Hand Gesture Recognition on Arduino Using Recurrent Neural Networks and Ambient Light

Matthew Lipski
Supervisor(s): Mingkun Yang, Ran Zhu
EEMCS, Delft University of Technology, The Netherlands
22-6-2022

A Dissertation Submitted to EEMCS faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering

Abstract

Touching physical buttons to interact with public electronic devices has raised some concerns regrading disease transmission following the COVID-19 pandemic. The use of hand gestures as a touchless replacement sounds appealing, but comes with the challenge of recognizing which gesture is being performed by the user, with only the processing power of a microcontroller. This paper explores the use of recurrent neural networks (RNNs) and their derivatives to recognize hand gestures on an Arduino Nano 33 BLE. The neural networks receive input from 3 OPT101 photodiodes, which emit a voltage that increases with the intensity of light that hits them, meaning they can effectively track hand shadows cast by the user's hand under ambient light. After testing various RNN-based neural network architectures, CNN-LSTMs produced the highest validation accuracy. However, due to issues with the testing setup, the highest validation accuracy measured for a CNN-LSTM was only 43%, indicating that further work is required.

1 Preface

I would like to thank both my responsible professor, Qing Wang, as well my supervisors, Mingkun Yang and Ran Zhu, for their time and dedication in the CSE3000 Research Project, as well as for helping to guide my research and testing. Prior to writing this paper, I had little experience with machine learning and their expertise was invaluable in the completing my research. I would also like to thank my project group members, Stijn van de Water, Dimitar Barantiev, Femi Akadiri, and William Narchi, who were excellent to work with for this project. Thanks to their professionalism and engagement, working on the CSE3000 Research Project together was gratifying and without their contributions, this research would not have been possible.

2 Introduction

2.1 Research Overview

Traditionally, physical buttons have been by far the most common way for users to interact with electronic devices in public settings, whether these are coffee machines, elevator panels, or train ticket machines. However, the concern of disease transmission has become increasingly prevalent in recent years due to the COVID-19 pandemic, making it enticing to develop an alternative solution which does not require touch. One such solution is the use of hand gestures to interact with public devices instead. By performing hand motions such as swiping and tapping, users can effectively and intuitively interact with electronic devices with minimum risk of disease transmission.

However, there are a few key challenges to this approach which stand in the way of it replacing physical buttons in real-world applications.

1. Additional hardware is needed to detect the positions of a user's hand while performing a gesture. The data output by this hardware is likely to be low in resolution since system costs should be minimized.
2. Additional software must be implemented to recognize gestures based on hand position over time. While buttons are just simple digital inputs, one gesture can also be performed differently between different users, yet the system must be able to accurately classify it regardless.
3. Gesture recognition must be done in real-time. Since the process of classifying gestures can be quite complex, the latency introduced by this may be significant. However, the user should not perceive any lag while using the system for a positive experience.

2.2 Research Question

The goal of this paper can be summarized with the following research question:

'Which recurrent neural network architecture is most appropriate for recognizing hand gestures on an Arduino Nano 33 BLE, using 3D-formatted data from OPT101 photodiodes?'

This can then be segmented into the following sub-questions:

1. Which recurrent neural network architectures produce the highest accuracy for hand gesture recognition?
2. What is the minimum acceptable accuracy for recognizing hand gestures on an Arduino Nano 33 BLE?
3. What is the maximum acceptable inference latency for recognizing hand gestures on an Arduino Nano 33 BLE?
4. How can 3D-formatting data be exploited for better gesture recognition performance?

2.3 Contributions

This research overcomes the challenges outlined in section 2.1 by using data from OPT101 photodiodes, which is fed into a CNN-LSTM neural network to recognize gestures on an Arduino Nano 33 BLE microcontroller. It is also part of a larger project which integrates this neural network into a full gesture recognition system, which is elaborated on in section 3.2.

Similar research which involves using photodiodes and machine learning to recognize hand gestures has already been conducted, but this paper improves on existing solutions in a number of ways:

1. The system uses fewer photodiodes, resulting in fewer neural network input features, than existing solutions.
2. The data from photodiodes is 3D-formatted, which better preserves temporal information and improves recognition accuracy. An explanation for what 3D-formatting involves can be found in section 3.4.
3. The CNN-LSTM architecture used yields a higher validation accuracy than architectures used in existing solutions.

A discussion regarding existing research in the field of gesture recognition on embedded devices is found in section 8, which provides more context to these improvements.

