Recognising Gestures Using Ambient Light and Convolutional Neural Networks

Adapting Convolutional Neural Networks for Gesture Recognition on Resource-constrained Microcontrollers

by

William Narchi

A Dissertation
Submitted to EEMCS faculty
Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 19, 2022

Responsible Professor:	Qing Wang
Supervisors:	Mingkun Yang, Ran Zhu
Department:	Embedded and Networked Systems
Faculty:	Electrical Engineering, Mathematics and Computer Science

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

TU Delft

Abstract

This paper presents how a convolutional neural network can be constructed in order to recognise gestures using photodiodes and ambient light. A number of candidates are presented and evaluated, with the most performant being adopted for in-depth analysis. This network is then compressed in order to be ran on an Arduino Nano 33 BLE microcontroller to present its feasibility in embedded operation. The final utilised network was observed to have accuracies between 75.4% and 86.8% depending on the testing conditions. Further, all candidates were found to be sufficiently compact and low-latency for real-time operation.

Preface

Readers should keep in mind some organisational quirks regarding the structure of CSE3000 Research Project, the course which encompasses Bachelor's graduation theses for the TU Delft Computer Science and Engineering Bachelor's programme. Research projects are divided into five research questions, each of which is answered by one student. The research questions involved in my assigned research project are especially interrelated and as such, several portions of this paper will refer to the work of my colleagues in brief, and interested readers are requested to read their papers for further details. Project-wide details are kept to a minimum where possible to allot more space for this paper's contributions.

I would like to extend a show of gratitude to my supervisors, without whom the work in this paper would not have been possible. First, my responsible professor Dr. Qing Wang whose organisation and management of the research project as a whole enabled our entire team to progress and whose assistance and understanding eased the burden of writing our first contributions to the scientific community. Second, my supervisor Mingkun Yang whose experience was detrimental in shaping my understanding of convolutional neural networks, which formed the backbone of this research. Lastly, my supervisor Ran Zhu who proposed the initial architectural choices that shaped the structure of the models considered for this paper.

To conclude, I would also like to thank my project mates: Dimitar Barantiev, Femi Akadiri, Matthew Lipski, and Stijn van de Water. Without their efforts, the realisation of the entire gesture recognition system would not have been possible and I am proud of the work that we have accomplished in our time together on this project.

William Narchi
Delft, June 2022

1 Introduction

1.1. Motivation

Gesture recognition is a long-standing problem within the field of human computer interaction and concerns how particular movements can be recognised and interpreted by computers [1]. Advancements within the field hold the promise of allowing for an increase in the fluidity with which people interact with electronic devices, as well as enabling the development of new forms of interaction not possible through more traditional interfaces, such as the classical mouse and keyboard. Existing applications of gesture recognition include enriching entertainment - as exemplified by gaming peripherals such as the Microsoft Kinect [2] and the Wii Remote which accompanied the highly successful Nintendo Wii [3], [4] - in addition to more industrious solutions such as directing the take-off and landing of aerial vehicles [5], fully controlling simple robotic vehicles [1], and interpreting sign languages [6], [7].

Utilising ambient light for real-time gesture recognition offers many potential benefits, notably in terms of affordability and compactness. A significant portion of existing gesture recognition solutions require specialised hardware which relies on precisely modulating the medium being used to facilitate the recognition process [8]–[13]. Such assemblies can be expensive and obtrusive, prohibiting the range of applications of their underlying techniques. Efforts in this field have attempted to utilise pre-existing infrastructure such as ambient Wi-Fi signals and non-intrusive sound waves in order to provide cost-effective solutions for gesture recognition that do not require extensive specialised hardware [14], [15]. Ambient light is another medium that can be taken advantage of in such a manner, owing to the ubiquity of both artificial and natural lighting in the modern world, making it an accessible and readily available resource.

Employing ambient lighting in this manner has also been explored previously using both algorithmic and machine learning-based approaches that rely on data captured by photodiodes. Photodiodes are devices that sense light and output an electrical signal that corresponds to the intensity of the detected light [16]. However, these attempts operate on a relatively rich set of resources, often utilising several dozen photodiodes, leaving room for potential improvement by reducing the number of photodiodes used while attempting to maintain gesture recognition accuracy [17]–[19].

This paper aims to address the challenge of recognising gestures given the data supplied by three photodiodes. More specifically, it aims to employ **Convolutional Neural Networks (CNNs)** in order to classify sets of signals as different gestures [20]. In addition, the resulting machine learning model is adapted to run on a resource-constrained embedded platform in order to facilitate real-time gesture recognition. This approach represents an attempt to adapt techniques common in 2-D image processing in a novel manner, allowing for accurate recognition and flexible applications owing to the affordability and portability of commonly available **Single Board Computers (SBCs)** and microcontrollers [21]–[23].

1.2. Challenges

Two principal challenges define the bounds of the problem that this paper aims to tackle:

- **Construction of a machine learning pipeline to facilitate gesture recognition**
  A machine learning architecture will have to be devised in order to recognise gestures. The most important considerations are how the input data from the photodiodes can be re-structured to allow for a CNN to operate on it and the concrete structure of the neural network itself. The latter

is particularly interesting owing to the great deal of variation that exists in approaches to CNN architecture construction [24], [25].

- **Compression of the resulting model to run in real-time on an embedded platform**
  Machine learning models can be extremely substantial in size; a (somewhat extreme) example is GoogLeNet, which consists of almost 6.8 million trainable parameters [26]. Though the range of possible neural network architectures is not particularly complex owing to the relative simplicity of the input data, the final size of the model remains an important consideration, as it dictates whether or not the model can be feasibly ran on resource-constrained platforms.

1.3. Contributions

Building on the challenges defined in section 1.2, the endeavours of this paper can be summarized through three key contributions:

- **Restructuring of photodiode readings in the style of image data**
  In order to allow for a CNN to operate on the readings, the raw stream of data needs to be reshaped to resemble a 2-D image [20]. The most important consideration will be the dimensions of the image, as this will dictate the pool of potential parameters of the neural network's layers.
- **Construction of application-specific CNN architecture**
  As this application of CNNs is atypical in light of the fact that CNNs are predominantly utilised in image processing, a custom architecture will have to be engineered. Different combinations of layers and parameters need to be considered and evaluated in order to achieve a favourable classification accuracy while maintaining a reasonable model size.
- **General purpose adaptation of machine learning models for resource-constrained computing platforms**
  In order to facilitate the operation of the proposed neural networks, techniques for compressing machine learning models need to be utilised.

2

Background

This chapter briefly discusses the core theoretical concepts underlying the approaches to be presented further in the paper. Namely, these are the basic principles of convolutional neural networks, presented in section 2.1, and how quantization can be used to reduce the complexity of machine learning models, presented in section 2.2.[1]

2.1. Convolutional Neural Networks

2.1.1. Basic Principles

Convolutional neural networks are a form of machine learning, more specifically a subclass of artificial neural networks. Inputs to CNNs consist of image-like arrays of data, which have a set 'length', 'width', and number of 'channels'.[2] As the name implies, CNNs' operation primarily consists of applying convolutions to the input data presented to each convolutional layer in the network; a convolution is simply a weighted sum of one or more data points.[3] The number of data points being summed and the weights contributing to this sum are dictated by the 'kernel' being used.

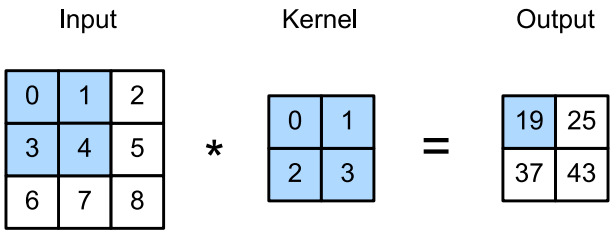


Figure 2.1: Applying a 2x2 kernel to a 3x3 array of numeric data [27].

An illustrative example can be seen in Figure 2.1. The kernel scans horizontally from left to right until it reaches the edge of the 'image', resets its horizontal position back to the very left, and then moves vertically downwards and repeats this horizontal motion until the entire array has been scanned through. This results in the following computations being carried out (in this order) to produce the given output:

$$(0 \times 0) + (1 \times 1) + (2 \times 3) + (3 \times 4) = 19$$
$$(0 \times 1) + (1 \times 2) + (2 \times 4) + (3 \times 5) = 25$$
$$(0 \times 3) + (1 \times 4) + (2 \times 6) + (3 \times 7) = 37$$
$$(0 \times 4) + (1 \times 5) + (2 \times 7) + (3 \times 8) = 43$$

Values for kernel weights are learned from the dataset during the model's training process. Unlike the given example, the weights of the kernel are not fixed for the entire convolution procedure. Different weights are used for each application of the kernel, i.e. the weights change each time the kernel takes a 'step' through the data array.

[1] It is assumed that readers are familiar with the basic workings of neural networks.

[2] Adopting this image-esque mental framework eases the understanding of many of the operations that CNNs carry out and this subsection is authored with this viewpoint in mind.

[3] These data points can be thought of as the pixels that make up the 'image'.

2.1.2. Padding and Stride

Padding is the process of adding additional 'artificial' data to the borders of a given data array. This is often done to preserve information close to the edge that would otherwise be lost by the reduction in resolution that a convolutional layer causes, or to augment low resolution data [27].

Stride refers to the size of the 'steps' that a convolutional kernel takes along each axis of a given data array. The example kernel shown earlier in Figure 2.1 has a horizontal stride of 1 and a vertical stride of 1 as well.

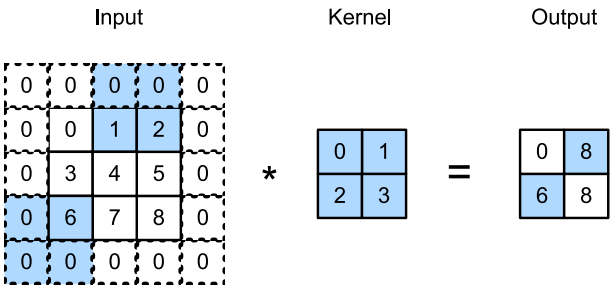


Figure 2.2: Applying a 2x2 kernel to a 3x3 array of numeric data. The data array is equally padded with zeroes across both axes, and the kernel has vertical and horizontal strides of 3 and 2 respectively. [27]

Figure 2.2 shows a combined application of padding and stride. The output values correspond to the following computations (in this order):[4]

$$(0 \times 0) + (1 \times 0) + (2 \times 0) + (3 \times 0) = 0$$
$$(0 \times 0) + (1 \times 0) + (2 \times 1) + (3 \times 2) = 8$$
$$(0 \times 0) + (1 \times 6) + (2 \times 0) + (3 \times 0) = 6$$
$$(0 \times 7) + (1 \times 8) + (2 \times 0) + (3 \times 0) = 8$$

2.1.3. Multiple Channels

As mentioned in subsection 2.1.1, layers can have multiple input channels and multiple output channels.[5] Each input channel is treated as a separate data array, complete with its own set of kernel weights that are each learned during training. Each output channel is computed as a weighted sum of the outputs of the convolutions performed on each input channel; these weights are also learned during training. Figure 2.3 shows an example of multiple input and output channels.

