Hand Gesture Recognition on Arduino Using Photodiodes and Neural Networks

Matthew Lipski
Supervisors: Mingkun Yang , Ran Zhu
EEMCS, Delft University of Technology, The Netherlands
m.s.lipski@student.tudelft.nl, m.yang-3@tudelft.nl, r.zhu-1@tudelft.nl

Abstract

1 Introduction

1.1 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.

This research overcomes these challenges by using data from OPT101 photodiodes, which is fed into a neural network to recognize gestures with high accuracy and low latency 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. A high level of accuracy is maintained with fewer photodiodes, and therefore fewer model 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 2.4.
3. The neural network used to classify hand gestures is shallower than competing solutions, leading to reduced system latency while maintaining a high level of accuracy.

A discussion regarding existing research in this field is found in section 7, which provides more context to these improvements.

1.2 Research Question

Given the challenges overcome and contributions to the field made, the goal of this paper can be summarized with the following research question:

"Which 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 neural network architectures produce the highest accuracy for hand gesture recognition?

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

2 Background

2.1 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, unseen data.

Artificial neural networks (ANNs), meanwhile, are a type of machine learning model which draws inspiration from the human brain [12]. These types of models feature "neurons" organized into densely connected layers. 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. In general, having more hidden layers and more neurons in hidden layers allows a neural network to model increasingly complex functions.

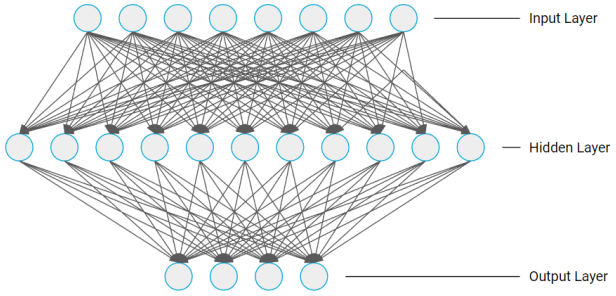

Figure 1: Visualization of a generic ANN with 8 input neurons and 4 output neurons, as well as a single hidden layer with 11 neurons.

Each 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 [12].

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.

2.2 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 unfeasible to use these technologies with embedded hardware such as microcontrollers, as they lack the performance to run inference on neural networks locally, and the dedicated power to be able to transmit sensor data to a remote processor.

However, recent advances into machine learning model compression and optimization have changed this, allowing deep neural networks to be run on devices even powered by coin batteries. The most prominent development in this field has been TensorFlow Lite for Microcontrollers, which is a Python framework specifically made for optimizing and running neural networks on microcontrollers [4]. The original Tensorflow has been an extremely popular machine learning framework for over a decade, but the extension to make it work effectively on microcontrollers has only been developed in recent years.

2.3 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 [9]. 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 [8]. 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 effectively track shadows cast by the user's hand when some ambient light is present, 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.

2.4 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 project, each photodiode outputs values at a predetermined sampling rate over the course of a gesture. This means that we can format said data as a 2D image in which x is the number of photo diodes we use and y is the number of total samples we receive from any of the photo diodes, while the value of each "pixel" in the image is a reading from a single photo diode 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 more appropriate when the data is sensitive to time, i.e. when data points should be considered in a specific sequence. Therefore, by 3D-formatting the photo diode data, lower error rates should be achievable as temporal information from the photo diodes isn't lost.

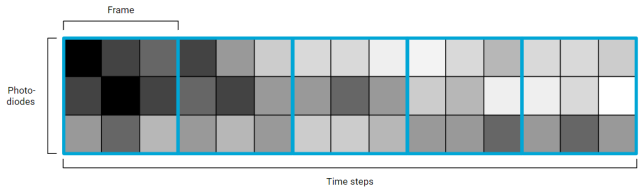

Figure 2: Visualization of photodiode data after 3D-formatting into 5 frames.

3 System Overview

3.1 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. This project is composed of the following tasks:

1. Optimizing the number and placement of OPT101 photo diodes.
2. Reading, processing, and sanitizing 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 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 8 different gestures, while each gesture is composed of 100 readings from each photodiode. This is also relevant for task 4, as it means that whatever architecture is used 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 8 output classes, as illustrated in figure 3.

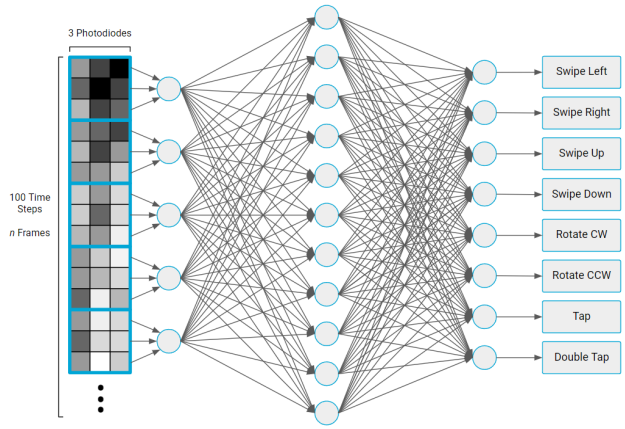

Figure 3: Visualization of the inputs & outputs for a generic neural network in the gesture recognition system.