3 Background

3.1 Machine Learning & Artificial Neural Networks

Machine learning is a sub-field of artificial intelligence which ”provides learning capability to computers without being explicitly programmed” [1]. More specifically, machine learning allows computers to ”learn” patterns and trends from existing data in order to make accurate predictions on new data, without any human input.

Artificial neural networks (ANNs) are a type of machine learning model which draw inspiration from the human brain [13]. These types of models feature ”neurons” organized into densely connected layers, as shown in figure 1. The first layer, which takes in the input data, is known as the input layer while the final layer, which yields the processed output data, is called the output layer. In between these is at least one hidden layer and in general, having more hidden layers and more neurons per hidden layer allows a neural network to approximate increasingly complex functions.

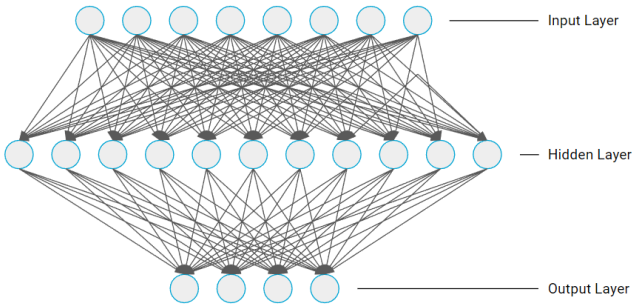

Figure 1: Visualization of a generic ANN with 8 input neurons and 4 output neurons, as well as a single hidden layer with 11 neurons.

Each neuron inputs and outputs a single value, and each connection between two neurons has an associated weight and bias, which determine how much the output from the source neuron affects the output of the destination neuron. These values change as the model ”learns” from existing data, causing the network’s performance to gradually improve. In addition to this, each neuron passes the combined input of all neurons in the previous layer through a non-linear activation function, which is what allows neural networks to effectively model any function possible [13].

Although ANNs are no longer considered state-of-the-art, understanding the purposes of neurons, neuron connections, layers, and activation functions remains useful, as these still form the building blocks of any neural network architecture.

3.2 Neural Networks on Embedded Hardware

Machine learning, and specifically neural networks, have traditionally been restricted to the realms of high-performance and in turn, high power devices [2]. Unfortunately, this means that it has been previously impractical to use these technologies with embedded hardware such as microcontrollers, as they lack the memory & performance to run inference on neural networks locally, while using a remote processor for inference is not always feasible.

Advances into machine learning model optimization have changed this, allowing deep neural networks with multiple hidden layers to be run on devices even powered by coin batteries. The most prominent development in this field has been TensorFlow Lite, which is a multi-language framework used to optimize existing neural networks such that they can be ran on mobile devices [4]. The key optimization introduced by TensorFlow Lite is quantization, which allows converts all point weights and activation functions in a neural network from 32-bit floating points to 8-bit integers, resulting in a tremendous improvement to the size of the neural network in memory as well as its runtime.

However, mobile devices still have considerably more processing power and hardware capabilities than embedded devices, which is an issue that led to the development of TensorFlow Lite for Microcontrollers in 2018 [4]. TensorFlow Lite for Microcontrollers is a fork of TensorFlow Lite which further optimizes models such that they can be run on devices with extremely limited resources by using exclusively C/C++ and cutting down massively on dependencies.

3.3 Hand Gesture Data

In this research, neural networks are used to detect gestures, but to do this, data from some sensor(s) must be fed into the network.

There are a variety of sensors which could be used to record hand gestures and provide this data. One of these is an accelerometer attached to the user’s wrist (typically from a smartwatch) to track the direction and acceleration of the their hand movements [10]. Another option is using a depth/range camera to record the user’s hand, which provides a huge amount of information but in turn requires a computationally intensive video processing pipeline to make sense of the data [9]. Photodiodes can also be used, which are sensors that output a signal which increases with the amount of light that hits them. This means they can track the shadows cast by the user’s hand under ambient light, therefore making it possible to recognize which gesture is being performed [5]. This project uses photodiodes for their much lower monetary and computational cost compared to cameras as well as the fact that they don’t require the user to wear a device on their wrists, unlike accelerometers.

3.4 3D-Formatted Data

The term ”3D-formatted data” is specific to this research, and must be explained to understand the choice of neural network architectures tested. This is best done by comparing it to ”2D-formatted data”.

2D-formatted data can be represented as an image, with some horizontal resolution x and vertical resolution y. In this research, each photodiode outputs values at a predetermined sampling rate over the course of a gesture. This data can be formatted as a 2D image in which x is the number of photodiodes used and y is the number of total samples received from any of the photodiodes. This results in the value of each ”pixel” in the image representing a reading from a single photodiode at a single point in time.