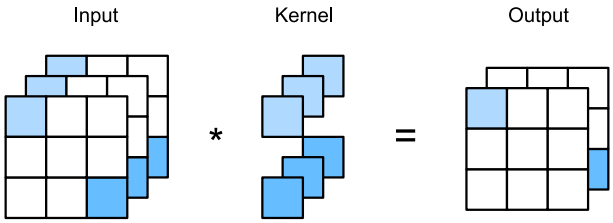


Figure 2.3: Applying a 1x1 kernel to 3 input channels, producing 2 output channels [27].

2.1.4. Pooling

Pooling layers can be thought of as extremely simplistic versions of convolutional layers. Instead of computing a weighted sum, they apply much simpler operations such as returning the maximum value or computing an unweighted average. As such, these layers do not have any trainable parameters. Besides this, pooling layers behave similarly to convolutional layers in terms of interaction with multiple input channels, and the application of padding and stride. Figure 2.4 shows an example involving a maximum pooling layer.

[4] Note that the middlemost row and rightmost column do not contribute to the computation. The former is skipped due to the size of the kernel and the value of the vertical stride, while the latter is similarly skipped as the horizontal stride causes the kernel to 'overflow' beyond the right edge of the array, thus proceeding to the next 'step' in the convolution.

[5] In line with the image analogy, one can compare CNN channels to the three RGB channels of an image.

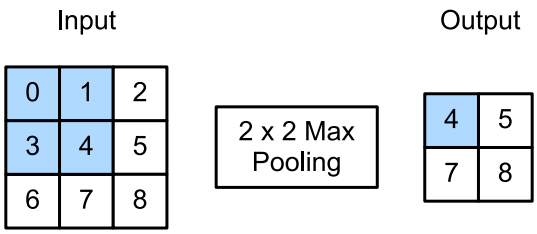


Figure 2.4: Applying a 2x2 maximum pooling layer to a 3x3 array. The procedure carried out is identical to that in Figure 2.1, except that the computation at each step is merely the maximum value instead of a weighted sum. [27]

2.2. Parameter Quantization

Quantization in the context of neural networks refers to simplifying the representation used for storing model parameters. This allows for models to be significantly compressed, greatly reducing their size and the time it takes to perform an inference; quantized models are up to four times smaller and can be almost twice as fast to execute compared to their non-quantized counterparts [28].

Generally, model parameters are stored as 32-bit floating point numbers, though this degree of precision is often not needed during inference, only during training [29]. As a result, these parameters can instead be stored as 16-bit or 8-bit numbers, most often as integers. Several techniques exist for performing this mapping and interested readers can find an overview of prevalent quantization techniques in [30].

3

System Overview

This chapter provides an overview of the goals of the presented research and the problems it aims to tackle, in addition to detailing aspects of the gesture recognition system not related to the machine learning stage, but whose mention is integral to understanding the system as a whole. Section 3.1 discusses the aims of the larger research project - of which this paper is a component - while section 3.2 presents the goals of this paper in isolation. Section 3.3 details the hardware setup of the project and summarises the processing performed on the raw photodiode signals. Section 3.4 concludes with a brief overview of the dataset being used to train and evaluate the utilised model.

3.1. Project Structure

The end goal of the five papers involved in the overarching research project is to construct a gesture recognition system using only three photodiodes. Each of these papers represents a stage in a pipeline which begins with fluctuations in photodiode readings based on hand movements and ends with the identification of a particular gesture. The research questions tackled by each of the five papers are as follows:

1. *How to design a receiver to detect visible light signals with one Arduino Nano 33 BLE and two/three OPT101 photodiodes, and how to proceed the signals efficiently?*
2. *What kind of gesture dataset should be constructed for the training purpose and how to build it? How to deal with the fact different people have different habits when performing the predefined gestures (such as left hand vs. right hand)?*
3. *What's the impact of the placement of photodiodes on the sensing performance? What's the impact of ambient light on the sensing performance?*
4. *Which model could be used for gesture recognition, based on 2-D pre-processed data (like picture recognition)? How to compress the used deep learning to make it real-time on Arduino Nano 33 BLE?*
5. *Which model could be used for gesture recognition, based on 3-D pre-processed data (like video)? How to compress the used deep learning to make it real-time on Arduino Nano 33 BLE?*

Each of these five research questions was addressed separately and in parallel with the other research questions. There exists a great deal of interdependency between them however, and so findings and results produced in the process of tackling one research question directly impacted the others.

3.2. Research Question Breakdown

This paper addresses research question 4 by utilising CNNs, which are common in the field of image processing. This research question can be divided into the following sub-questions:

1. How can a time series composed of photodiode readings be represented as a 2D image?
2. What CNN architecture is best suited for gesture recognition based on these readings?
3. Which metrics matter the most for this application, and on the basis of which different models can be compared and evaluated?
4. What constitutes 'real-time' classification and how can it be achieved given a constrained embedded environment?

3.3. Hardware and Signal Processing

The assembly used for gesture recognition consists of 3 OPT101 photodiodes and an Arduino Nano 33 BLE. It includes a number of capacitors and variable resistors which smooth the photodiode outputs and ensure that the output signal is varied according to different lighting conditions such that signal patterns pertaining to different gestures are as distinct as possible. Figure 3.1 showcases this assembly and further specifics about the assembly can be found in [31].

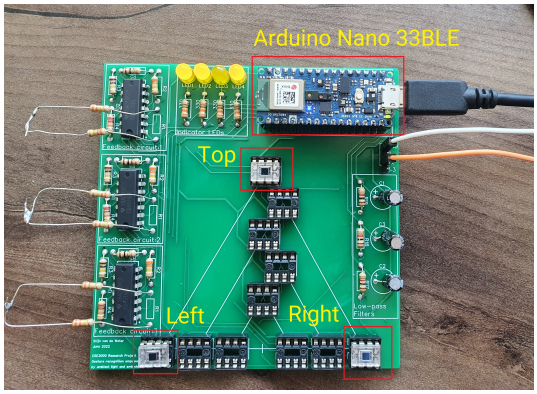


Figure 3.1: Hardware assembly used for classification. The Arduino Nano 33 BLE microcontroller and triangularly arranged OPT101 photodiodes are highlighted. [31]

Following the performance of a gesture, the raw photodiode signals are processed. First, the window of readings corresponding to the actual gesture are isolated. Following this, the signals are filtered to minimise noise, interpolated to a fixed length for processing by the neural network and normalised to values between 0 and 1. Figure 3.2 showcases the described effects and further specifics about the pipeline can be found in [32].

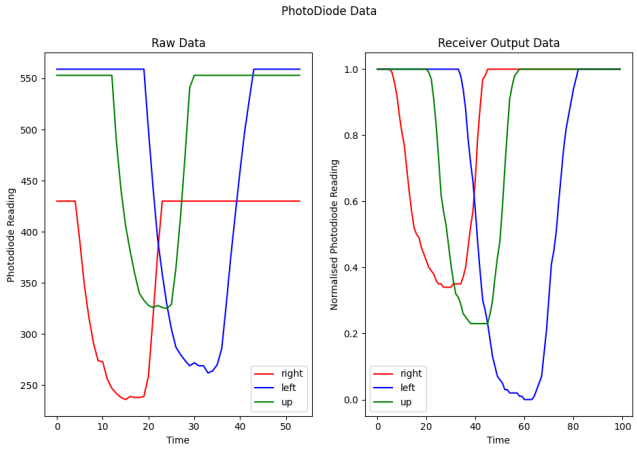


Figure 3.2: Processing pipeline effect on the signals corresponding to a leftwards swipe. [32]

3.4. Dataset

The dataset constructed for this project consists of data recorded from about 50 volunteers and encompasses 10 gestures in total. Ambient lighting conditions reflected in the dataset range from dim indoor lighting to direct sunlight, covering a light intensity range from 0 to 100,000 lux. Further, hand width, hand length, the volunteer's handedness (whether they are left-handed or right-handed), and gender were all recorded and varied in order to make the dataset balanced and representative. Data from the first 29 participants was recorded at 20Hz and all subsequent samples were recorded at 100Hz. Further details about the dataset can be found in [33].

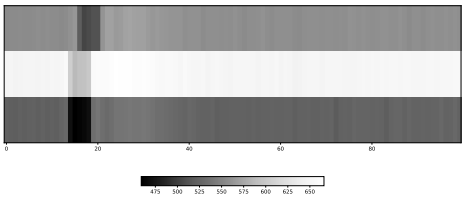
4

Design

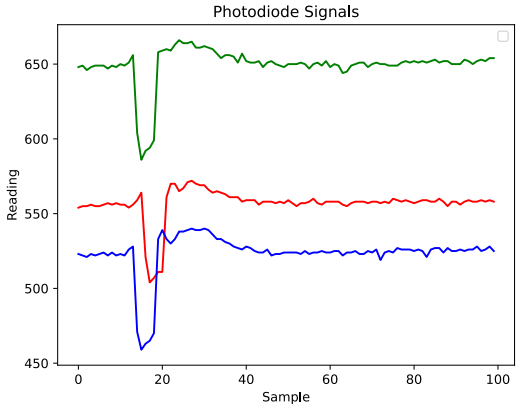
This chapter provides an overview of the design decisions made for the components of the utilised neural network. First, how the photodiode readings are restructured for input to the CNN is discussed in section 4.1. Second, the structure of the chosen network itself is detailed in section 4.2.[1] Lastly, section 4.3 details how the model is compressed for resource-constrained operation.

4.1. Data Restructuring

In order to be operated on by a CNN, photodiode readings are reshaped into an $n \times 3$ 'image' that represents a gesture attempt. Figure 4.1a shows a visualisation of this restructuring for the data corresponding to an upwards swipe and figure 4.1b shows a graph of the corresponding signals. The disruptions in the signals which act as the 'fingerprint' of the gesture can be seen as dark patches in the image and as dips in the signal graph.



(a) Visualisation of photodiode readings as an image. Note that the given image is $3 \times n$ for visibility, but the CNN operates on $n \times 3$ input data.



(b) Graph of corresponding photodiode readings.

Figure 4.1: Visualisation of photodiode readings corresponding to an upwards swipe.

4.2. Neural Network Design

The final utilised model (referred to as **Narrow LilConv Padding Pyramid (NLCPP)**) has the following structure:

[1] Several models were devised and evaluated, but only the one utilised for the final gesture recognition assembly is detailed in this chapter. An overview of the architecture of all considered models can be found in appendix A.

