Towards a low-cost air-written character recognition system

Designing an embedded machine learning system to recognise the first 10 letters of the Latin alphabet using 3 photodiodes

by

Paco Jeraldo Pronk

Project duration:	April, 2023 – July, 2024	
Thesis committee:	Dr. ir. Q. Wang,	TU Delft, supervisor
	Ir. R. Zhu,	TU Delft
	Ir. M. Yang,	TU Delft

Style: TU Delft Report Style, with modifications by Daan Zwaneveld

Abstract

This study introduces a novel system that leverages three photodiodes and ambient light to identify air-written characters on a resource-constrained device. Through experimentation, suitable methods of data preprocessing, machine learning and model compression were selected to recognize the first 10 characters of the Latin alphabet. The final system was able to recognize these characters with a *between*-participant accuracy of 50.80% and a *within*-participant accuracy of 67.82%.

Contents

1 Introduction

1.1. General introduction

The development of touchless interfaces has been an area of interest in human-computer interaction research for several decades. These interfaces aim to provide users with a way to interact with technology without the need for physical touch, offering a more intuitive and immersive experience. One of the earliest examples of a touchless interface is the wired glove or data glove, which was developed in the 1980s [1]. Since then, touchless interfaces have continued to evolve and find applications in various domains, including healthcare, automotive, retail, and smart homes. Moreover, touchless interfaces have made significant appearances in the entertainment industry, as exemplified by the Microsoft Kinect [2] and the Nintendo Wii [3]. These innovations not only demonstrate the promise of touchless interfaces but also highlight their potential in making human-computer interaction more accessible, immersive and intuitive.

Another significant advantage of touchless interfaces became apparent during the COVID-19 pandemic. The global health crisis prompted a substantial increase in the development and adoption of touchless interfaces [2]. These interfaces, utilizing gestures, motion sensors and hand interaction, offered a solution to reduce the spread of harmful bacteria and viruses by eliminating the need for physical contact. With their contactless nature, touchless interfaces became a valuable tool in halting the transmission of the COVID-19 virus and other infectious diseases. Consequently, touchless interfaces could be used to prevent viral infections which could save lives.

One method of creating touchless interfaces that was researched during the COVID pandemic was the use of ambient light to detect in-air-written digits. This approach leverages the variations in ambient light caused by hand movements while writing in the air. By capturing and analyzing these light patterns, sophisticated machine learning models can accurately recognize air-written digits [3]. Building upon this research, the focus of this paper is to implement a system capable of recognizing the first 10 characters of the Latin alphabet when written in the air, using a resource-constrained device. The system will employ ambient light and three photodiodes to detect air-written characters, utilize a preprocessing pipeline to extract meaningful features and employ a Convolutional Neural Network for character recognition. Throughout this work, the goal is to contribute to the advancement of touchless interface technologies, enabling intuitive and contactless interactions while addressing the limitations of resource-constrained devices.

1.2. Challenges

To create a fully functional system that can detect air-written characters four main challenges have to be overcome:

- **Creating a dataset of air-written characters**
  Currently, there exists no dataset of air-written characters captured by photodiodes. While there are datasets for written characters such as EMNIST [4], in air gestures [5] and air-written digits [3], these do not contain data specifically for air-written characters. Thus a new dataset has to be constructed which can be used to train a machine-learning model.

- **Developing a suitable data preprocessing and data structuring algorithm**
  Raw data collected from the three photodiodes has to be preprocessed and structured in such

a way that it can be introduced to a machine-learning model. The most optimal to preprocess raw data as well as the most optimal way to structure it for a machine learning model have to be investigated.

- **Constructing a machine learning model that can identify air-written characters**
  A machine learning model that can identify air-written characters has to be constructed. Within the construction of the model, a balance needs to be achieved between accuracy and latency. This balance should ensure that the model operates with high accuracy while maintaining a latency that aligns with real-time capabilities.
- **Compression of this machine learning model to run on a resource-constrained device**
  To allow for a system that can run independently on resource-constrained devices such as an Arduino Nano 33 BLE the machine learning model has to be compressed. The model size and the rate at which it can be compressed are heavily dependent on memory resources available on the Arduino Nano 33 BLE. Here another balance needs to be found between accuracy, latency and model size.

1.3. Contributions

The contributions towards a complete air-written character recognition system posed by this research are as follows:

- **Successfully created an air-written character dataset**
  During the course of this research, a dataset containing air-written character samples captured using 3 photodiodes was created. The final dataset consists of 30 participants, with each participant writing the characters A to J in air 5 times with their dominant hand. This resulted in a dataset with 1500 samples and a 13.33% left-handed and 86.67% right-handed split.
- **Developed a robust and effective data preprocessing pipeline**
  To preprocess raw data captured by photodiodes a preprocessing pipeline was created. This pipeline first strips regions where no shadow is detected from the raw data then uses Z-score normalization to standardize the data and finally smooths the data using moving window averaging. The data preprocessing pipeline is able to increase the *within*-participant accuracy with 36.45% between simple normalization and the full data preprocessing pipeline when used on the optimal model.
- **Discovered an optimal model for identifying air-written characters**
  Using a generative hyperparameter search, an optimal model for the air-written character identification problem was found. The ClearCNN model consists of two convolutional layers, followed by a spatial dropout layer, a max pooling layer and two densely connected layers. ClearCNN is able to reach a *between*-participant accuracy of 50.80% and a *within*-participant accuracy of 67.82%.
- **Successfully performed model compression using TensorflowLite**
  Model quantization with the TensorflowLite framework was used to reduce the model size such that it could be used on a resource-constrained device. Using dynamic range conversion the ClearCNN model could be compressed by a factor of 3.8, effectively reducing the size from 412 KB to 108 KB.