3D-formatted data can meanwhile be thought of as a video, which splits this 2D-image into a sequence of n frames, as

shown in figure 2. 3D-formatting is generally appropriate when the data is sensitive to time, i.e. when data points should be considered in a specific sequence.

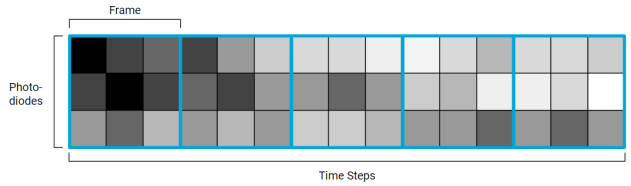

Figure 2: Visualization of 2D photodiode data after being 3D-formatted into 5 frames.

4 System Overview

4.1 Full Gesture Recognition System

This research focuses on finding an appropriate neural network architecture to perform gesture recognition on a microcontroller, but it is only part of a larger project to create an entire gesture recognition system/pipeline. The creation of this pipeline is composed of the following tasks:

1. Optimizing the number and placement of OPT101 photo diodes.
2. Pre-processing data from photo diodes.
3. Creating an appropriate dataset for training a neural network to recognize gestures.
4. Finding an appropriate neural network architecture on the created dataset and ensuring gestures can be recognized in real-time on an Arduino Nano 33 BLE.

The research presented in this paper aims to complete task 4. Although tasks 1–3 were completed by other project group members and are beyond the scope of this research, they are worth mentioning to provide some context regarding the rest of the gesture recognition system. Due to the findings from these tasks, the final system uses 3 photodiodes and can recognize 10 different gestures, while each gesture is composed of 100 time steps from each photodiode. This is relevant for task 4, as it means that whatever neural network is implemented must use a 2D array of size 3 by 100 (split into n frames after 3D-formatting) as an input feature and be able to distinguish between 10 output classes, as illustrated in figure 4. The 10 gestures that the system can recognize are illustrated in figure 3.

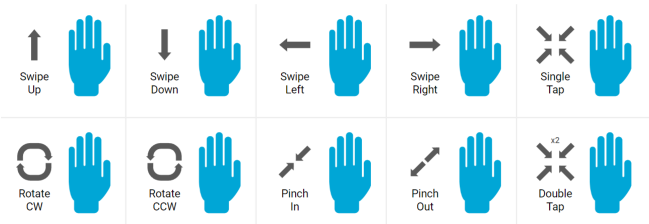

Figure 3: Illustrations of the 10 different gestures that the system can recognize.

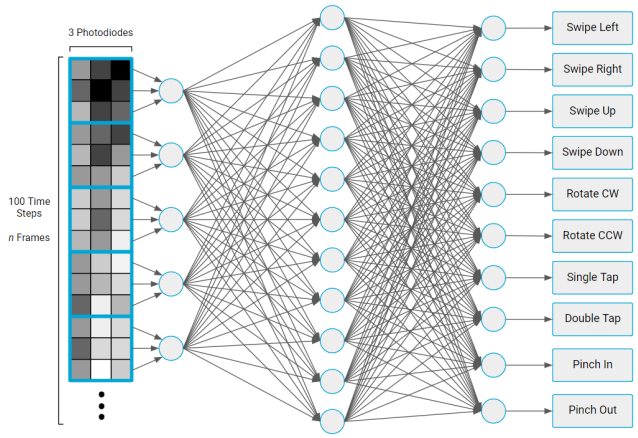

Figure 4: Visualization of the input features & output classes using a generic artificial neural network as an example.

4.2 System Caveats

The overarching goal of the project that this research contributes to is the creation of a full gesture recognition pipeline, which presents some issues when considering the fact that each part of this pipeline was developed in parallel due to the limited time allotted for the project. In reality, it would make much more sense to complete each step of the pipeline sequentially, as the performance of later parts of the pipeline relies on the performance of previous parts. To put this in the context of the gesture recognition system, it is impossible to train a neural network to recognize gestures without first having a dataset to train it on. However, the creation of that dataset relies on photodiode count and placement, as well as sampling rate and pre-processing, being finalized. If they change after the dataset is completed, it will not be representative of real-world data, leading to poor gesture classification performance from the neural network trained on it.

4.3 Dataset

Having a varied, expansive, and representative dataset is crucial for training a machine learning with high real-world accuracy. Fortunately, a dataset for recognizing gestures using photodiode data was done by another member of the project group, as mentioned in section 4.1.

