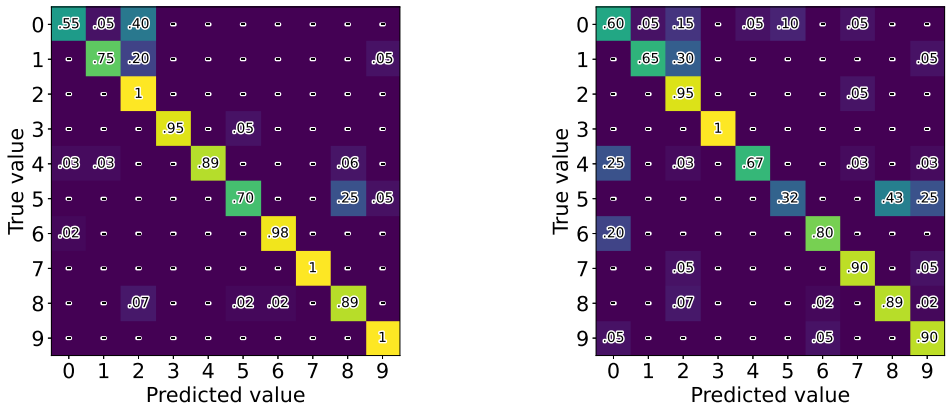
(a) **CNN 1000 lux** (b) **ConvLSTM 1000 lux**

Figure C.2: **Setup 1 confusion matrices.**



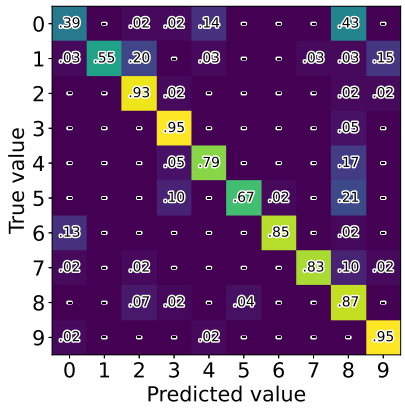
(a) **CNN 5000 lux** (b) **ConvLSTM 5000 lux**

Figure C.3: **Setup 1 confusion matrices.**

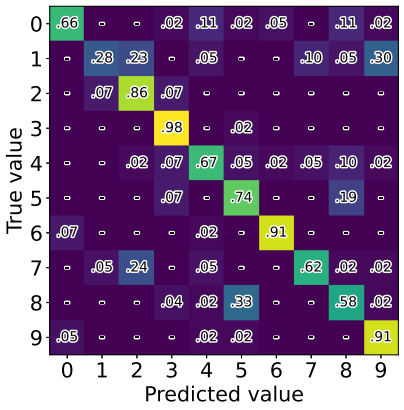


(a) **CNN θ=15°** (b) **ConvLSTM θ=15°**

Figure C.4: **Setup 2 confusion matrices.**

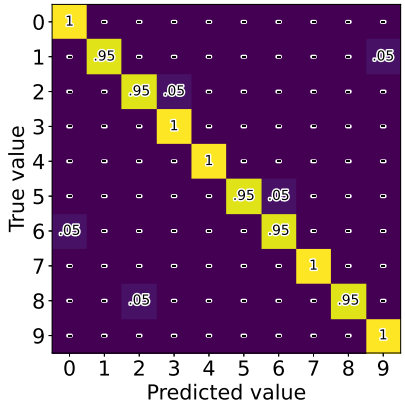


(a) **CNN θ=30°**

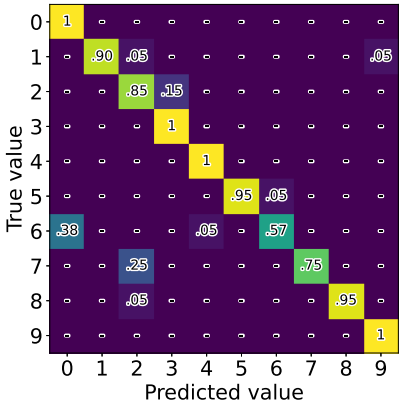


(b) **ConvLSTM θ=30°**

Figure C.5: **Setup 2 confusion matrices.**

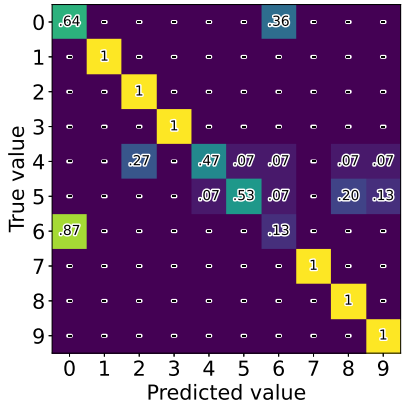


(a) **ConvLSTM φ=45°**

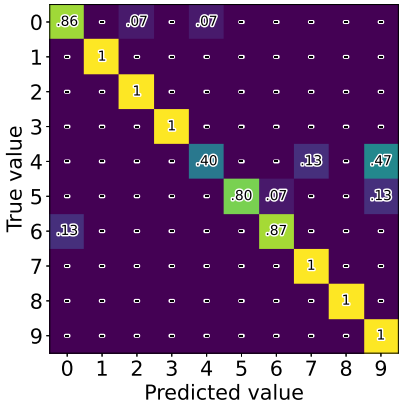


(b) **ConvLSTM φ=45°**

Figure C.6: **Setup 3 confusion matrices.**



(a) **CNN 10k lux**



(b) **ConvLSTM 10k lux**

Figure C.7: **Setup 4 confusion matrices.**

Appendix D

NUCLEO-H743ZI2 pins

Table D.1: **NUCLEO-H743ZI2 pin assignment.**

Pin name	Connector	Pin
OPT101_1	CN11	PD4
OPT101_2	CN11	PC1
OPT101_3	CN11	PG3
OPT101_4	CN11	PF2
OPT101_5	CN11	PD5
OPT101_6	CN11	PC0
OPT101_7	CN11	PE2
OPT101_8	CN11	PF8
OPT101_9	CN11	PD6
OPT101_10	CN11	PD3
OPT101_11	CN11	PE4
OPT101_12	CN11	PF9
OPT101_13	CN11	PD7
OPT101_14	CN11	PG2
OPT101_15	CN11	PE5
OPT101_16	CN11	PG1
CF1_10pf	CN11	PE6
CF2_22pf	CN11	PG15
CF3_10pf	CN11	PG10
CF4_47pf	CN11	PG13
COUT_DISCHARGE	CN11	PD2
COUT_CHARGE	CN11	PC11
VMIN_DISCHARGE	CN11	PG11
I2C1_SDA	CN9	PB7
I2C1_SCL	CN9	PB6
ADC_IN	CN10	PF5
APDS9930_INT	CN10	PE8