2
Background

2.1. Previous research

To fully comprehend the context of this research paper and its implications it is necessary to understand that previous research was conducted on this topic at the TU Delft in the 2021/2022 Research Project. The problem posed in the previous iteration of this project was aimed at designing a system to recognise in-air gestures using 3 photodiodes on an Arduino Nano 33 BLE. The design phase of this system was split into 5 parts, with each of these parts being addressed by different students in the research project. The contributions of these students will be listed below. Important results and design choices from their research will be highlighted and the implications for this research will be explained.

- **Hardware design and optimal positioning of photodiodes for gesture recognition**
  In the paper written by Stijn van de Water[6] the design of a hardware receiver to capture ambient light is considered. Crucial results from this paper include the design of a printed circuit board, a calibration mechanism for multiple light conditions and optimal placement of 3 photodiodes for gesture recognition. The PCB designed in the 2021/2022 iteration of the research project is used directly in this project. The placement of the photodiodes and the light calibration mechanism are also utilized.
- **Development of a dataset containing gesture samples**
  The contributions of Femi Akadiri [5] consisted of a dataset containing gesture samples under multiple lighting conditions. Important results of this research were the effect of lighting conditions on sensor values and the absence of the effect of different hand sizes on sensor readings. The 100Hz sampling rate proposed in the 2021/2022 iteration of the project is also used in this iteration. The dataset created by Akadiri is not used seeing as this dataset contains gesture samples and does not contain air-written character samples.
- **Design of a software receiver to preprocess raw photodiode signals**
  To handle incoming photodiode signals and convert these signals to a data sample on which interference could be run, Dimitar Barantiev [7] created a software receiver that could be run on an Arduino Nano BLE. The software receiver proposed consisted of an algorithm to perform gesture edge detection and an algorithm to perform signal processing. The latter algorithm consists of a combination of Fast Fourier Transform, Linear Interpolation and normalization. The contributions made in Barentiev's paper offered a starting point for the preprocessing steps evaluated in this paper.
- **Constructing Convolutional Neural Networks to recognise in air gestures**
  One way to recognise gestures based on preprocessed data is with the use of Convolutional Neural Networks (CNN). The use of CNNs for gesture recognition is explored in the paper by William Narchi [8]. The most important results of this paper were the 2D structuring of photodiode data, the difference between *between-* an *within-* participant study design[1] and the best CNN model architecture. The paper reported a final accuracy of between 75.4% and 86.8% with a *within-*participant design.

[1]Please refer to Section 4.1 for the difference between these study designs

- **Cunstructing Recurrent Neural Networks to recognise in air gestures**
  Another avenue explored by the 2021/2022 iteration of the research project was the use of Recurrent Neural Networks to perform gesture recognition. The paper by Matthew Lipski [9] explores this avenue and reports a final accuracy of 43 percent with a *between*-participant design. Within this research paper, the paper by Lipski is the least relevant, because it explores a different Neural Network architecture.

2.2. 1-Dimensional Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a form of deep learning, more specifically, a CNN is a deep feed-forward neural network which uses convolutions kernels to extract features from input data. Classically, CNNs have been used in tasks such as image recognition, object detection, and semantic segmentation, where spatial relationships and local patterns are crucial for accurate predictions. However, recently 1-Dimensional Convolutional Neural Networks (1DCNNs) have seen increased utilization in time series classification, providing state-of-the-art performance on problems such as real-time patient-specific electrocardiogram (ECG) monitoring [10], ECG anomaly detection [11] and power grid switch fault detection [12]. These quite recent developments in time series classification provide a clear indication that 1DCNNs could be an important tool in air-written gesture recognition.

2.2.1. Basic building blocks: CNN Layers

Convolutional Layers

The core building block of 1DCNNs are convolutional layers. These layers apply a set of trainable filters (kernels) to an input time series. In Figure 2.1, an example of a 1D convolution on a 7×1 input vector is depicted. The kernel slides from left to right over the input vector, repeatedly performing element-wise multiplications and aggregating the results to produce an output vector.

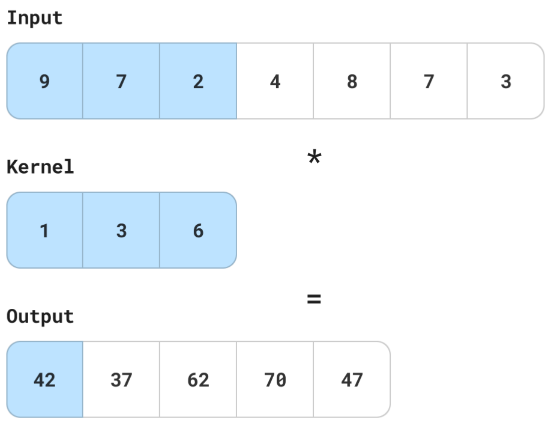

Figure 2.1: A 3x1 kernel applied to a 7x1 input vector. The output is a 5x1 vector.

The resulting output vector is called a feature map and is created by performing the following multiplications:

$$(9 \times 1) + (7 \times 3) + (2 \times 6) = 42$$
$$(7 \times 1) + (2 \times 3) + (4 \times 6) = 37$$
$$(2 \times 1) + (4 \times 3) + (8 \times 6) = 62$$
$$(4 \times 1) + (8 \times 3) + (7 \times 6) = 70$$
$$(8 \times 1) + (7 \times 3) + (3 \times 6) = 47$$

The kernel weights, which are depicted in red, are trained by the Convolutional Neural Network when the network is trained on a dataset.

Dropout layers and Spatial Dropout layers
One of the benefits of CNNs is that they can learn patterns from data extremely quickly. This however is also one of the downsides of CNNs, causing a tendency to overfit. To combat this downside, dropout layers can be used. Dropout layers work by randomly dropping units and their connections from neural networks [13]. This prevents the model from using neurons to remember training data because these neurons might drop out during each training step. In the context of CNNs, a dropout layer removes random positions from a feature map.