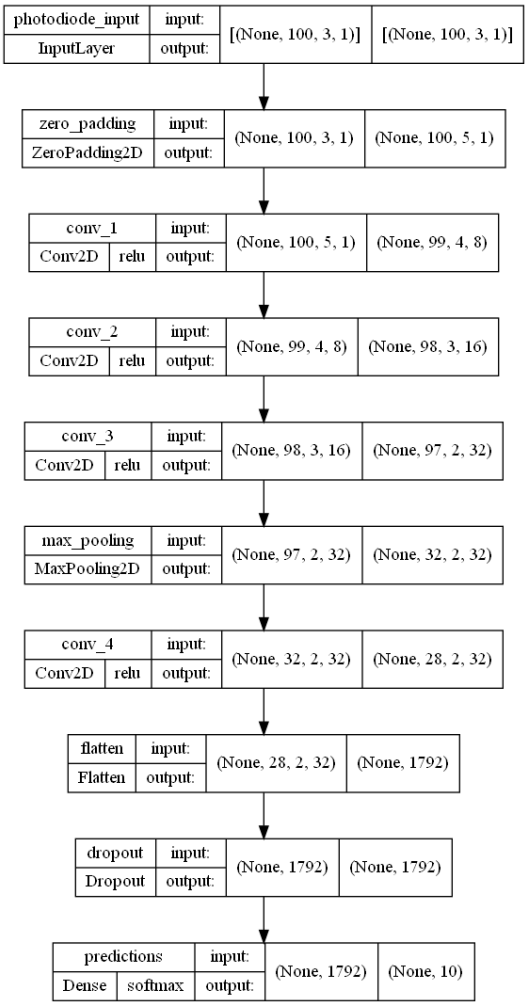


Figure 4.2: Visualisation of the structure of the chosen CNN. The ordering of the layers can be seen in addition to the input shape, output shape, and subsequent activation function of each layer.

As demonstrated by figure 4.2, NLCPP begins with an input layer that provides an input of dimensions $(100 \times 3 \times 1)$. This input is then padded with two zero-valued columns, producing a $(100 \times 5 \times 1)$ output. Following this is a sequence of three convolutional layers which primarily reduce the horizontal resolution of the image while increasing the number of output channels, producing a $(97 \times 2 \times 32)$ output. After this comes a (3×1) max pooling layer followed by a final convolutional layer, producing a $(28 \times 2 \times 32)$ output. Lastly, this final output is flattened to produce a 1792 layer of nodes that are densely connected to a 10 node layer, where each of these last 10 nodes represents the probability that the input corresponds to each of the 10 possible gestures; this final layer of nodes utilises the Softmax activation function. Each of the convolutional layers is accompanied by a **Rectified Linear Unit (ReLu)** activation function. Additionally, a dropout layer is placed between the 'flatten' layer and the 'predictions' layer in order to randomly set the input to 50% of the nodes of the 'flatten' layer to zero; this layer is only utilised during training and its inclusion helps to minimise overfitting [34].

4.3. Model Compression

Quantization provides the vast majority of the space savings leveraged for embedded operation. This is done by fully quantizing the model, such that it utilises 8-bit integer values for input, output and model parameters. Post-training full integer quantization was chosen for maximal latency and size improvements while retaining accuracy. While quantization-aware training would have potentially provided better latency and size metrics, it was observed that quantized models behave differently during training compared to their non-quantized counterparts, potentially complicating model comparison.

5 Implementation

This chapter provides an overview of the concrete implementation and tools used to realise the design details specified in chapter 4.[1] Section 5.1 discusses the implementation of the model itself and section 5.2 summarises how the model is compressed and made to run on the Arduino Nano 33 BLE microcontroller.

5.1. Model Construction

The presented models were constructed using the TensorFlow machine learning and AI library. More specifically, the high-level Keras API [35] was used to define, fit, and evaluate the models. The Adam optimizer [36] was utilised together with sparse categorical cross-entropy as a loss function. The former was configured with a learning rate of 0.001 as this was found to be a satisfactory value that allowed for relatively quick network convergence while minimising the probability of encountering a local minimum; other parameters were kept at their default value. 500 epochs were used for all training sessions as this was found to be roughly the point beyond which the tested models would begin to overfit.

5.2. Embedded Conversion and Arduino Deployment

The TensorFlow library features functionality for converting and running machine learning models on microcontrollers and embedded platforms. First, the trained model is processed by the TensorFlow Lite converter, which quantizes the model and converts it into a FlatBuffer format [37]. The entire dataset is provided as the representative dataset to be used during post-training quantization. Following this, the `xxd` command-line tool is used to convert this file into a C array so that it may be operated on by the TensorFlow Lite interpreter which runs the model on the platform of choice; this is to circumvent the lack of a file system on the vast majority of such platforms. Interested readers can find further details about TensorFlow Lite for embedded operation in [29].

In order for the TensorFlow Lite interpreter to function, a portion of memory needs to be allocated for the input and output tensors and computations that are performed during the inference process, referred to as the 'tensor arena'. The utilised network was found to require a tensor arena of 14,336 bytes as shown in figure 5.1b. Further, the interpreter must first load the code needed for the operations that the neural network performs. To achieve maximal space efficiency, only the set of operations strictly necessary for the CNN's operation are loaded as demonstrated in figure 5.1a. This results in an overall memory usage of 69,536 bytes of RAM and 162,224 bytes of flash memory.

```
1 static tflite::MicroMutableOpResolver<7> resolver;
2 resolver.AddConv2D();
3 resolver.AddFullyConnected();
4 resolver.AddMaxPool2D();
5 resolver.AddPad();
6 resolver.AddRelu();
7 resolver.AddReshape();
8 resolver.AddSoftmax();
```

(a) Source code of the operations resolver initialised with the operations strictly needed for the CNN's operation.

```
1 constexpr size_t kTensorArenaSize = 14U * 1024U;
2 uint8_t tensor_arena[kTensorArenaSize];
```

(b) Source code of the static memory allocation pertaining to the tensor arena.

Figure 5.1: Source code which facilitates the CNN's operation on the Arduino Nano 33 BLE.

[1] The full implementation of all components of the overall research project can be found at `https://github.com/StijnW66/CSE3000-Gesture-Recognition`

6

Results and Evaluation

This chapter presents an evaluation of the models considered for this paper. The raw photodiode readings gathered for the dataset were used to generate these figures; the data sampled at 20Hz was left unaltered while the data sampled at 100Hz was downsampled to 20Hz in order to conform to the input dimensions that the CNNs expect. All of the gathered samples were combined into a single contiguous dataset and randomised k-fold cross-validation was utilised with a constant random seed of 69 in order to produce constant splits and confront each of the models with identical scenarios for a fair evaluation.

Table 6.1 showcases the mean value and standard deviation of the accuracies obtained with both 5 and 10 folds; this is given for both the unoptimized and quantized versions of each of the presented models. Further, the quantized size of each model is given alongside the latency needed for a single inference to be made on the Arduino Nano 33 BLE; this is computed as the mean of 1000 inferences made on a single fixed piece of dummy data obtained from the dataset.

Model Name	Accuracy (5-fold)	Accuracy (10-fold)	Quantized Accuracy (5-fold)	Quantized Accuracy (10-fold)	Size (Bytes)	Latency (ms)
LilConv	70.315% (± 5.406%)	78.044% (± 6.561%)	9.225% (± 0.735%)	8.947% (± 0.759%)	17,329	**44**
LilConv Padding	75.367% (± 5.595%)	79.134% (± 4.934%)	70.292% (± 3.535%)	69.864% (± 2.879%)	25,089	128
LilConv Padding Lite	71.192% (± 4.299%)	73.205% (± 5.265%)	66.996% (± 2.430%)	65.861% (± 2.008%)	**15,897**	86
LilConv Padding Pyramid	71.192% (± 4.299%)	81.146% (± 4.711%)	72.091% (± 5.637%)	75.302% (± 2.969%)	37,817	81
LilConv Padding Pyramid Lite	72.284% (± 4.651%)	76.374% (± 5.266%)	70.678% (± 4.044%)	72.498% (± 3.877%)	22,681	81
Narrow LilConv Padding Pyramid	**79.220% (± 6.516%)**	**86.798% (± 5.722%)**	**75.388% (± 6.376%)**	**80.954% (± 4.753%)**	32,849	78

Table 6.1: Table showcasing the values of the metrics considered for evaluation. The row corresponding to the final chosen model is underlined and the most favourable value obtained for each metric is highlighted in bold font.

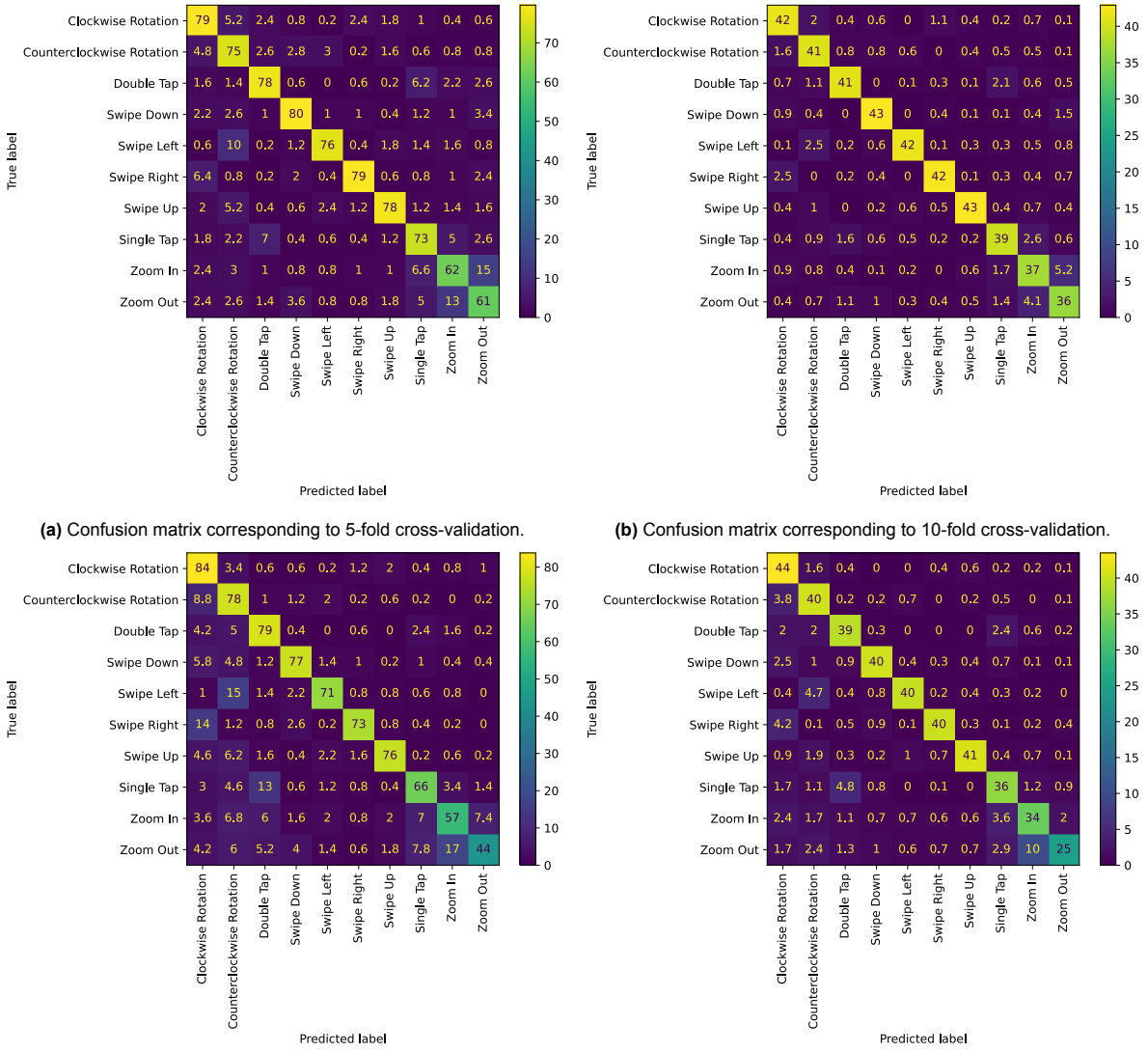
A number of trends can be observed in the results. The accuracies obtained for 10-fold validation are significantly higher than those obtained for 5-fold validation, likely as a result of the models being exposed to a greater range of lighting conditions and more variation in terms of people's preferences for how they articulate their hands in the latter compared to the former. Further, quantization seems to adversely affect accuracy for all models, causing drops between 2% and 10%, with the exception of 'LilConv' which seems to be rendered unusable after quantization. Optimized size and latency figures for all models seem sufficient to facilitate real-time operation, as the largest model registers at 37,817 bytes and the model with the greatest latency registers at 128ms.

