# Radar Imagery: Hand Gesture Recognition

A DISSERTATION

*submitted in partial fulfillment of the requirement for the degree of*

**Master of Technology**

*in*

**Computer Technology**

*by*

**Chandragupta**

(2019EET2341)

Under the guidance of

**Dr. Seshan Srirangarajan and Dr. Sumantra Dutta Roy**


Department of Electrical Engineering
Indian Institute of Technology Delhi
June 2021

## CERTIFICATE

This is to certify that the work contained in this thesis titled **"Radar Imagery: Hand Gesture Recognition"** by **Chandragupta** (2019EET2341) in partial fulfillment of the requirement for the degree of *Master of Technology* in *Computer Technology* program of the Department of Electrical Engineering, Indian Institute of Technology Delhi, is a bonafide work carried out by him under our supervision. The matter submitted in this dissertation has not been admitted for the award of any other degree anywhere unless explicitly referenced.

Signed:
**Dr. Seshan Srirangarajan**
Project Supervisor
Assistant Professor
Dept. of EE, IIT Delhi
Dated:

Signed:
**Dr. Sumantra Dutta Roy**
Project Co-Supervisor
Professor
Dept. of EE, IIT Delhi.
Dated:

## ACKNOWLEDGEMENT

I am feeling very fortunate and delight to present this thesis, written testimony of my research work. I take this good fortune to express sincere gratitude towards my thesis supervisor Dr. Seshan Srirangarajan and co-supervisors, Dr. Sumantra Dutta Roy for selecting me to work on this prominent area of research. This project is inter-disciplinary project which involve Radar sensing, signal processing and deep learning. At early state it was difficult to completely understand Radar imagery and deep learning involved. I am fortunate to have expert supervision providing me continuous source of domain knowledge related to radar, signal processing and machine learning. This Covid-19 pandemic bring hard time for all of us ,in spite of that I am grateful to my supervisor for regularly guiding and motivating me in my research work. I am indebted to them for holding on me throughout the research work. I would like to thank my supervisors for providing me honorarium for my research work. I want to acknowledge the continuous support of Texas Instrument for troubleshooting the hardware and providing us datasets and Mr.Sandeep Rao for their valuable suggestion and guidance.
This work would not be possible without the computational power provide by High-Performance computing facility of IIT Delhi. Remote IEEE login provided great resource for research and study. I would like to thank my project partner Mr. Parveen Bajaj, and Dr Gaurav Jaiswal for having detail discussion, sharing of ideas, concepts and continuously motivating me. My sincere appreciation to all my dear CTech batch-mates, and all those friends whose name could not be mentioned here due to lack of space, for cheering me, motivating me, and for being a part of my life.
Lastly, yet importantly, I would like to express my profound, genuine, and heartfelt thanks to my parents, Mrs. Radha Devi and Mr. Krishna Kumar Sahu, and my elder brother Mr. Kautilya Gupt for their unconditional love.

**Chandragupta**

16 June 2021

## ABSTRACT

In this work we explore human computer interaction through dynamic hand gestures using radar as the input sensor. Radar has the capability to capture fine gestures of the human hand and due to various innovation in semiconductor technologies and reduced size, radar can be embedded in mobile devices and can be operated at ultra low power consumption. We have used Texas Instrument (TI) IWR6843 radar module with mmWAVEICBOOST and DCA1000 EVM module. We explore various machine learning algorithms including deep learning models to classify a range of gestures. First we planned to use our own data set but we were not able to collect data due to the Covid-19 pandemic. Therefore, we have used two publicly available data sets from Soli [22] and TinyRadarNN [16], and a data set provided by TI. Due to the high temporal resolution of the radar we use various sequential deep learning models. First, we use CNN to extract features from range-doppler images generated from radar and used various sequential models such as RNN, TCN on these features. We also eliminate 2D CNN and use 1D CNN as the sequence model on hand crafted features and achieve state of the art accuracy of 98% with 11 gestures on the Soli data set. This not only increases accuracy from 87% [22] but also uses only $441,195$ parameters as compared to $46,763,532$ parameters in Soli's model. This improvement in gesture recognition accuracy and smaller size of the model and the ability to use parallel computations make it ideal for resource constrained mobile devices.

However we were not able to achieve any significant improvement in gesture recognition accuracy with the TinyRadar data set because this data contains only five frames per gesture instance (as compared to Soli which has 40 to 100 frames). Finally we eliminated four fine gestures and explored gesture recognition with the remaining seven gestures. We also used the raw data from radar and found that raw data does not improve accuracy as compared to the range-Doppler images.