Spatial Dropout layers are dropout layers that are specifically designed with CNNs in mind. Spatial Dropout layers perform operations similarly to Dropout layers, but instead of removing positions from feature maps randomly, they remove entire feature maps at random. This is done to promote independence between feature maps [14].

Pooling layers
Pooling layers are used to shrink the input size and prevent overfitting. They condense data by selecting either the maximum or average values in local regions. *Max* Pooling keeps the most important features, while *Average* Pooling calculates the mean. Pooling layers reduce complexity, capture important patterns, and make models more robust to noise.

2.2.2. Key components of CNNs

With the previously mentioned layers in mind, it is crucial to provide a concise overview of the key components of Convolutional Neural Networks (CNNs). These key components, or hyperparameters, can be varied to create appropriate models for different problems.

Kernel
As mentioned earlier, the kernel serves as a crucial component in a Convolutional layer. It slides over the input, performing mathematical operations and generating a feature map as the output. Within a Convolutional layer, a kernel size can be specified, which determines the number of trainable parameters in a kernel. In Figure 2.1 a kernel size of 3×1 is specified, but kernel sizes of 5×1 and 7×1 are equally viable options.

Filters
The number of filters specifies the number of kernels that are trained per Convolutional layer. Each filter, along with its corresponding kernel, captures distinct features from the input. The output of every filter is thus a distinct feature map.

Stride
The stride value describes how much the kernel moves after each convolutional step. The most common value for stride is 1. With a stride value of 1, the kernel moves along one column after every convolution. With a stride value of 2, the kernel moves 2 columns. Changing the stride size can cause a kernel to extract different features.

3 Methodology

This section provides an overview of the system as a whole as well as the methodology used to tackle the specific challenges associated with each part of the system. Section 3.1 will provide a clear overview of the complete system. Section 3.2 will provide an overview of the hardware used in the system. The method for creating an air-written character dataset will be explained in Section 3.3. Section 3.4 will explain the details of the preprocessing pipeline. Section 3.5 discusses the search for the optimal CNN architecture. Finally in Section 3.6 a brief insight into model quantization and compression will be given.

3.1. System overview

The system designed in this paper consists of five parts. A hardware receiver to capture photodiode values, a software receiver to transform the values into workable signals, a preprocessing pipeline to extract meaningful features from these signals, a trained and compressed CNN and a dataset used to train the aforementioned CNN. A schematic overview of the system can be seen in Figure 3.1

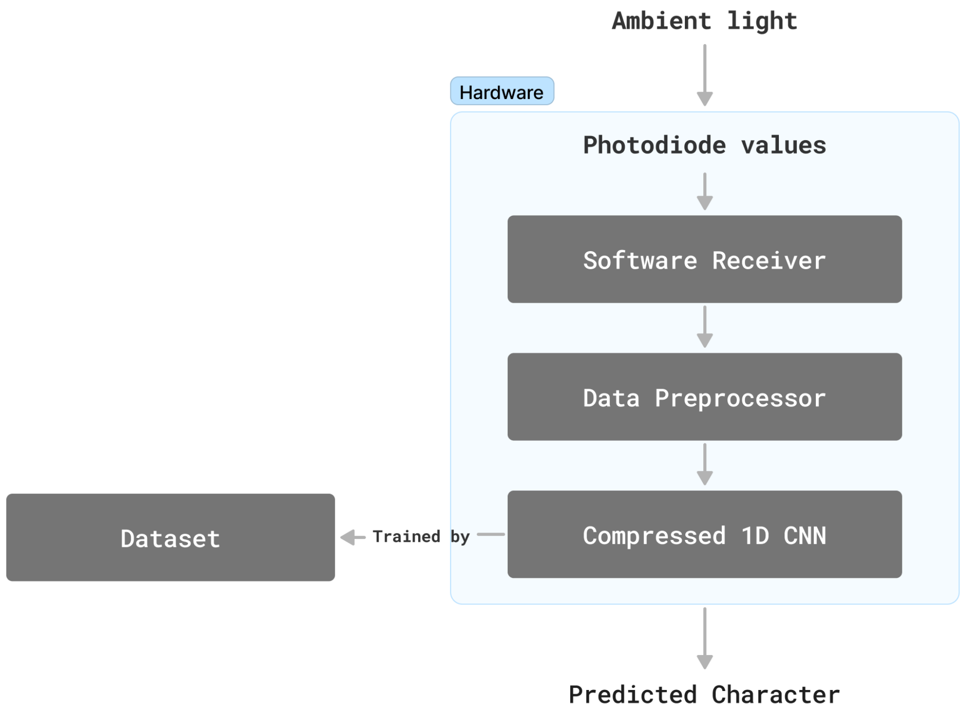

Figure 3.1: An overview of the inference pipeline for the character recognition system proposed in this paper. Raw data is first captured by the hardware, next it is preprocessed and restructured, then it is evaluated by a compressed 1DCNN and the prediction of this network is chosen as the output letter. The compressed 1DCNN is trained by an air-written character dataset.

3.2. Hardware

The hardware used in this study is the PCB created in the previous iteration of the research project. The system consists of 3 OPT101 photodiodes, an Arduino Nano 33 BLE and variable resistors to cope with different lighting conditions. The full specifications can be found in [6].

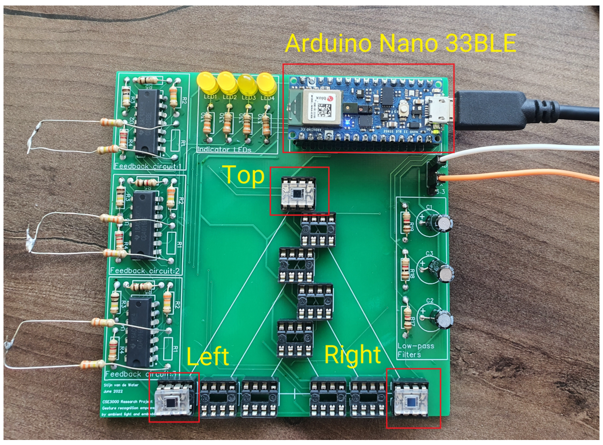

Figure 3.2: The PCB used for the air-written character recognition system. The Arduino and the 3 photodiodes are highlighted.