Looking at model design trends, a few design decisions appear to prove themselves particularly advantageous. Models which feature a number of output channels that increases the further one goes along in the model seem to exhibit better performance. Additionally, the usage of an initial layer of zero padding seems to also increase performance, though conservative usage of this padding seems to be optimal as can be seen by NLCPP's usage of a smaller padding layer producing results that are more favourable than the other models. A detailing of the structure of each model can be found in appendix A in order to lend additional context to the trends noted in this paragraph.

Figure 6.1 shows the confusion matrices obtained while evaluating the NLCPP CNN, showcasing a number of trends in classification performance.[1] Each of the given matrices was computed by averaging the confusion matrices produced by each fold during each evaluation scenario. Related gestures seem to be confused fairly often; the most common false classification for the 'Clockwise Rotation' gesture appears to be 'Counterclockwise Rotation'. This trend also holds in reverse, as the most common false classification for the 'Counterclockwise Rotation' gesture also appears to be 'Clockwise Rotation'.

[1] Confusion matrices corresponding to all presented models can be found in appendix B.

This confusion of related gestures also occurs with the ‘Double Tap’ and ‘Single Tap’ gestures, as well as the ‘Zoom In’ and ‘Zoom Out’ gestures, though to a much greater degree with the latter. This is likely caused by the similarity in the signals corresponding to the two elements of each of these gesture ‘pairs’. Further, the zoom and rotation gestures appear to cause the greatest amount of confusion in classification, likely owing to the relative complexity of their signals compared to other gestures in the dataset.



(a) Confusion matrix corresponding to 5-fold cross-validation.

(b) Confusion matrix corresponding to 10-fold cross-validation.

(c) Confusion matrix corresponding to 5-fold cross-validation with quantization.

(d) Confusion matrix corresponding to 10-fold cross-validation with quantization.

Figure 6.1: Confusion matrices corresponding to various evaluations of the NLCPP CNN.

7

Related Work

This chapter presents related work in the classification portion of ambient light based gesture recognition systems. Section 7.1 begins by discussing non machine learning based algorithmic approaches which rely on observable commonalities in signals pertaining to certain gestures. Section 7.2 summarises approaches relying on traditional machine learning models, such as **Support Vector Machines (SVMs)**, **k-Nearest Neighbours (kNN)**, **Decision Trees (DTs)**, and **Random Forests (RFs)**. Section 7.3 concludes by briefly discussing deep neural network approaches.

7.1. Algorithmic Approaches

In [38], gesture recognition is facilitated by an algorithmic approach relying on detecting when different photodiodes are in shadow relative to each other. This relies on a modified version of the **Constant False Alarm Rate (CFAR)** algorithm [39] and divides the 12 gestures that make up its dataset to 5 that are performed on the hinges of a pair of glasses and 7 that are performed on a watch-like grid of photodiodes. Average precision and recall figures of 99.7%/98.3% and 99.2%/97.5% are achieved for the glasses and watch gestures respectively.

ViHand [17] divides its gestures into two sets: 8 directional sliding gestures and digital gestures, which correspond to drawing the digits 0 and 2 through 9. The former is classified by observing which photodiodes first experience a dip in their output. The latter is classified through a combination of **Dynamic Time Warping (DTW)** [40] and the kNN machine learning algorithm, discussed in the following section. The algorithmic approach achieves 100% classification accuracy for the sliding gestures.

7.2. Machine Learning

The kNN approach employed in ViHand relies on DTW as a similarity measure between different time series. This is used as the distance function for kNN classification and achieves an average recognition accuracy of 82.3% for the digital gestures.

In [18], a 6ft x 6ft platform is used for recognizing 5 full body gestures. **Principal Component Analysis (PCA)** is used to reduce the feature space of the produced time series and SVMs are utilised in a 1-vs-all configuration. An average accuracy of 96.36% is achieved.

SolarGest [41] compares the usage of SVMs, kNN, DTs and RFs for the recognition of 6 gestures. Average accuracies between 93.0% and 96.1% are achieved.

In [42], an approach similar to [18] is applied to smaller scale hand gestures. A 3x3 photodiode array is utilised to recognise 10 gestures. **Linear Discriminant Analysis (LDA)** is utilised for dimensionality reduction and kNN is used for classification with euclidean distance as the distance function. An average accuracy of 99% is achieved.

7.3. Neural Networks

Recurrent Neural Networks (RNNs) are the most commonly adopted owing to their suitability for time series data. [19] compares the usage of different RNN units for recognising 7 gestures, achieving accuracies of up to 99%.

8

Responsible Research

Conducting scientific research of any manner presents an opportunity to discover previously unknown information within a particular field of human knowledge. As such, research must be carried out in a transparent manner where every step of all relevant procedures is detailed to the greatest possible extent. This allows for research to be reproducible and verifiable such that its findings can be corroborated (or potentially falsified), in addition to allowing for the findings of the research to be expanded upon with new techniques.

In an effort to satisfy these elements of reproducibility, verifiability and transparency, this paper attempts to detail all experimental details relevant to obtaining the presented results. This is reflected in the provision of the designs of the considered models and the provided description of both the experimental setup and the procedures that were carried out in order to create this experimental setup. Interested readers can inspect the design specifics of the model used for the final gesture recognition assembly in chapter 4, make use of the source code referred to in chapter 5 so that they may re-create this paper's findings, and refer to chapter 6 for the details of the experimental setup used to produce the results presented in this paper.

9 Conclusion

9.1. Research Findings

This paper demonstrates the feasibility of utilising convolutional neural networks for gesture recognition by using photodiodes to detect fluctuations in ambient lighting caused by the hand movements pertaining to certain gestures. The considered models exhibit amicable performance given the resource-constrained 3 photodiode setup and have been shown to be compact and performant enough to facilitate real-time operation on an embedded platform, namely an Arduino Nano 33 BLE microcontroller.

9.2. Future Work

9.2.1. CNN Architecture Experimentation

The models presented in this paper were mostly devised through an iterative process of repeated tweaking, experimentation, and evaluation. A number of design choices, such as the usage of an initial padding layer and gradually increasing the number of channels in the network, were found to be advantageous to classification performance.

Further experimentation will likely yield additional improvements. Potential avenues for exploration could be the inclusion of additional padding layers in the middle of the network, changing the size of the kernels used for the convolutional layers, and the adoption of proven structural patterns present in modern CNNs, particularly those used for low-resolution image processing.

9.2.2. Integration of Processing Pipeline

Due to time constraints and the parallel nature of the overarching research project, the processing pipeline detailed in section 3.3 could not be utilised for pre-processing the dataset for this paper's purposes. This is due to unforeseen trends in the collected data causing the pipeline to exhibit unintended behaviour which negatively affects classification performance. As a fully integrated pipeline holds the potential for dramatically improving classification performance by isolating only relevant portions of the photodiode signals and removing noise, this presents a strong opportunity for realising a more performant gesture recognition system.

References

[1] M. Yasen and S. Jusoh, "A systematic review on hand gesture recognition techniques, challenges and applications," *PeerJ Computer Science*, vol. 5, e218, 2019.

[2] Z. Zhang, "Microsoft kinect sensor and its effect," *IEEE MultiMedia*, vol. 19, no. 2, pp. 4–10, 2012. DOI: 10.1109/MMUL.2012.24.

[3] S. E. Jones and G. K. Thiruvathukal, *Codename revolution: the Nintendo Wii platform*. MIT Press, 2012.

[4] M. Liebl, *Wii lifetime sales surpass 100 million units*, Jul. 2013. [Online]. Available: https://www.gamezone.com/news/wii-lifetime-sales-surpass-100-million-units/.

[5] M. Gupta, R. Rohini, P. Reddy, P. B. Prakash, and K. Kumar, "Gesture controlled metal detection land rover," *International Journal of Engineering Trends and Technology*, vol. 21, no. 05, pp. 229–231, 2016.

[6] N. Siddiqui and R. H. M. Chan, "A wearable hand gesture recognition device based on acoustic measurements at wrist," in *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2017, pp. 4443–4446. DOI: 10.1109/EMBC.2017.8037842.

[7] C. Zhang, Y. Tian, and M. Huenerfauth, "Multi-modality american sign language recognition," in *2016 IEEE International Conference on Image Processing (ICIP)*, 2016, pp. 2881–2885. DOI: 10.1109/ICIP.2016.7532886.

[8] A. Tewari, B. Taetz, F. Grandidier, and D. Stricker, "[poster] a probabilistic combination of cnn and rnn estimates for hand gesture based interaction in car," in *2017 IEEE International Symposium on Mixed and Augmented Reality (ISMAR-Adjunct)*, IEEE, 2017, pp. 1–6.

[9] D. L. Quam, "Gesture recognition with a dataglove," in *IEEE Conference on Aerospace and Electronics*, IEEE, 1990, pp. 755–760.

[10] S. Rani, K. Dhrisya, and M. Ahalyadas, "Hand gesture control of virtual object in augmented reality," Sep. 2017, pp. 1500–1505. DOI: 10.1109/ICACCI.2017.8126053.

[11] M. Yuntao, L. Yuxuan, J. Ruiyang, *et al.*, "Hand gesture recognition with convolutional neural networks for the multimodal uav control," *Piscataway: IEEE*, pp. 198–203, 2017.

[12] I. Alnujaim, H. Alali, F. Khan, and Y. Kim, "Hand gesture recognition using input impedance variation of two antennas with transfer learning," *IEEE Sensors Journal*, vol. 18, no. 10, pp. 4129–4135, 2018.

[13] Y. Zhang, C. Cao, J. Cheng, and H. Lu, "Egogesture: A new dataset and benchmark for egocentric hand gesture recognition," *IEEE Transactions on Multimedia*, vol. 20, no. 5, pp. 1038–1050, 2018. DOI: 10.1109/TMM.2018.2808769.

[14] S. Gupta, D. Morris, S. Patel, and D. Tan, "Soundwave: Using the doppler effect to sense gestures," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2012, pp. 1911–1914.

[15] T. Zengshan, W. Jiacheng, Y. Xiaolong, and Z. Mu, "Wicatch: A wi-fi based hand gesture recognition system access," *IEEE Access*, vol. 6, pp. 16 911–16 923, 2018.

[16] T. P. Pearsall, *Photonics essentials*. McGraw-Hill Education, 2003.

[17] Q. Hu, Z. Yu, Z. Wang, B. Guo, and C. Chen, "Vihand: Gesture recognition with ambient light," in *2019 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, 2019, pp. 468–474. DOI: 10.1109/SmartWorld-UIC-ATC-SCALCOM-IOP-SCI.2019.00122.