3.2 Dataset

Having a varied, expansive, and representative dataset is crucial for training a machine learning with high real-world accuracy. Fortunately, the task of creating such a dataset for recognizing gestures using photodiode data was done by another member of the project group. This dataset is contains 5 repetitions of each of the 8 gestures per hand across 50 participants, leading to 4000 total data instances. 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 final model performance.

4 Model Design

4.1 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 gives RNNs the ability to exploit the time-sensitive nature of time-series data, as each time step also has temporal information associated with it. 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, the weight of the hidden state of a time step in an RNN tends to exponentially decrease for future time steps. 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, i.e. 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 & temporal information. This issue can be solved by using convolutional LSTMs, which use convolutional kernels in place of traditional neurons [10], treating each time step as an image, which is exactly how 2D frames should be processed. Due to this, both 1D and 2D convolutional LSTMs were investigated for this paper, with the main difference being that the former performs 1D convolutions on each row of the frame individually, therefore keeping the data from each photodiode separate, while the latter performs 2D convolutions on the entire frame.

Finally, transformer encoders were also tested for this project as a novel alternative to RNN based architectures. Transformers use a concept called self-attention which identifies which elements in a sequence carry most semantic value based on the other elements of the sequence [**?**], which is fundamentally different to the hidden state mechanism used by RNNs and their derivatives. They are typically used in language translation and are made up of two parts, an encoder and a decoder. The encoder converts the input into an internal representation, which can then be manipulated and decoded into a more useful format. In a study by Yüksel *et al.* [14], a transformer encoder was used to classify topics from sequences of Turkish text taken from Twitter. Although this is clearly different to the goal presented in this paper, i.e. classifying gestures based on photodiode readings, both are classification tasks using input data in which order is important. Given the positive results found in this study, it seemed reasonable to test transformer encoders for gesture recognition as well.

4.2 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

- Frame size
- Number of neurons in hidden layer

Convolutional LSTMs

- Filter kernel size

Transformer Encoder

- Number of attention heads

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 performance.

It is also worth mentioning why hidden layer count and training time are not included in the parameter tuning. Each architecture is restricted to a single hidden layer to minimize latency, but also because the relative simplicity of the input data should not require more than a single hidden layer of processing to achieve high accuracy. Training time, meanwhile, is controlled to ensure fairness when testing validation accuracy across the various neural network architectures. More detail regarding how this is done can be found in section 6.1.

5 Model Implementation

5.1 Training Code

The implementation and testing of each neural network architecture was done using the TensorFlow Python library and high-level Keras API. While the dataset is split into training and validation sets when evaluating architecture performance, the final models used on the Arduino are trained on all available data, after which they are converted into TensorFlow Lite models.

5.2 Inference Code

To run the TensorFlow Lite models created during training on an Arduino, they must first be converted into C files. 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" [13]. TODO: ADD SPECIFICS REGARDING FINAL IMPLEMENTATION ON ARDUINO.

6 Results

6.1 Testing Methodology

Accuracy Testing Methodology

To ensure that the measured validation accuracy for a given architecture was as high as possible, as well as being representative of real world performance, four distinct measures were put in place. The first of these was the addition of a dropout layer with $p = 0.5$, before the output layer, to ensure that overfitting would not be an issue [11]. The second measure was the use of k-fold cross-validation [3] with $k = 10$, as validation accuracy can be skewed based on how the test and validation sets are split. Thirdly, each model architecture was trained until the value of the loss function did not improve after 200 consecutive epochs. Once training was complete, the model weights were also reverted to the epoch after which the loss function value was lowest. This removes the variance caused by randomizing initial model weights as well as differences in model training time, and allows each model to achieve close to its maximum possible performance. Finally, the dataset was shuffled randomly before each model was validated as ordered data can cause anomalies with training and k-fold cross-validation.

Latency Testing Methodology

6.2 Testing Results

RNN

LSTM

GRU

1D Convolutional LSTM

2D Convolutional LSTM

Transformer Encoder

7 Related Work

8 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 completely reproducible. However, using the same code and hardware, it should be possible for anyone to reach the same conclusions based on their own findings.

9 Conclusion

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

[9] 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.

[10] 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, 2015.

[11] 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.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[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.

[15] Atıf Emre Yüksel, Yaşar Alim Türkmen, Arzucan Özgür, and Berna Altınel. Turkish tweet classification with transformer encoder. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2019)*, pages 1380–1387, Varna, Bulgaria, September 2019. INCOMA Ltd.