3.3. Dataset

Data was collected using the hardware setup mentioned in Section 3.2. The hardware setup was placed in a well-lit room in the faculty of Industrial Design of the TU Delft. During the experiment, the lighting conditions were carefully controlled to ensure minimal sensor value variance within the dataset. This was done by monitoring the sensor values and ensuring that the starting threshold values did not stray too far from a set value. Air-written character data was collected for 4 seconds at a sampling frequency of 100Hz.

At the beginning of the experiment, participants were asked to write the capital letters A to J on a piece of paper. This was done to ensure that participants wrote only capital block or print letters. If the participants did not write block letters they were asked to write the letter again as a block letter. Furthermore, a "writing area" was specified, to keep the air-written character between the bounds of the photodiodes. This writing area was defined as the square that encompassed all photodiodes. Participants were also asked to keep their finger $\pm 2cm$ above the writing area.

With these instructions, participants were asked to write each letter from A to J above the photodiode array 5 times. If a collected sample seemed to deviate from what was expected, the sample was removed and the participant was asked to repeat the character. This was done if the participant had started moving above the photodiode array too quickly, too late, or if the shadow of the hand did not activate all photodiodes.

3.4. Data preprocessing

The raw data per character collected by the 3 photodiodes can be seen as a matrix $X_{t,c}$ with dimensions $t \times c$ where t represents the number of samples and c represents the number of channels. Since data is collected for 4 seconds at a sampling rate of 100Hz using 3 photodiodes the resulting raw data is comprised of a 400×3 matrix. To extract meaningful features from the collected data, different methods of data preprocessing were evaluated. The resulting data preprocessing pipeline is as follows.

1. Edge removal
The first step of the data preprocessing pipeline handles *edge removal*, which is the removal of parts of the signal where no shadow is detected. This is done by removing values from the start of a signal and from the end of a signal. Values are removed from either side of the data until a certain threshold value γ is reached. Here for each channel c the threshold value γ_c is calculated as $\gamma_c = \alpha * \max(X_{400,c})$. Experimentation on the collected data revealed that an α value of 0.95 resulted in the best edge removal.

The threshold is calculated for each photodiode separately and thus results in varying length signals for different photodiodes. To make sure the edge-removed data can be used by a neural network it is transformed back to 400 samples per photodiode using linear interpolation[1]. This method of edge removal is inspired by the methods used in [7] [3]

2. Data scaling
The next step in the data preprocessing pipeline is to scale the data such that any difference in sensitivity between photodiodes is mitigated. To this end, the raw data is scaled using Z-score normalization also called standardization. Various other data scaling techniques such as normalization and standardization followed by normalization were also considered for data scaling. However, when training the 1DCNN model standardization proved to generate the best results in terms of validation accuracy and validation loss.

3. Data smoothing
The final step of the data preprocessing pipeline consists of a smoothing algorithm to remove noise from the acquired signals. The smoothing algorithm chosen for this step consists of moving average smoothing with a window size of 5. This method was chosen because it provided an algorithm with an appropriate balance between speed and smoothing power.

Figure 3.3 shows raw data before and after going through the data preprocessing pipeline.

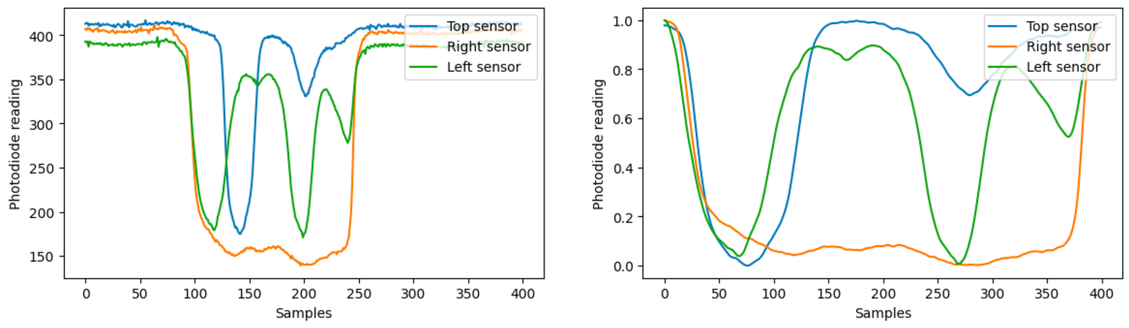

Figure 3.3: The effects of data preprocessing. On the left a raw data sample is plotted. On the right a data sample is plotted after applying edge removal, data scaling and data smoothing.

3.5. CNN Model Architecture Selection and Optimization

The accurate classification of air-written characters in the proposed system depends on the success of the 1DCNN constructed to perform the classification task. The success of this 1DCNN is in turn dependent on finding the best model architecture to fit the problem of recognizing air-written characters. The most straightforward approaches to finding a suitable model architecture are choosing a reference model and adapting it to the problem at hand or constructing and customizing an entirely new architecture. [15] To find an appropriate architecture a reference architecture was chosen and hyperparameters were optimized using the KerasTuner [16]. The KerasTuner is a hyperparameter optimization framework that allows for quick and easy hyperparameter optimization. The KerasTuner was used with GridSearch and an early stopping mechanism with a patience of 5, aimed at validation loss.

Generation 1
The first generation was used to select the number of filters, the kernel sizes and the number of Convolutional layers. The optimization goal for this generation was fixed on validation loss. A summary of the search space during this generation is as follows:

- Total search space: $9 + 9^2 + 9^3 + 9^4 = 7380$ combinations
- Convolutional layers: $[1, 2, 3, 4]$
- Kernel sizes: $[3 \times 1, 5 \times 1, 7 \times 1]$

[1]Linear interpolation is performed with `numpy.interp` from the `numpy` python package.