[18] R. H. Venkatnarayan and M. Shahzad, “Gesture recognition using ambient light,” *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 2, no. 1, pp. 1–28, 2018.

[19] H. Duan, M. Huang, Y. Yang, J. Hao, and L. Chen, “Ambient light based hand gesture recognition enabled by recurrent neural network,” *IEEE Access*, vol. 8, pp. 7303–7312, 2020.

[20] S. Albawi, T. A. Mohammed, and S. Al-Zawi, “Understanding of a convolutional neural network,” in *2017 international conference on engineering and technology (ICET)*, Ieee, 2017, pp. 1–6.

[21] Chris, *Arduino cost guide: How much for boards, kits, components*, Aug. 2021. [Online]. Available: https://chipwired.com/arduino-cost-guide/.

[22] A. Allan, *A cheaper single-core esp32 module?* May 2022. [Online]. Available: https://www.hackster.io/news/a-cheaper-single-core-esp32-module-53fa9c4143b5#:~:text=If%5C%20you're%5C%20looking%5C%20for,around%5C%20%5C$2.40%5C%20in%5C%20low%5C%20volumes..

[23] V. Mohan, *Raspberry pi models comparison: Choose the right pi for you!* May 2021. [Online]. Available: https://raspberryexpert.com/raspberry-pi-models-comparison/.

[24] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, “A survey of deep neural network architectures and their applications,” *Neurocomputing*, vol. 234, pp. 11–26, 2017.

[25] A. Bashar *et al.*, “Survey on evolving deep learning neural network architectures,” *Journal of Artificial Intelligence*, vol. 1, no. 02, pp. 73–82, 2019.

[26] C. Szegedy, W. Liu, Y. Jia, *et al.*, “Going deeper with convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[27] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, “Dive into deep learning,” *arXiv preprint arXiv:2106.11342*, 2021.

[28] Martín Abadi, Ashish Agarwal, Paul Barham, *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/.

[29] P. Warden, *Tinyml*. Sebastopol, CA: O’Reilly Media, Jan. 2020.

[30] R. Krishnamoorthi, “Quantizing deep convolutional networks for efficient inference: A whitepaper,” *arXiv preprint arXiv:1806.08342*, 2018.

[31] Stijn van de Water, “Designing an adaptable and low-cost system for gesture recognition using visible light,” 2022.

[32] Dimitar Barantiev, “Designing a software receiver for gesture recognition with ambient light,” 2022.

[33] Femi Akadiri, “Constructing a dataset for gesture recognition using ambient light,” 2022.

[34] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[35] F. Chollet *et al.*, *Keras*, https://keras.io, 2015.

[36] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2014. DOI: 10.48550/ARXIV.1412.6980. [Online]. Available: https://arxiv.org/abs/1412.6980.

[37] Google, *Flatbuffers: Memory efficient serialization library*. [Online]. Available: https://github.com/google/flatbuffers.

[38] Y. Li, T. Li, R. A. Patel, X.-D. Yang, and X. Zhou, “Self-powered gesture recognition with ambient light,” in *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, 2018, pp. 595–608.

[39] L. L. Scharf and C. Demeure, *Statistical signal processing: detection, estimation, and time series analysis*. Prentice Hall, 1991.

[40] M. Müller, “Dynamic time warping,” *Information retrieval for music and motion*, pp. 69–84, 2007.

[41] D. Ma, G. Lan, M. Hassan, *et al.*, “Solargest: Ubiquitous and battery-free gesture recognition using solar cells,” in *The 25th Annual International Conference on Mobile Computing and Networking*, 2019, pp. 1–15.

[42] M.-D. A. Kaholokula, “Reusing ambient light to recognize hand gestures,” 2016.

A

Model Designs

This appendix contains visualisations of all of the models considered for this paper. The structure of each model's layers is showcased similarly to section 4.2.

A.1. LilConv

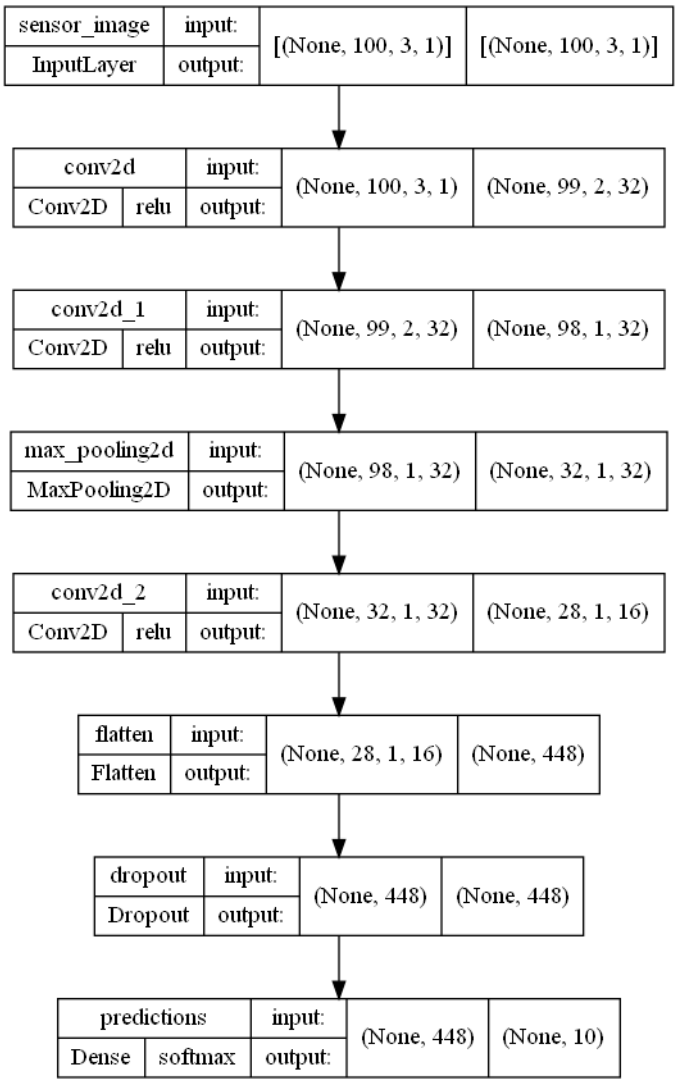


Figure A.1: Visualisation of the structure of the 'LilConv' CNN. The ordering of the layers can be seen in addition to the input shape, output shape, and subsequent activation function of each layer.

A.2. LilConv Padding

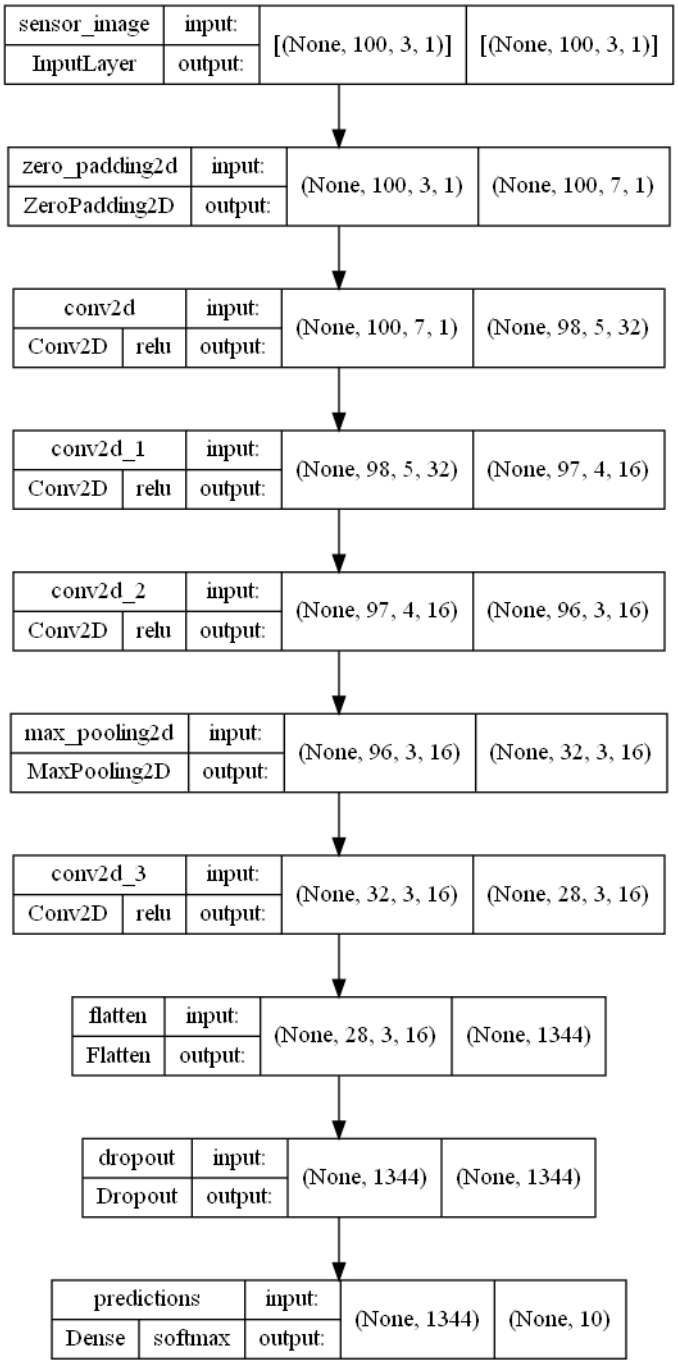


Figure A.2: Visualisation of the structure of the 'LilConv Padding' CNN. The ordering of the layers can be seen in addition to the input shape, output shape, and subsequent activation function of each layer.