Unfortunately, because the allotted time for the project meant that all group members had to work in parallel, as stated in section 4.2, the neural networks evaluated in this paper had to be trained on a dataset that is not final. Specifically, the dataset used in this research is not passed through the data pre-processing stage of the system. This means that the neural networks presented in this paper were trained on raw data from the photodiodes, whereas real-world data on the final system would be passed through this pre-processing stage. This leads to model accuracy being lower than it could be, as there is noise and other artifacts in the raw photodiode data.

The dataset used for this research contains 5 repetitions of each of the 10 gestures per hand across 47 participants/candidates. This would result in 4700 data instances, but some were removed for various reasons. Therefore, the

dataset instead has 4672 total data instances. Each instance is a 5 second window during which a gesture is performed with a sampling rate of 20Hz for candidates 1–27, and 100Hz for candidates 28–48, resulting 100 samples per photodiode and 500 samples per photodiode respectively. However, the instances recorded at 100Hz are down-sampled back to 20Hz so that the data format remains consistent across the whole dataset.

The only other preprocessing that was done on the dataset besides 3D-formatting the data and down-sampling 100Hz instances to 20Hz, was normalizing the values of each data instance to a range of [0.0, 1.0].

Although the dataset contains instances from a variety of environments and lighting setups, these are mostly indoor locations as this is the planned use case for the system. The dataset was also mostly recorded on the TU Delft campus, meaning the demographic of participants is somewhat skewed. Most notably, the dataset contains substantially more instances of males compared to females, and more right-handed participants than left-handed. However, this is not expected to have a large impact on the performance on any neural networks trained on this dataset.

5 Model Design

5.1 Neural Network Architectures Tested

Recognizing hand gestures based on photodiode data can be thought of as a time-series classification task, which is a type of problem that recurrent neural networks (RNNs) are especially well suited for [7]. RNNs differ from conventional neural networks as they do not process the input in its entirety, but instead process each time step, or sample, from the input sequentially. Each time step is used to update a "hidden state", which is fed back into the RNN along with the next time step. This makes RNNs suitable for operating on sequences, as the order of the data is considered as well as the content of the data, unlike typical ANNs. Thanks to this desirable property, RNNs were the first type of neural network tested.

Although RNNs are highly suitable for time-series classification, they suffer from the "vanishing gradient problem", which has since been overcome by long short-term memory cells (LSTMs) and gated recurrent units (GRUs) [6]. To briefly explain this problem, time steps further in the past tend to have an exponentially lower weight in determining the hidden state, meaning that only the past few time steps are actually represented in the hidden state. This means that in practice, RNNs tend to ignore if earlier time steps are out of order, which limits performance for long data sequences. LSTMs and GRUs solve this issue using so-called "gates", which determine which contents of the previous hidden state should be kept and which contents of the current hidden state should be propagated to the next time step. This largely mitigates the vanishing gradient problem as only relevant information is kept, therefore greatly reducing the rate at which hidden state weights decay. Given that LSTMs and GRUs are should yield better classification performance than RNNs due to this advantage, these were also investigated.

One issue of using LSTMs and GRUs, however, is that they only accept a single data sequence with multiple features as input, in which each time step contains a single value from each feature, resulting in a 1D array. This is relevant as with 3D-formatted data, each time step is a 2D frame - not a 1D array. Therefore, each frame has to be flattened first, which causes a loss of spacial information. This issue can be solved by using a 2D convolutional neural network (CNN) to first extract features from the input data, which are then fed into the LSTM [8]. 2D CNNs logically accept 2D images as input, which means that frames do not have to be flattened. The convolutional and pooling layers of the CNN can reduce the resolution of this image to just a single value for each neuron in the final layer. Therefore, the final CNN layer effectively outputs a 1D array with a size equal to the number of neurons in it, which can then be used as input to the LSTM with no loss in spacial information. Due to this, the CNN-LSTM architecture was also investigated in this study.

5.2 Parameter Tuning

When training a machine learning model, the model parameters can have a substantial impact on final performance. For the neural networks tested in this paper, some of these parameters affect almost all model types, while some are specific to individual architectures. These are outlined below:

All Architectures

- 3D-formatting frame length
- Number of layers
- Number of neurons per layer

CNN+LSTM

- Number of convolutional layers
- Number of neurons per convolutional layer
- Filter kernel sizes

In reality, there are more parameters that could be tuned, but the ones listed above are likely to have the largest impact on final model. Default values are used for parameters that are not mentioned in this section. This paper does not go into detail regarding the testing of different parameters to determine which combinations yield optimal performance, but the best performing parameter values found are stated in section 7.3.