- Filter sizes: [4, 8, 16]

Generation 2
During the search in the first generation, it was noticed that the validation loss of the first generation was relatively high compared to the validation accuracy. To solve this a spatial Dropout layer was introduced after each Convolutional layer. The goal of this generation was to find an appropriate dropout rate for each Dropout layer. The optimization goal for this generation was fixed on validation loss. A summary of the search space during this generation is as follows:

- Total search space: 25 combinations
- Dropout rates: [0.0, 0.2, 0.4, 0.6, 0.8]

Generation 3
The third generation was used to determine the effects of changing the units in the densely connected layer.

- Total search space: 4 combinations
- Units: [8, 16, 32, 64]

The results of the hyperparameter search can be found in 4.2.

3.6. Model compression

To successfully deploy a machine learning model on an Arduino Nano 33 BLE the model needed to be compressed. To this end, the TensorflowLite framework was used [17]. The TensorflowLite framework is a framework for deploying machine learning models on embedded systems. The compression technique employed by the TensorflowLite framework is post-training quantization. Post-training quantization is the process of reducing the precision and range of numerical values in the model. This is done by converting 64 bit model weights to 32 bit or even integer model weights. Quantization reduces memory and computational requirements, enabling the model to fit and operate efficiently on resource-constrained devices.

4 Results

In the upcoming chapter, an evaluation of the designed system will be conducted. The assessment begins with Section 4.1, where the resulting dataset is presented and analyzed. Section 4.2 provides detailed information on the outcomes of model architecture selection and hyperparameter optimization. Subsequently, in Section 4.3, the model resulting from the model optimization approach is showcased. The effects of different preprocessing methods are discussed in Section 4.4, while Section 4.5 offers an insight into how the trained CNN model can be compressed.

4.1. Resulting dataset

The resulting dataset consists of data collected from 30 participants. Each letter from A to J was recorded 5 times using the participant's dominant hand. This means the dataset consists of $30 \times 10 \times 5 = 1500$ air-written character samples. Within the dataset, there is a split between left-handed and right-handed participants. Four participants were left-handed and 26 were right-handed resulting in a 13.33% left-handed and 86.67% right-handed split. This falls roughly in line with the global average handedness [18]. The dataset consists of a rough 50/50 split between male and female participants. The resulting dataset was split into a training set and a holdout test set with a 0.8 to 0.2 split respectively. Furthermore, the dataset was split into two different study designs following this 0.8 to 0.2 split.

While using a 0.8 to 0.2 split, different study designs were considered. The first study design is the *between*-participant design. In this design, the dataset is split up in such a way that no participant present in the test set is present in the training set and vice versa. The second design is the *within*-participant design. Here the dataset is split up using random shuffle. This means that data gathered from a single participant could be in the training and test sets.

These two designs were chosen to illustrate two different system uses. In the *within*-participant case the system is geared towards use in a controlled environment such as a company or university. Here the air writing styles of the participants are already known. The *between*-participant case is the more general case, where the system does not know anything about the person writing a character in air. During the course of this research, the *within*-participant design is used for hyperparameter tuning, while both designs are used for evaluation.

4.2. Model Architecture Selection and Optimization

The search for the optimal model architecture and hyperparameters resulted in the following improvements to the base model.

Generation 1

In generation 1 the optimal shape of the Convolutional layers in the network was investigated. The search yielded that two Convolutional layers operating with a kernel size of 3×1 yielded the optimal architecture. The first Convolutional layer operates with 4 filters, while the second layer operates with 16 filters.

Generation 2

In generation 2 the optimal positioning of Dropout layers and the respective dropout rate was investigated. The search resulted in a single dropout layer with a dropout rate of 0.8 after the aforementioned Convolutional layers.

Generation 3
In generation 3 the optimal unit size for the final dense layer was investigated. The search concluded with a total unit size of 32.

During the generative search for the optimal model architecture, each generation was cross-validated on the training data using 5-fold cross-validation. This was done to get a better understanding of how the performance of hyperparameter tuning among these models. The results of this process can be seen in Figure 4.1

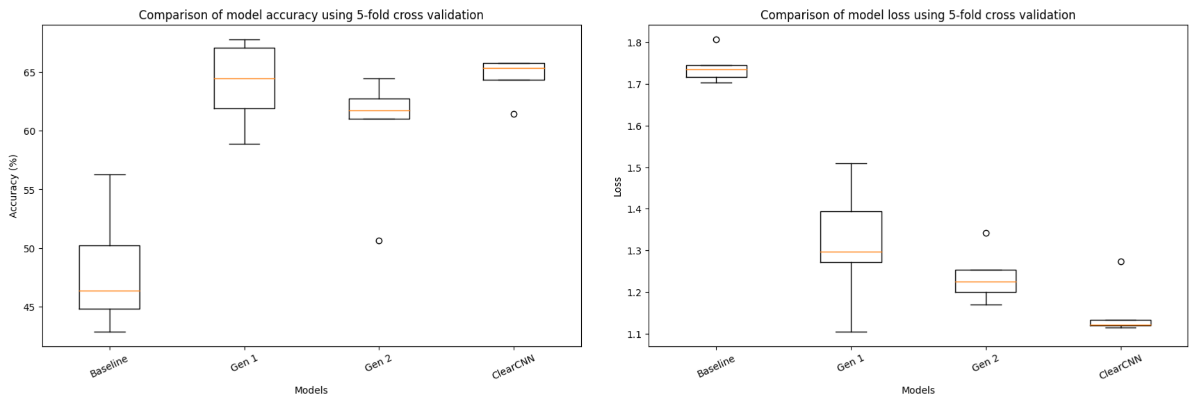

Figure 4.1: Comparison of different preprocessing methods under 5-fold cross-validation. The graph on the left shows validation accuracy and the graph on the right shows final validation loss.