A.3. LilConv Padding Lite

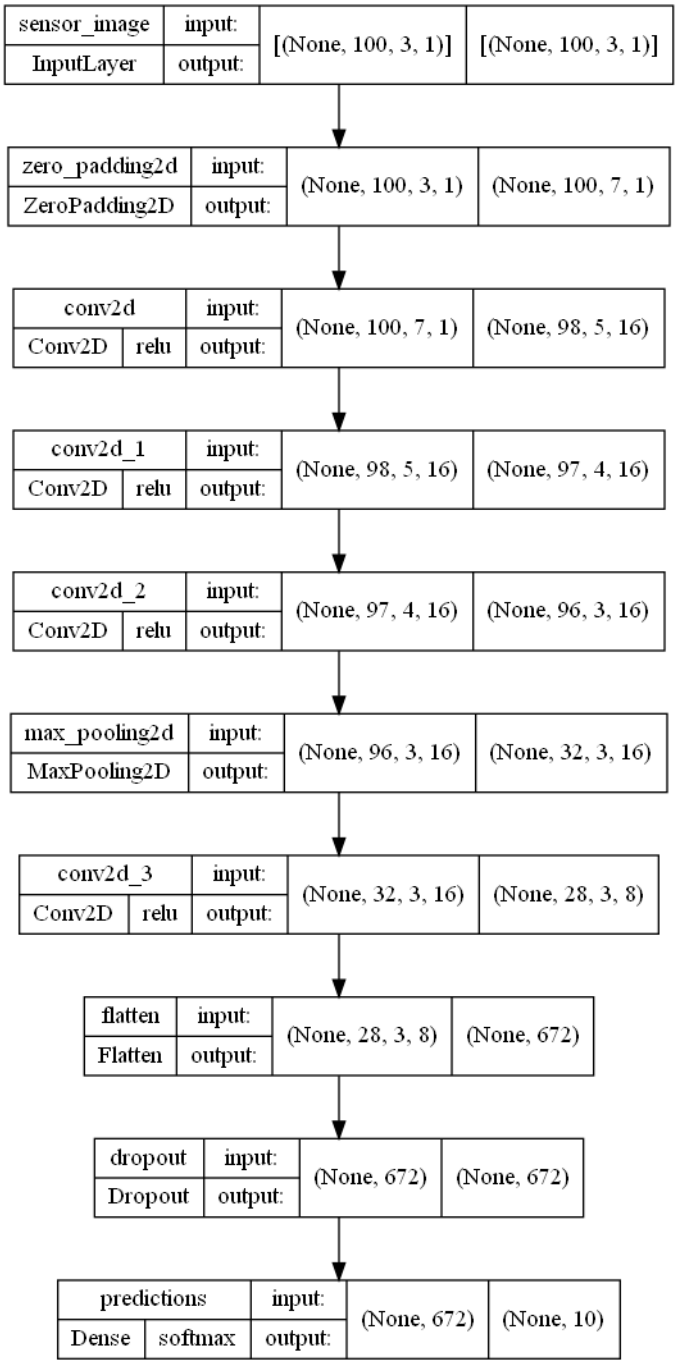


Figure A.3: Visualisation of the structure of the 'LilConv Padding Lite' CNN. The ordering of the layers can be seen in addition to the input shape, output shape, and subsequent activation function of each layer.

A.4. LilConv Padding Pyramid

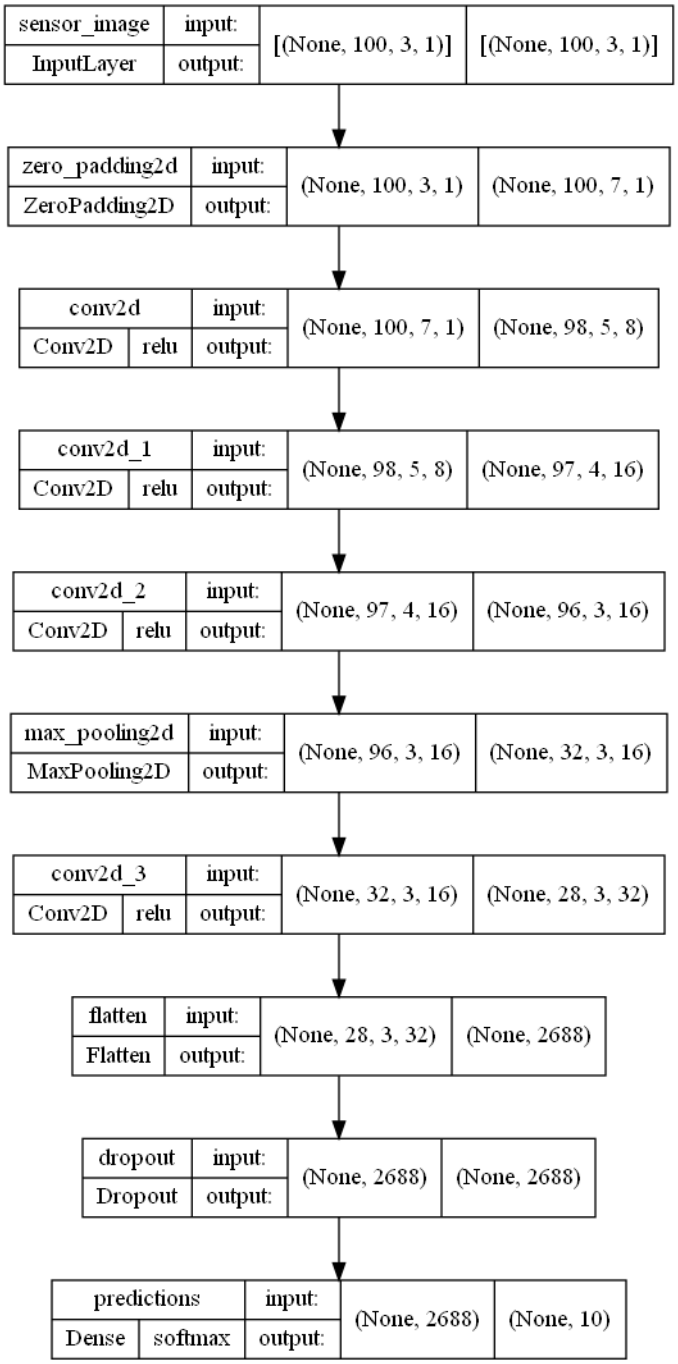


Figure A.4: Visualisation of the structure of the 'LilConv Padding Pyramid' CNN. The ordering of the layers can be seen in addition to the input shape, output shape, and subsequent activation function of each layer.

A.5. LilConv Padding Pyramid Lite

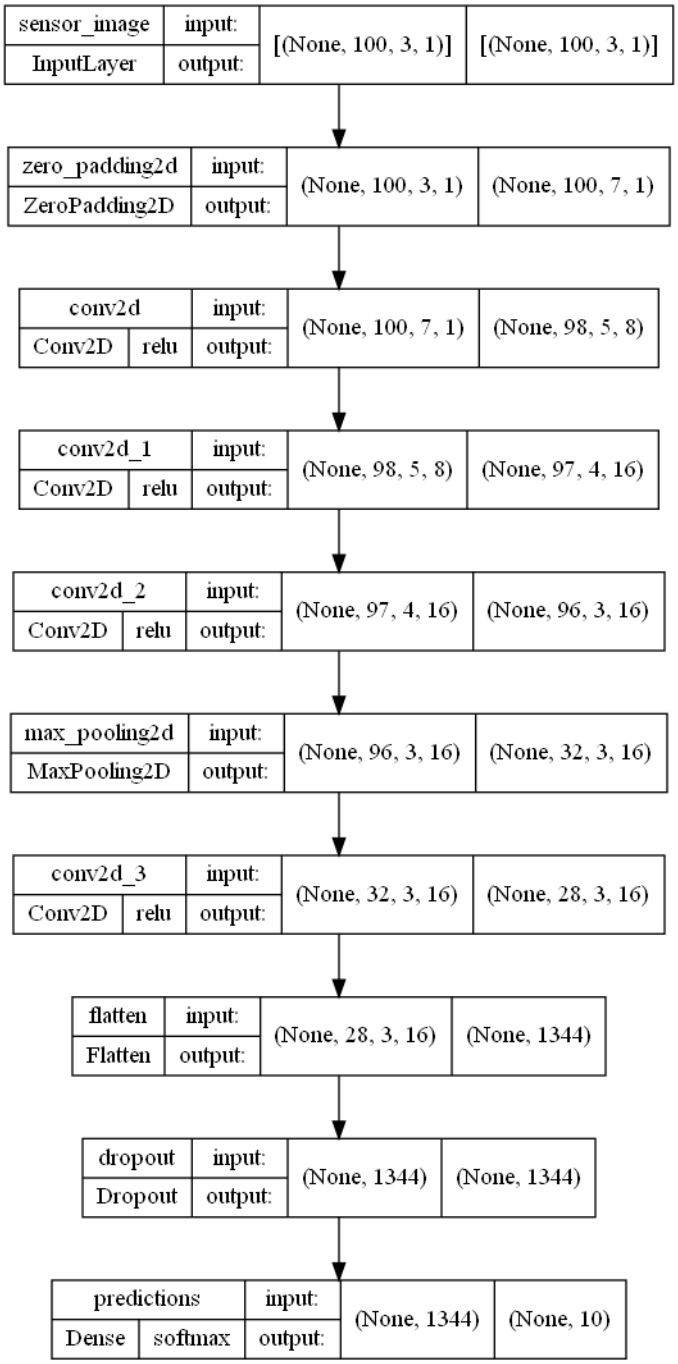


Figure A.5: Visualisation of the structure of the 'LilConv Padding Pyramid Lite' CNN. The ordering of the layers can be seen in addition to the input shape, output shape, and subsequent activation function of each layer.

A.6. Narrow LilConv Padding Pyramid (NLCPP)

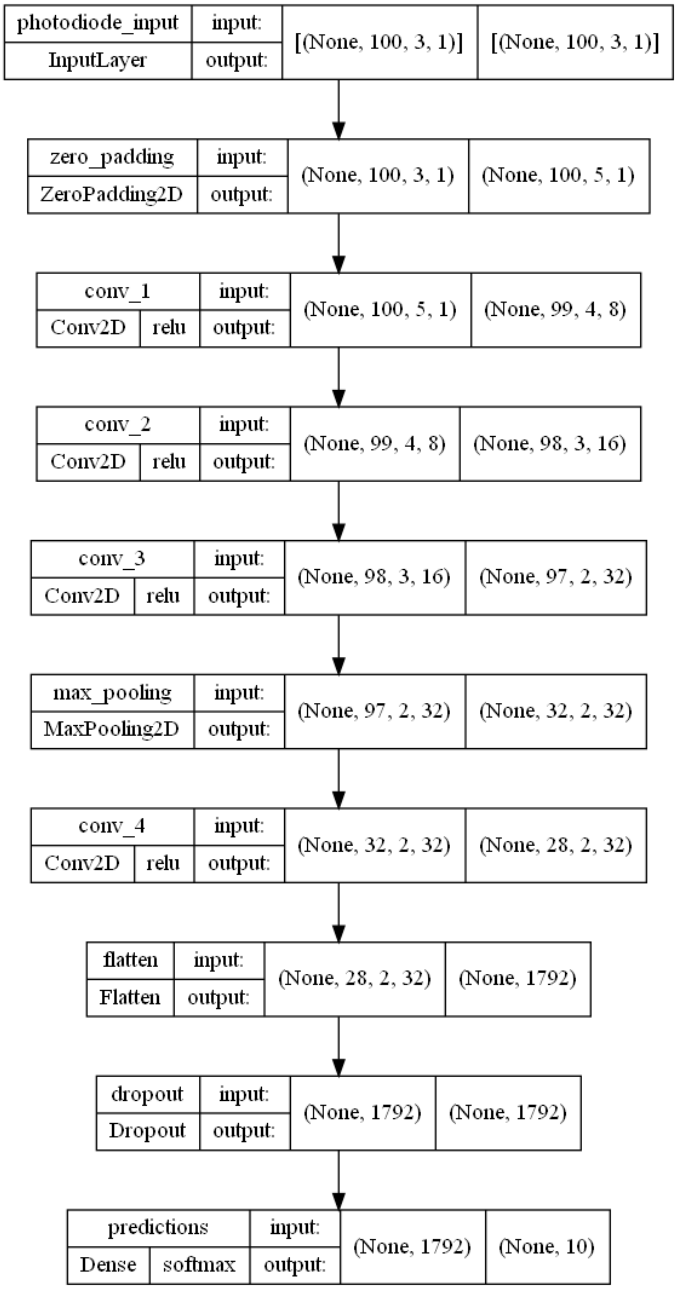


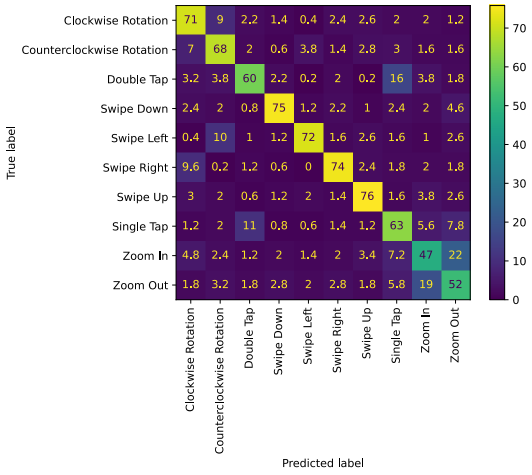
Figure A.6: Visualisation of the structure of the 'Narrow LilConv Padding Pyramid (NLCPP)' CNN. The ordering of the layers can be seen in addition to the input shape, output shape, and subsequent activation function of each layer.