6 Model Implementation

6.1 Training Code

The implementation of each neural network architecture was done using the TensorFlow Python library and high-level Keras API (version 2.9), which were then converted into TensorFlow Lite models for validation with 8-bit integer neuron weight quantization. The model training was done with a batch size of 32, Adam optimizer, and default learning rate. While the dataset is split into training and validation sets when evaluating architecture performance, the final models to be deployed on Arduino are trained on all available data.

6.2 Inference Code

To run trained Keras/TensorFlow models on the Arduino Nano 33 BLE, they were first converted to TensorFlow Lite model files (.tflite extension) with 8-bit integer neuron weight and activation function quantization, which are explained in section 3.2. The TensorFlow Lite models were then converted into C++ files so that they can be loaded and ran using TensorFlow Lite for Microcontrollers. This was done using the method detailed in chapter 4.5 of "TinyML Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers" [14].

7 Results

7.1 Final Model Parameters

RNN

- 3D-formatting frame length: 10
- Number of layers: 2
- Number of neurons per layer: 256/128

LSTM

- 3D-formatting frame length: 10
- Number of layers: 2
- Number of neurons per layer: 128/64

GRU

- 3D-formatting frame length: 10
- Number of layers: 4
- Number of neurons per layer: 64/64/64/64

CNN+LSTM

- 3D-formatting frame length: 5
- Number of layers: 1
- Number of neurons per layer: 64
- Number of convolutional layers: 3
- Number of neurons per convolutional layer: 128/128/64
- Filter kernel sizes: (2, 2), (2, 2), (3, 1)

7.2 Inference Latency Testing

Methodology

Unfortunately, it is not currently possible to test the inference latency performance of most neural networks architectures tested directly on the Arduino Nano 33 BLE. As mentioned in section 6, the TensorFlow library is used to train the neural networks, while TensorFlow Lite for Microcontrollers is used run inference on them. However, the Arduino implementation of TensorFlow Lite for Microcontrollers does not currently have support for recurrent neural networks, which are also used in all other architectures tested. While it is possible to use RNNs with the most recent version of TensorFlow Lite for Microcontrollers (version 2.9), the library's Arduino implementation is a few versions behind (currently version 2.4), and is outdated enough that it is missing certain tensor operations necessary for running RNNs and their derivatives.

Configuration	File Size (MB)	Latency (ms)
(1/128)	42	103.9
(1/256)	81	179.6
(1/512)	160	333.0
(2/64)	27	74.7
(2/128)	59	134.5
(2/256)	147	293.7
(4/32)	17	55.4
(4/64)	59	92.3
(4/128)	93	195.0

Table 1: Table comparing the file sizes and inference latencies of various TensorFlow Lite artificial neural networks.

Therefore, a different method was used to estimate the inference latency of RNNs on Arduino. Since regular artificial neural networks are compatible with TensorFlow Lite for Microcontrollers version 2.4, several configurations of these with varying neuron and densely connected layer counts were trained and deployed on the Arduino Nano 33 BLE. These configurations are listed in table 1, formatted as (number of densely connected layers layers/number of neurons per layer). Each configuration also includes a final densely connected layer of 10 neurons for output. Since ANNs require 1D input data, a dummy data instance was taken from the dataset and flattened into a single array of 300 values (3 photodiodes $\times$ 100 samples). This dummy data instance was then used as input to the neural networks so that their latencies could be measured. The inference times of these generic networks on the Arduino Nano 33 BLEs were compared to their TensorFlow Lite model file (.tflite extension) sizes, to see if file size could be used to predict inference latency.

Results

Figure 5 illustrates that a neural network's inference latency can indeed be estimated using the function $y = 1.89x + 23.6$, where x is the model file size in kilobytes and y is the model's expected inference latency in milliseconds.

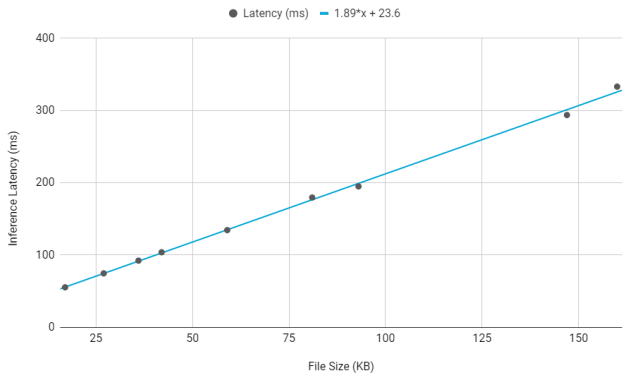

Figure 5: Graph showing the relationship between file size and inference latency for various TensorFlow Lite artificial neural networks.