In Figure 4.1 a strong increase in accuracy can be seen after generation 1, with the mean accuracy going from 48.09% ± 4.754 to 64.041% ± 3.321. Between generations 1 and 2 a small decrease in accuracy can be seen with the mean accuracy dropping to 60.14% ± 4.886. This decrease in accuracy, however, is paired with a decrease in validation loss, with validation loss dropping from 1.315% ± 0.134 to 1.238 ± 0.059. Finally, the resulting CNN, dubbed ClearCNN was able to achieve a mean accuracy of 64.558 ± 1.632 with a validation loss of 1.153 ± 0.061

4.3. Performance of Convolutional Neural Network

Using the generative approach reported in 3.5 the model pictured in Figure 4.2 was found to be the best fit for the air-written character recognition problem.

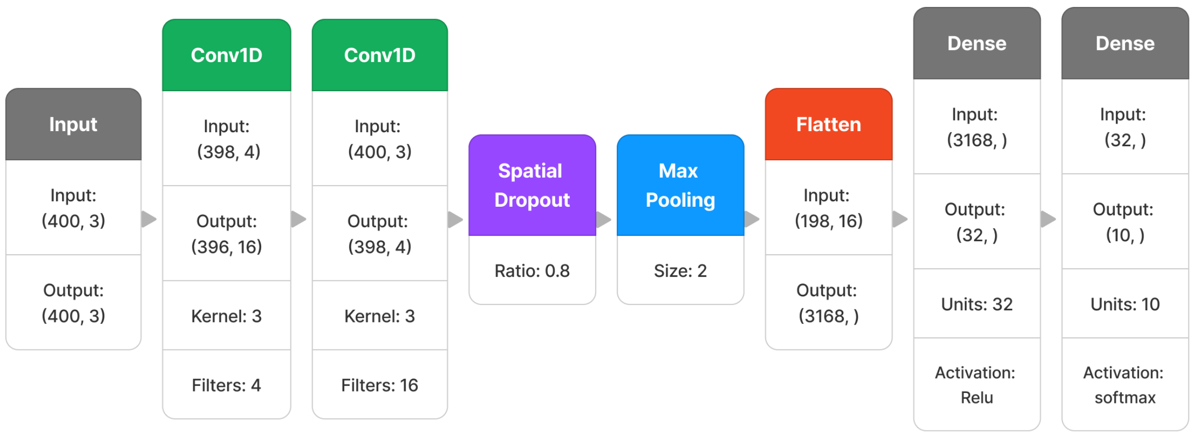

Figure 4.2: Resulting model from hyperparameter optimization. The model consist of 2 Convolutional layers, a Spatial Dropout layer, a Max Pooling layer, a Flatten layer and two Densely Connected layers.

The model has an input layer followed by two Convolutional layers. The first Convolutional layer operates with a 3×1 kernel and trains 4 filters per channel. The second Convolutional layer also operates with a 3×1 kernel, but trains 16 filters per channel. Next, a Spatial Dropout with a dropout rate of 0.8 and a max pooling layer with a pooling size of 2 is placed to combat the effects of overfitting. The pooled feature maps are then flattened and supplied to a densely connected layer with 32 units. This densely connected layer is responsible for acting as a buffer between the Flatten layer and the predictive layer after it. It allows the neural network to extract meaningful information from the feature maps. The final dense layer is a predictive layer, operating with a softmax activation function. The output of the machine learning model is a prediction vector containing the predicted probability for the data sample being in each of the 10 output classes.

When trained on the complete training set and evaluated on the test set ClearCNN was able to achieve a *within*-participant accuracy of 50.80% with a 1.56 validation loss and a *between*-participant accuracy of 67.82% with a 1.06 validation loss. The confusion matrices generated by the evaluation runs can be seen in Figure 4.3

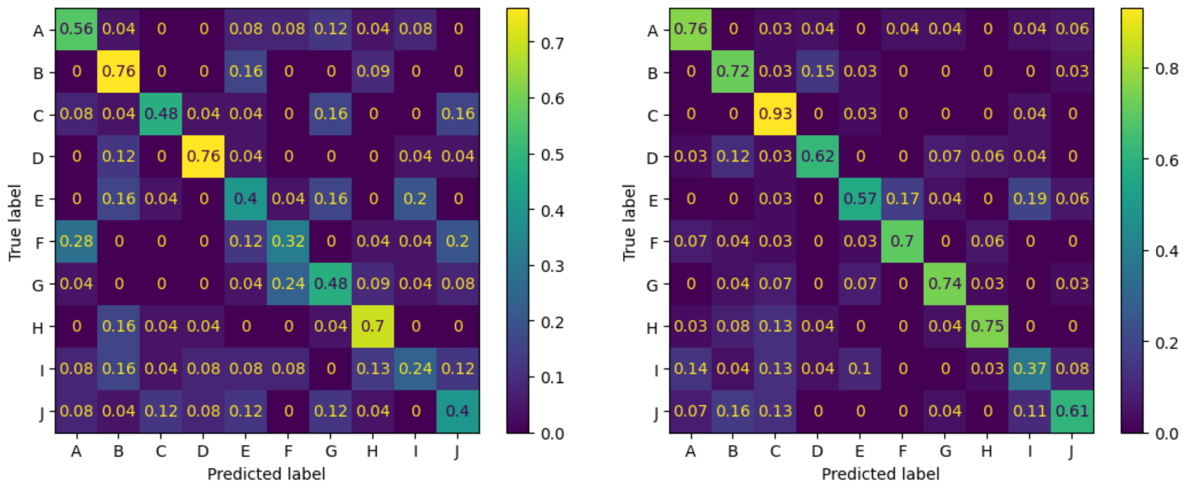

Figure 4.3: Confusion matrices generated by evaluation runs of the model on the test set. Left the *within*-participant evaluation can be seen, right the *between*-participant evaluation.