B

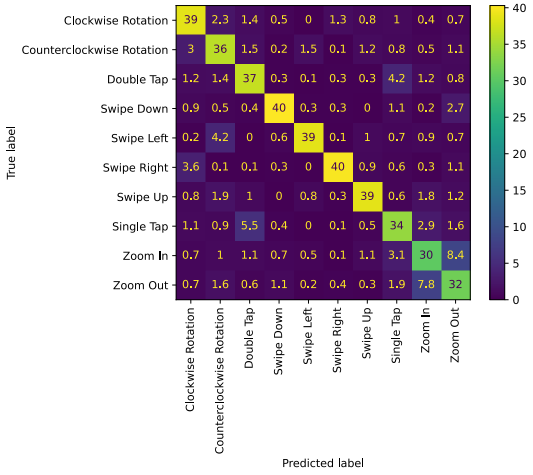
Confusion Matrices

This appendix contains the confusion matrices obtained while evaluating the models considered for this paper. They are presented with the same structure and were computed in the same manner as the confusion matrices for the NLCPP CNN in chapter 6.

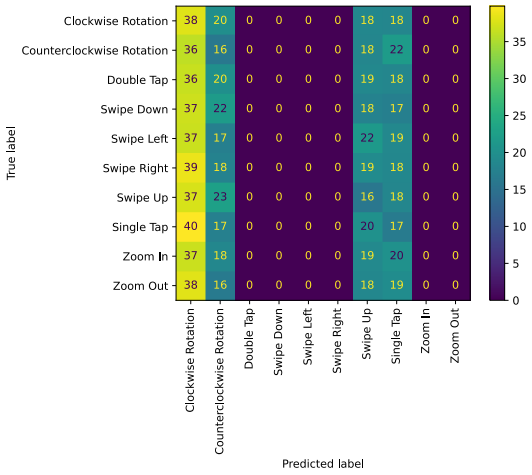
B.1. LilConv



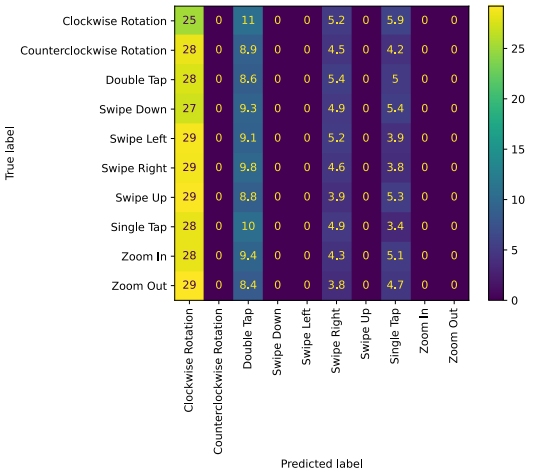
(a) Confusion matrix corresponding to 5-fold cross-validation.



(b) Confusion matrix corresponding to 10-fold cross-validation.



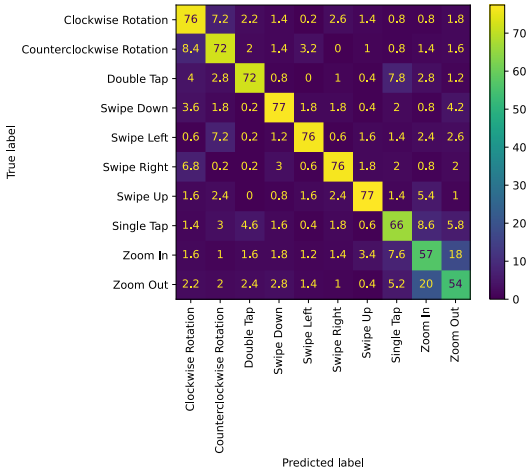
(c) Confusion matrix corresponding to 5-fold cross-validation with quantization.



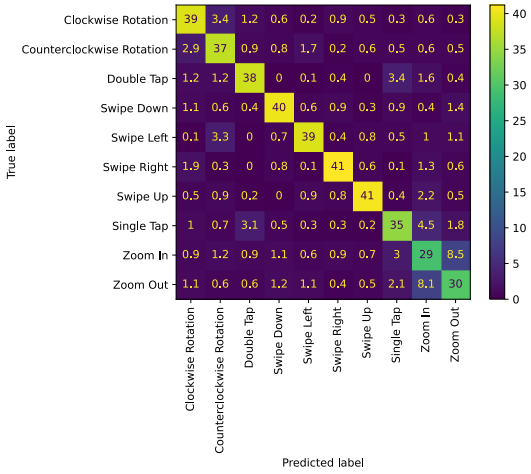
(d) Confusion matrix corresponding to 10-fold cross-validation with quantization.

Figure B.1: Confusion matrices corresponding to various evaluations of the 'LilConv' CNN.

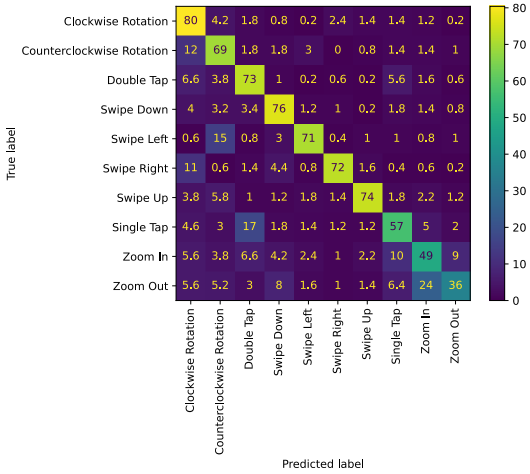
B.2. LilConv Padding



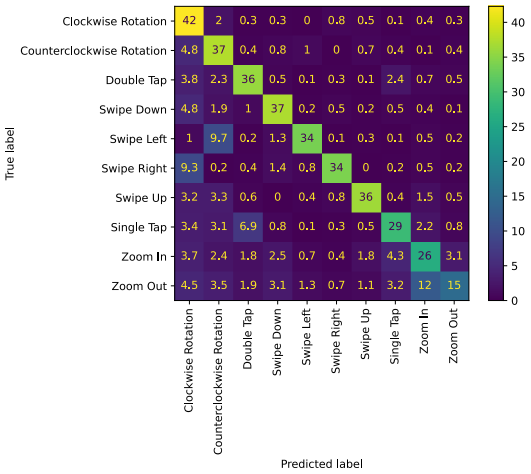
(a) Confusion matrix corresponding to 5-fold cross-validation.



(b) Confusion matrix corresponding to 10-fold cross-validation.



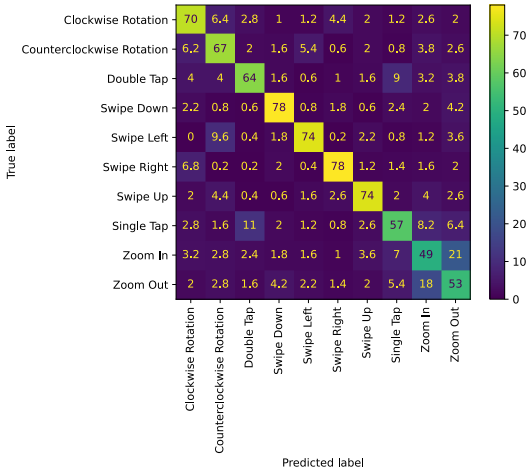
(c) Confusion matrix corresponding to 5-fold cross-validation with quantization.



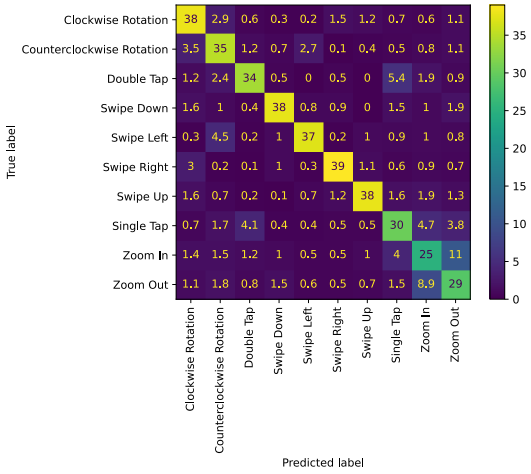
(d) Confusion matrix corresponding to 10-fold cross-validation with quantization.

Figure B.2: Confusion matrices corresponding to various evaluations of the ‘LilConv Padding’ CNN.

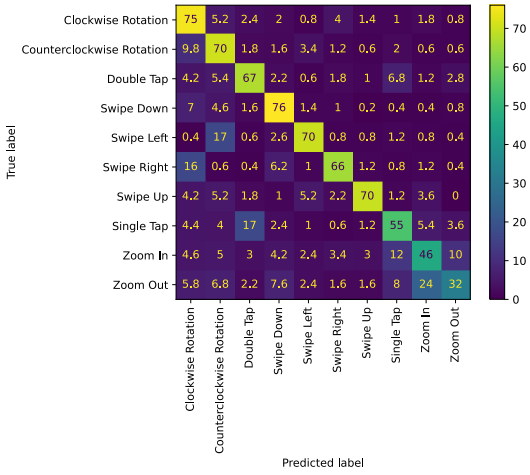
B.3. LilConv Padding Lite



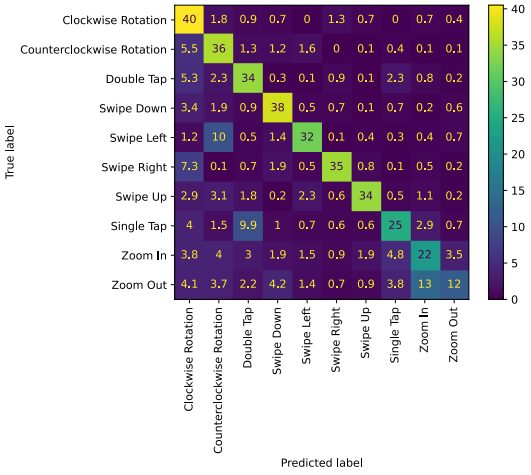
(a) Confusion matrix corresponding to 5-fold cross-validation.



(b) Confusion matrix corresponding to 10-fold cross-validation.