In a paper by Duan *et al.*, a nearly identical gesture recognition model was implemented for which real-time inference latency was classified as being <62.5 milliseconds [5], but this is highly restrictive using the system presented in section 4.1. Instead, a more conservative real-time latency value was used of <627ms, which is the average human reaction time [11] and should still ensure a positive user experience.

The largest artificial neural network which was tested, with a 160KB file size, produced a latency of 333ms despite using 97% of the available RAM on the Arduino Nano 33 BLE (this memory usage includes the boilerplate code needed for loading and running the model), which is well within the limit of what is considered real-time for this paper.

Therefore, inference latency is not much of a concern, though it was important that all models tested in this study did not exceed a file size of ~160KB, so that they could be run on the Arduino in the future. The file sizes of each architecture tested using their final parameters, as well as their expected inference latencies, can be seen in figure 6.

Figure 6: Graph showing the TensorFlow Lite file sizes of all architectures tested using their final parameters, as well as their expected latencies on the Arduino Nano 33 BLE.

7.3 Accuracy Testing

Methodology

To ensure that the measured validation accuracy was representative across all neural network architectures tested, four distinct measures were put in place:

1. A dropout layer with $p = 0.5$ was added before the output layer for each architecture to reduce overfitting [12].
2. K-fold cross-validation [3] was used with $k = 5$, as validation accuracy can be skewed based on how the test and validation sets are split.
3. Each neural network was trained until the value of the validation function no longer improved for 100 consecutive epochs, mitigating the variation in training time caused by randomizing initial neuron weights as well as differences in architecture.
4. Data instances for each candidate in the dataset were shuffled randomly, as ordered data can cause slow training times and poor performance. However, the order of candidates was not shuffled to ensure that gestures in the training data were recorded by different candidates to gestures in the validation data, making validation more representative of a real-world scenario.

Results

As can be seen from figure 7, none of the neural network architectures tested achieved an acceptable validation accuracy. The best performing architecture was the CNN-LSTM, which had an average validation accuracy of 43% across the 5 folds. This shows that the loss in spacial data caused by flattening frames in the other architectures causes a measurable loss in accuracy, and that the CNN is effective at extracting 1D features from a 2D frame. The LSTM meanwhile, performed considerably worse than its counterparts with 28% accuracy, likely due to the fact that LSTM layers require more memory per neuron than RNN, GRU, and 2D convolutional layers, causing the model to be more restricted by the 160 KB file size limit.

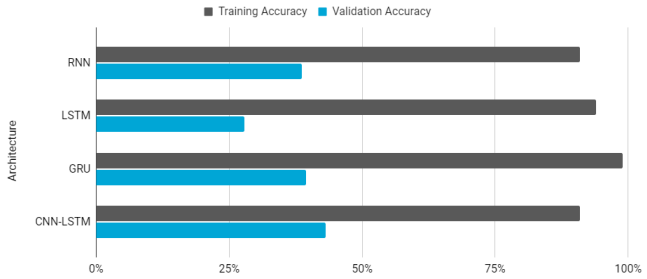

Figure 7: Graph showing training and validation accuracies of all architectures tested using their final parameters.

However, the clear issue with all of these results is that validation accuracy is not only low, but also significantly lower than training accuracy, which indicates overfitting is occurring. Figure 8 shows the validation accuracy of each architecture per fold, and may suggest a reason for why this is the case, as folds 1 and 2 produce a considerably higher accuracy than the others. When considering that the order of candidates remains the same when the dataset is loaded, this shows that gestures were performed by certain candidates are not representative of the dataset as a whole. Therefore, when the neural networks were trained using data from these candidates, they struggled to generalize to the rest of the dataset, resulting in poor validation accuracy. This is hypothesis is falls in line with results found from an informal test which involved evaluating the CNN-LSTM again, but shuffling the whole dataset beforehand instead of shuffling only the gestures for each candidate. By doing this, the training and validation sets contained data instances from all candidates in the dataset, meaning that the neural network was not presented with previously unseen candidates during validation. In this test, the CNN-LSTM achieved validation accuracy of 79% over the 5 folds, a near twofold improvement in performance, further reinforcing that gestures are performed or recorded inconsistently between candidates. Although there is some variation in how different people will perform the same gesture, it should not be this high, meaning that the inconsistency found in the dataset is likely due to environment and ambient lighting changes which are not accounted for by the rest of the system.