4.4. Effects of data preprocessing

The effects of the data preprocessing pipeline on the optimal model performance can be seen quite clearly. The graph below shows the evaluation scores of the ClearCNN when trained on raw data and when applying different preprocessing methods.

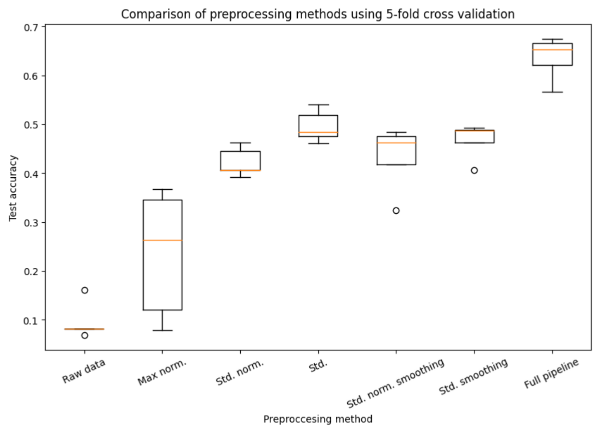

Figure 4.4: The effect of different data preprocessing measures. The plot shows 5-fold cross-validation accuracy per preprocessing method.

4.5. Model compression

When introduced to TensorflowLite without quantization, the resulting model had a size of 412.716 KB. Compression with dynamic range quantization resulted in a compressed model of 108.664 KB, which means a total compression ratio of 3.8. This made the model small enough to be deployed on resource-constrained hardware.

5 Discussion & Conclusion

In the following chapter, a conclusion for each part of the system will be presented and improvements possible to each part will be discussed. In Section 5.1 the resulting dataset will be discussed. Section 5.2 will evaluate the resulting data preprocessing pipeline. Lastly, the ClearCNN model architecture will be considered in Section 5.3.

5.1. Air written character dataset

The resulting air-written dataset contains 1500 data points obtained from 30 participants. This dataset is the first dataset created for an air-written character recognition system and could thus be used in future research. During data collection, the dataset was deliberately constrained to minimize variability. This was done by constraining data collection to a single location. Constraining the location of the dataset, however, did not prove to aid in keeping a low variability in the dataset. The individual writing style of the participants alone was enough to introduce a strong variance in the dataset.

Further improvements to the dataset could be made by incorporating different lighting conditions. The addition of different lighting conditions would however introduce problems to the dataset related to the variability which would be difficult to overcome. During this research experiments were performed under different lighting conditions, but these proved to introduce a considerable amount of variance. The effect of different lighting conditions on the dataset can be seen in Figure 5.1

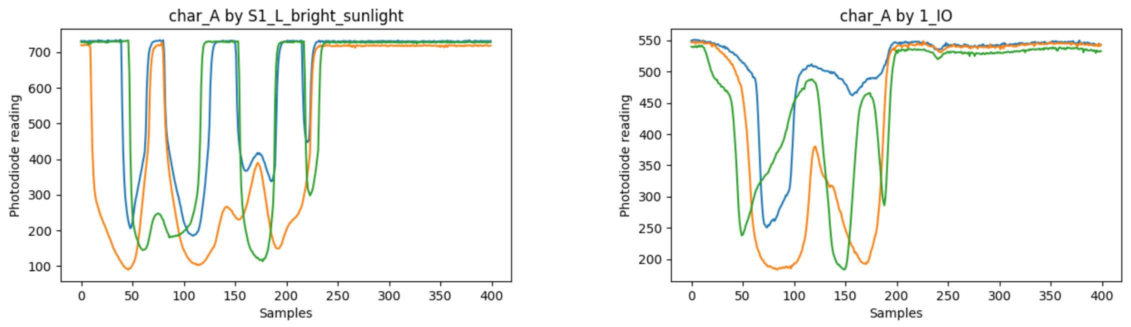

Figure 5.1: The effect of different lighting conditions on photodiode signals. On the left a signal captured in bright sunlight can be seen, while on the right a signal captured in the conditions described in Section 3.3. Both character signals were created by the same participant with the same writing style.

An additional improvement which could be made to the dataset is the addition of a description of the writing style used to create an air-written character. This could be used by a machine learning system to make a distinction between different character writing styles, which could in turn increase character recognition accuracy.

5.2. Preprocessing pipeline

The preprocessing pipeline used in the proposed system was able to increase the *within*-participant accuracy greatly. It was able to increase the *within*-participant accuracy by 36.45% between simple normalization and the full data preprocessing pipeline. The preprocessing pipeline described in this paper contains a good balance between simplicity and efficiency. Further improvements could be made by investigating more complex preprocessing techniques such as Fast Fourier Transform, [] and [] to extract even more features from the raw data.

5.3. The ClearCNN model

The ClearCNN model was able to reach a between-participant accuracy of 50.80% and a within-participant accuracy of 67.82%. This is, however, far from an accuracy that would be acceptable for a robust air-written character recognition system. Furthermore, this accuracy was achieved under the knowledge that the model tended to overfit the training data. Several measures such as dropout and batch normalization were used to decrease the tendency of the model to overfit. Sadly no method could completely eliminate this tendency.

Further improvements to the ClearCNN model could be made by re-evaluating the created dataset as well as performing a hyperparameter search using additional training data. Further research into a different model architecture using different hyperparameters is necessary to design a system that can operate in multiple lighting conditions. The ClearCNN model is a solid step in the direction towards a low-cost air-written character recognition, but there are some critical steps needed in order to create a fully functional air-written character recognition system.