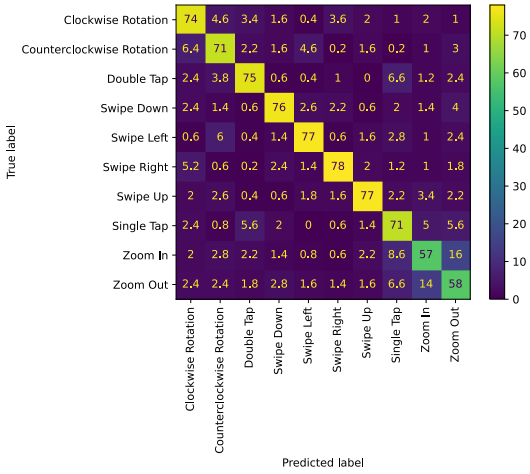
(c) Confusion matrix corresponding to 5-fold cross-validation with quantization.



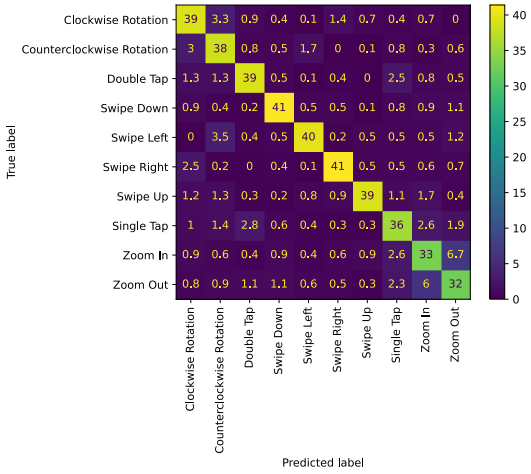
(d) Confusion matrix corresponding to 10-fold cross-validation with quantization.

Figure B.3: Confusion matrices corresponding to various evaluations of the ‘LilConv Padding Lite’ CNN.

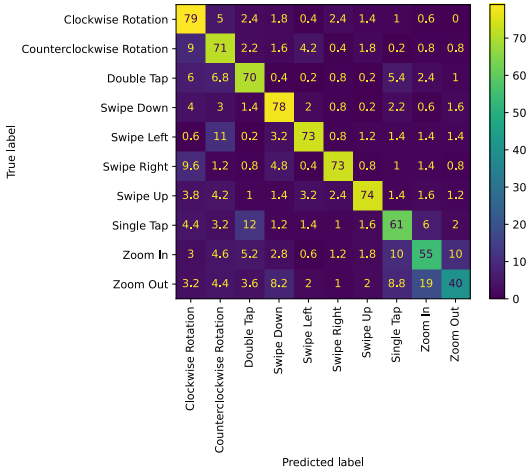
B.4. LilConv Padding Pyramid



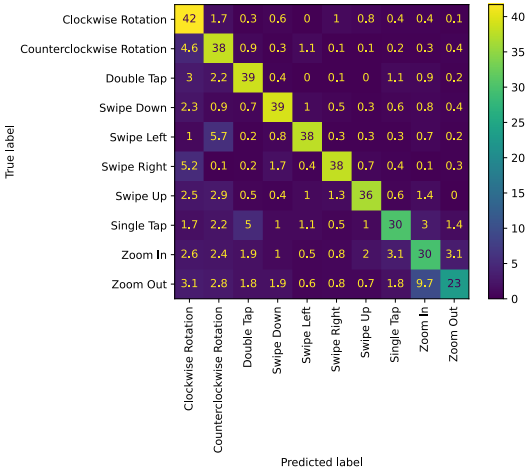
(a) Confusion matrix corresponding to 5-fold cross-validation.



(b) Confusion matrix corresponding to 10-fold cross-validation.



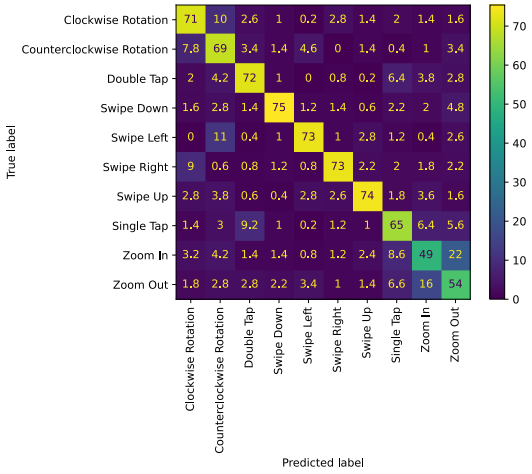
(c) Confusion matrix corresponding to 5-fold cross-validation with quantization.



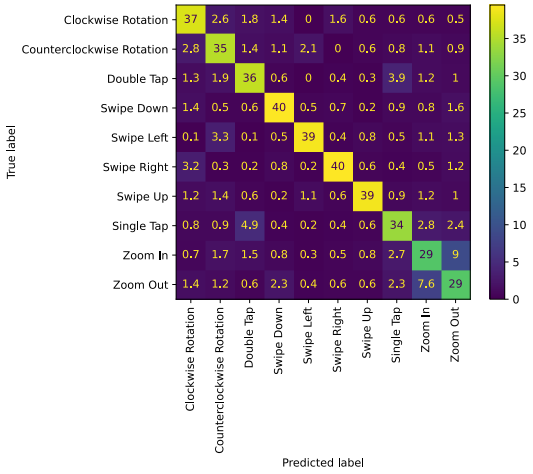
(d) Confusion matrix corresponding to 10-fold cross-validation with quantization.

Figure B.4: Confusion matrices corresponding to various evaluations of the 'LilConv Padding Pyramid' CNN.

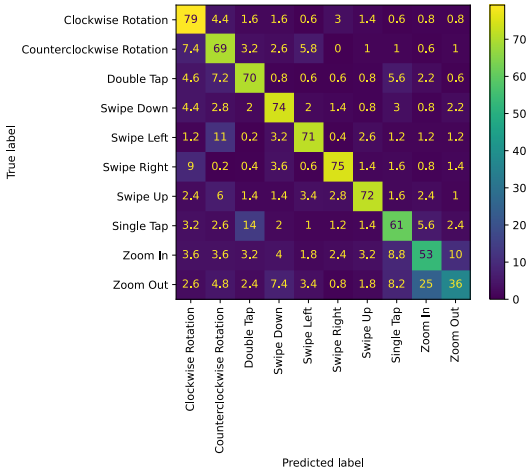
B.5. LilConv Padding Pyramid Lite



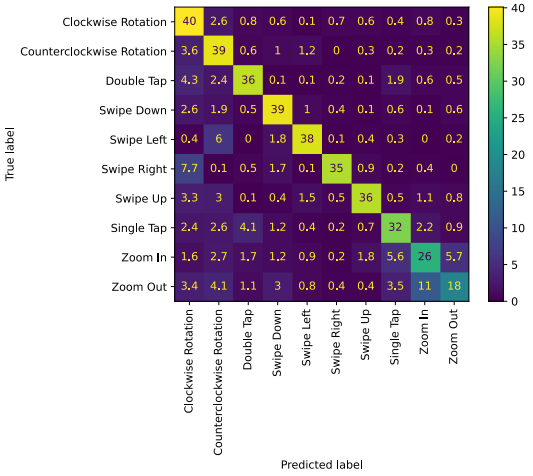
(a) Confusion matrix corresponding to 5-fold cross-validation.



(b) Confusion matrix corresponding to 10-fold cross-validation.



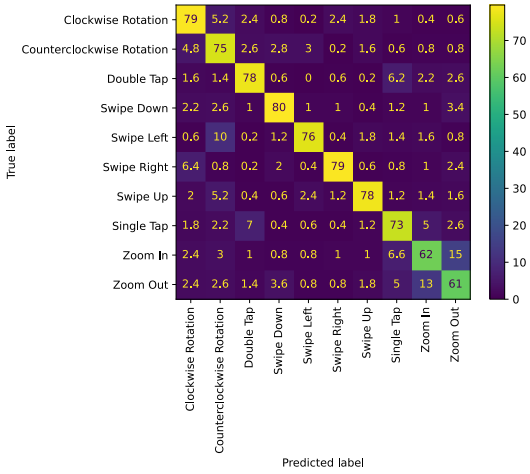
(c) Confusion matrix corresponding to 5-fold cross-validation with quantization.



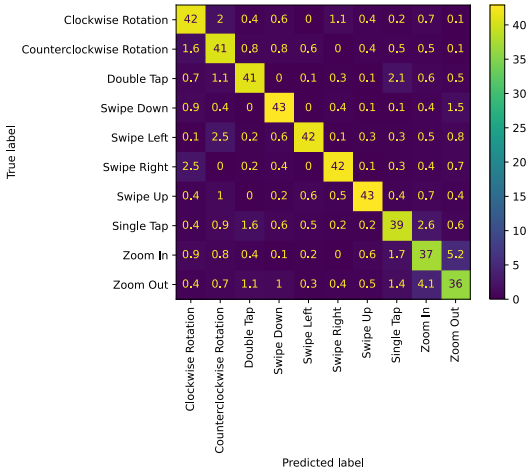
(d) Confusion matrix corresponding to 10-fold cross-validation with quantization.

Figure B.5: Confusion matrices corresponding to various evaluations of the ‘LilConv Padding Pyramid Lite’ CNN.

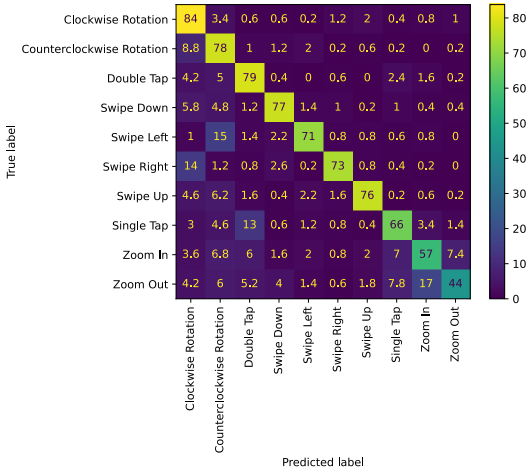
B.6. Narrow LilConv Padding Pyramid (NLCPP)



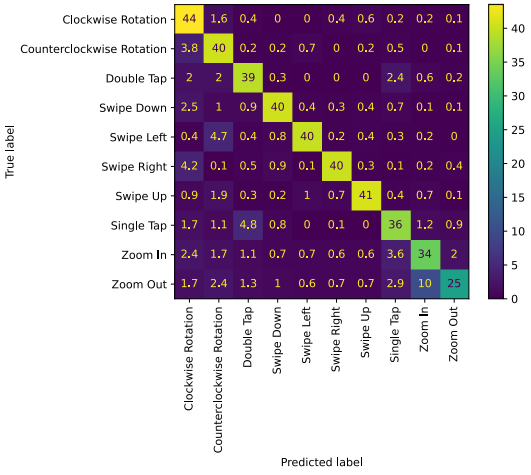
(a) Confusion matrix corresponding to 5-fold cross-validation.



(b) Confusion matrix corresponding to 10-fold cross-validation.



(c) Confusion matrix corresponding to 5-fold cross-validation with quantization.



(d) Confusion matrix corresponding to 10-fold cross-validation with quantization.

Figure B.6: Confusion matrices corresponding to various evaluations of the ‘Narrow LilConv Padding Pyramid (NLCPP)’ CNN.