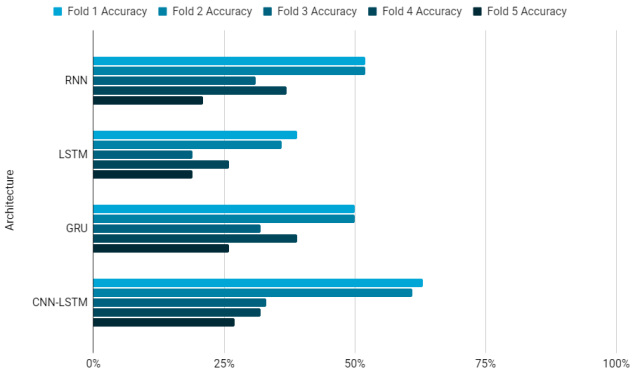

Figure 8: Graph showing the validation accuracies of all architectures tested using their final parameters for each individual validation fold.

While validation accuracy improved greatly in this test, overfitting was still present as the training accuracy was still higher, at 95%. The cause of this is more difficult to diagnose, but it seems that the single dropout layer used for each architecture was not enough to prevent significant overfitting, and other methods of regularization are needed.

One positive aspect of the results is that the optimal parameters for all architectures included a frame length of 5 or 10, meaning that 3D-formatting the input data does indeed have a positive impact on neural network performance, as otherwise the optimal frame length would be 1. This is likely because without 3D-formatting, an RNN based model would only have readings from each photodiode at a single as features for each time step, which equates to 3 features total per time step as the system uses 3 photodiodes. Given the complexity of recognizing gestures from photodiode data, 3 features are unlikely to provide enough information to the model at a given time step. By grouping time steps into frames, 3D-formatting effectively provides more features per time step, at the expense of a reduced total number of time steps. At a frame size of 5–10, this trade-off appears to be the most beneficial.

8 Related Work

8.1 Hand Gesture Recognition Using uWave

In 2009, Liu *et al.* [10] proposed uWave, an algorithm which could accurately recognize hand gestures using smartwatch accelerometer data. The main technology powering uWave is dynamic time warping (DTW), which calculates the distance/error between two sequences. This allows data output by the accelerometer to be mapped to a gesture by comparing it to internal models of the 8 gestures [10] that uWave can recognize. Whichever internal representation yields the lowest error using DTW is chosen as the output gesture.

Although uWave achieves a high accuracy and DTW is computationally cheap, there are some fundamental drawbacks to this algorithm. The main drawback is that the gesture recognition is user-dependent, meaning that the internal gesture models are created from data generated by a single person, and that accuracy would plummet if uWave when attempting to recognize gestures of other users. While this approach makes sense for software using a smartwatch, which are personal belongings, it requires the user to first record gestures before the system is able to recognize them, and is impractical for public devices, which are the focus of this paper.

8.2 Hand Gesture Recognition Using Range Cameras

In 2012, Lahamy and Lichti used a method of recognizing the number of extended fingers on a person's hand using data from range cameras which was fed through an image processing pipeline [9]. Range cameras generate "a full 3D point cloud with an array sensor at video rates" [9] of the environment , meaning they can capture spacial, depth, and temporal data, something that typically requires multiple sensors of different types.

The full pipeline implemented by Lahamy and Lichti is beyond the scope of this research, but some conclusions about the use of range cameras for gesture recognition can be drawn from their results. Firstly, one of the improvements suggested in their paper was to improve the speed at which the system can segment parts of the image which contains the user's hand from the background [9], suggesting that the pipeline has a slow runtime overall. This is coupled with the fact the the pipeline was run on, admittedly decade old, PC hardware. Despite the age of the hardware, the processor used still has significantly more computational performance than a what can be found on microcontrollers, such as the Arduino Nano 33 BLE. Secondly, the accuracy of the pipeline was only measured to be 38% [9], which is lower than what was achieved using a CNN-LSTM neural network. Overall, the high computational cost and low accuracy of this approach makes it unsuitable for gesture recognition on embedded devices.

8.3 Hand Gesture Recognition Using Photodiodes and Recurrent Neural Networks

In 2020, Duan *et al.* [5] built a system for gesture recognition on embedded systems using photodiodes and recurrent neural networks. This is almost identical to the system proposed in this paper, albeit with a few changes. The system in this paper uses 3 photodiodes and can recognize 10 gestures, while the system mentioned in the paper by Duan *et al.* uses an array of 8 photodiodes but can only recognize 7 gestures [5]. Additionally, this research also tests CNN-LSTMs for recognizing gestures, while the paper by Duan *et al.* only tests RNNs, LSTMs, and GRUs.