6 Responsible Research

6.1. Ethical implications

The ethical implications within this research stem mainly from the data collection portion of the research. Data was collected from Human Research Subjects and thus falls under the TU Delft Regulations on Human Trials [19]. To mitigate risks to human participants a risk assessment was made and approval from the TU Delft Human Research Ethics Committee (HREC) was requested. Participants in the study were asked to sign a consent form to ensure their consensual participation.

6.2. Reproducible Research

The reproducibility of scientific work is of utmost importance, as well as the accurate reporting of results. During this study great care was taken to ensure the reproducibility of the results and methods. This was done by reporting every step taken in the research as accurately yet concisely as possible. The process of data collection especially, which bears a great impact on the created dataset is documented as accurately as possible. Furthermore, the dataset created during this project and the code used during this project will be uploaded to the public CSE3000 repository. With this, anyone should be able to recreate and validate the results presented in this paper.

6.3. Use of Large Language Models)

The rise of Large Language Models (LLMs) has posed new ethical implications within academic research and the scientific endeavor. LLMs such as ChatGPT [20] can be used as a tool supporting the research and writing process, but great care should be taken when using these tools.

Within this paper, ChatGPT was used as a tool to simplify and reformulate information supplied by reputable sources. An example of the use of ChatGPT in such a case is the input of an article on a complex subject such as kernel convolutions, with the prompt to simplify and expand on the given concepts. The information given by ChatGPT was used to better understand the original source. Other uses of LLMs included the use of Github Copilot [21] to support in the coding process.

References

[1] D. Sturman and D. Zeltzer, "A survey of glove-based input," *IEEE Computer Graphics and Applications*, vol. 14, no. 1, pp. 30–39, 1994. DOI: 10.1109/38.250916.

[2] M. Z. Iqbal and A. G. Campbell, "From luxury to necessity: Progress of touchless interaction technology," *Technology in Society*, vol. 67, p. 101 796, 2021, ISSN: 0160-791X. DOI: https://doi.org/10.1016/j.techsoc.2021.101796. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0160791X21002712.

[3] H. Liu, *Lightdigit: Embedded deep learning empowered fingertip air-writing with ambient light*, Aug. 2021. [Online]. Available: http://resolver.tudelft.nl/uuid:afc5471f-c2a3-4ecf-b15e-6edf43a9d22b.

[4] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "Emnist: Extending mnist to handwritten letters," in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 2921–2926. DOI: 10.1109/IJCNN.2017.7966217.

[5] F. Akadiri, *Constructing a dataset for gesture recognition using ambient light*, Jun. 2022. [Online]. Available: http://resolver.tudelft.nl/uuid:fbe34222-63d8-4d1d-a750-0ce6e0a9c132.

[6] , *Designing an adaptable and low-cost system for gesture recognition using visible light*, Jun. 2022. [Online]. Available: http://resolver.tudelft.nl/uuid:d41f29df-8958-433a-8e83-defc292bb27f.

[7] D. Barantiev, *Designing a software receiver for gesture recognition with ambient light*, Jun. 2022. [Online]. Available: http://resolver.tudelft.nl/uuid:3c89a9f1-13c7-4ba8-bbfd-7c5ebb01c042.

[8] W. Narchi, *Recognising gestures using ambient light and convolutional neural networks*, Jun. 2022. [Online]. Available: http://resolver.tudelft.nl/uuid:c4a0207e-8ebe-4f5a-ab4c-5675b6d92551.

[9] M. Lipski, *Hand gesture recognition on arduino using recurrent neural networks and ambient light*, Jun. 2022. [Online]. Available: http://resolver.tudelft.nl/uuid:366e8529-e282-42e5-9173-18443fb7504e.

[10] S. Kiranyaz, T. Ince, R. Hamila, and M. Gabbouj, "Convolutional neural networks for patient-specific ECG classification," en, *Conf. Proc. IEEE Eng. Med. Biol. Soc.*, vol. 2015, pp. 2608–2611, 2015.

[11] Z. Xiong, M. K. Stiles, and J. Zhao, "Robust ECG signal classification for detection of atrial fibrillation using a novel neural network," in *2017 Computing in Cardiology (CinC)*, ieeexplore.ieee.org, Sep. 2017, pp. 1–4.

[12] S. Kiranyaz, A. Gastli, L. Ben-Brahim, N. Al-Emadi, and M. Gabbouj, "Real-Time fault detection and identification for MMC using 1-D convolutional neural networks," *IEEE Trans. Ind. Electron.*, vol. 66, no. 11, pp. 8760–8771, Nov. 2019.

[13] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, Jun. 2014.

[14] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler, *Efficient object localization using convolutional networks*, 2015. arXiv: 1411.4280 [cs.CV].

[15] J. Rala Cordeiro, A. Raimundo, O. Postolache, and P. Sebastião, "Neural architecture search for 1D CNNs-Different approaches tests and measurements," en, *Sensors*, vol. 21, no. 23, Nov. 2021.

[16] T. O'Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi, *et al.*, *Kerastuner*, https://github.com/keras-team/keras-tuner, 2019.

[17] R. David, J. Duke, A. Jain, *et al.*, *Tensorflow lite micro: Embedded machine learning on tinyml systems*, 2021. arXiv: 2010.08678 [cs.LG].

[18] M. Papadatou-Pastou, E. Ntolka, J. Schmitz, *et al.*, “Human handedness: A meta-analysis.,” *Psychological Bulletin*, vol. 146, no. 6, pp. 481–524, Jun. 2020. DOI: 10.1037/bul0000229.

[19] Jun. 2016. [Online]. Available: https://filelist.tudelft.nl/TUDelft/Over_TU_Delft/Strategie/Integriteitsbeleid/Research%5C%20ethics/HREC-Articles_of_Association.pdf.

[20] [Online]. Available: https://openai.com/blog/chatgpt.

[21] [Online]. Available: https://docs.github.com/en/copilot/overview-of-github-copilot/about-github-copilot-for-individuals.