Unsurprisingly, the study conducted by Duan *et al.* has led much of the research for this paper. It is worth noting, however, that despite the lower number of input features due to the decreased photodiode count, and increased number of gestures that can be recognized, none of the neural networks tested in this research were unable to produce even close to the accuracy of $> 99\%$ achieved in the study [5].

9 Responsible Research

All code used for this research is open source can be found at https://github.com/matthewlipski/research_project. Due to the inherent randomness present when training neural networks, the results presented in this paper may not be exactly reproducible. However, using the same code and hardware, it should be possible for anyone to reach the same conclusions based on their own findings. The code being open source also allows others to find potential flaws in it, allowing it to be worked on and used for future research.

The work conducted for this study was also heavily focused on experimentation rather than reviewing literature. Naturally, research into existing work was still required to provide a starting point for testing and find which neural network architectures would be most suitable for recognizing gestures, as well as find potential improvements to existing work. However, this emphasis on testing and experimentation rather than reading literature means that the testing methodology falls under the under most scrutiny, and that ensuring the results of this research are reproducible is paramount. This is why much of the paper is devoted to explaining the system and testing setup. Although it may seem overly detailed at times, it is crucial to go over the methodology in depth in order for the results presented in this study to be reproducible.

10 Conclusion

Based on the results of this study CNN-LSTMs appear to be the most suitable neural network architectures for recognizing gestures on an Arduino Nano 33 BLE using 3D-formatted input data. However, the obvious caveat to this statement is that the CNN-LSTM tested was still unable to achieve a validation accuracy that would be sufficient for a positive user experience. With an accuracy of 43%, users would have to perform gestures 2–3 times on average before they are correctly recognized, leading to only frustration.

Overfitting is one key area in which this research could be improved, as this was an issue that was found across all neural network architectures tested. The addition of further dropout layers, and investigation of other regularization techniques such as L2 regularization may provide a solution to this.

However, this overfitting is also due to parts of the dataset not being representative of the rest, which limited the neural networks tested from being able to generalize to the dataset as a whole. Addressing this issue should not require building a new dataset from scratch, but rather finding which data instances cause issues with model training. These instances could simply be removed, or preferably, additional pre-processing could be implemented to make them better represent the dataset as a whole. As mentioned in section 4.3, a pre-processing pipeline was created by a fellow project group member, but the dataset used for training did not make use of it due to time constraints. Overall, better integration of all parts of the gesture recognition system is undoubtedly needed.

References

[1] Jafar Alzubi, Anand Nayyar, and Akshi Kumar. Machine learning from theory to algorithms: An overview. *Journal of Physics: Conference Series*, 1142:012012, nov 2018.

[2] Liliana Andrade, Adrien Prost-Boucle, and Frédéric Pétrot. Overview of the state of the art in embedded machine learning. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1033–1038, 2018.

[3] Daniel Berrar. *Cross-Validation*. 01 2018.

[4] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Tiezhen Wang, Pete Warden, and Rocky Rhodes. Tensorflow lite micro: Embedded machine learning for tinyml systems. In A. Smola, A. Dimakis, and I. Stoica, editors, *Proceedings of Machine Learning and Systems*, volume 3, pages 800–811, 2021.

[5] 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.

[6] Yuhuang Hu, Adrian E. G. Huber, Jithendar Anumula, and Shih-Chii Liu. Overcoming the vanishing gradient problem in plain recurrent networks. *CoRR*, abs/1801.06105, 2018.

[7] Michael Hüsken and Peter Stagge. Recurrent neural networks for time series classification. *Neurocomputing*, 50:223–235, 2003.

[8] Tae-Young Kim and Sung-Bae Cho. Predicting residential energy consumption using cnn-lstm neural networks. *Energy*, 182:72–81, 2019.

[9] Herve Lahamy and Derek Lichti. Real-time hand gesture recognition using range cameras. 05 2012.

[10] Jiayang Liu, Zhen Wang, Lin Zhong, Jehan Wickramasuriya, and Venu Vasudevan. uwave: Accelerometer-based personalized gesture recognition and its applications. In *2009 IEEE International Conference on Pervasive Computing and Communications*, pages 1–9, 2009.

[11] Charles Arthur Nagler and William Merle Nagler. Reaction time measurements. *Forensic Science*, 2:261–274, 1973.

[12] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[13] Sun-Chong Wang. *Artificial Neural Network*, pages 81–100. Springer US, Boston, MA, 2003.

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