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 Motivation

Human computer interaction (HCI) is an active area of research. Since the advent of computers, we have witnessed the miniaturization of computers from the size of a large room to small form factors such as in mobile phones and smart watches. Accordingly, the input devices range from stylus pen to touch interfaces to hand gestures. Touch free interfaces such as hand gesture reorganisations have shown promising results such as with leap motion sensors [12] and Kinect depth map cameras [17]. Most input sensing modalities from capacitive sensing to camera based systems have their pros ans cons. Capactive sensing such as touch screens do not have fine tracking capability (in the range of millimeters), do not work with overlapping gestures, and are limited to a 2D input space. Camera based solutions also do not capture fine hand gestures, overlapping gestures, and do not work across different material and illumination conditions.

Next we consider the frequencies at which the different sensing modalities work as shown in Fig. 1.1. We see that capacitive sensors work in range of kHz to MHz whereas cameras work in the visible spectrum ($10^{14}$ Hz) and mmWave radars work in GHz which is in between that of capacitive and camera sensors. Traditionally radars are bulky devices used in civil aviation and military applications. Due to the technological advancements in the field of semiconductors, radars in very small form factors with

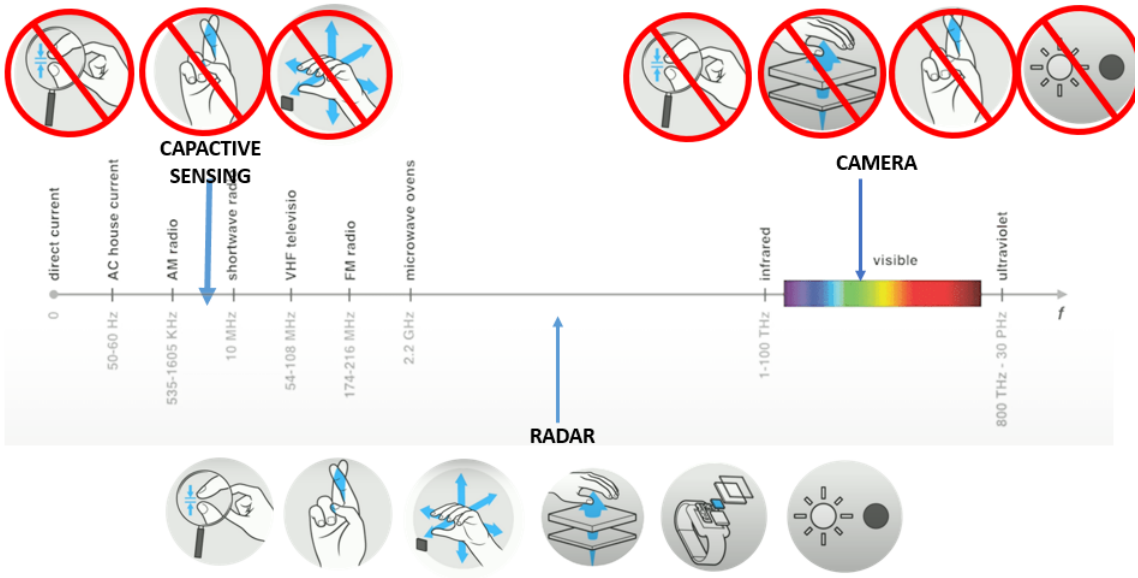



Figure 1.1: Working of different sensing techniques along the frequency spectrum (created using images from google).

low power consumption have been made possible. These radars can have a range of a few meters and are suitable for various tasks. They are being studied for applications such as self-driving vehicles and robotics. Our work is inspired from the Google Soli device [6,11] which introduced a tiny radar chip that can be embedded in any electronic device. In a follow-up paper [22], deep learning methods were investigated for gesture recognition using the Soli radar sensor.

## 1.2 Problem Statement

Radar has two sets of antenna: one set for transmitting the mmWave signal and a second set for receiving the reflected signals. Based on these signals we can determine the range, speed, and angle of the objects in the field of view of the radar. The signal obtained by mixing the transmitted and reflected signals is known as the intermediate frequency (IF) signal. The IF signal is passed through a filter and an analog to digital converter (ADC) to generate the I and Q samples. This output of the ADC is processed to generate range, Doppler, and angle information as described in Chapter 4.

In particular, we obtain range-Doppler image (RDI) and range-angle image (RAI)

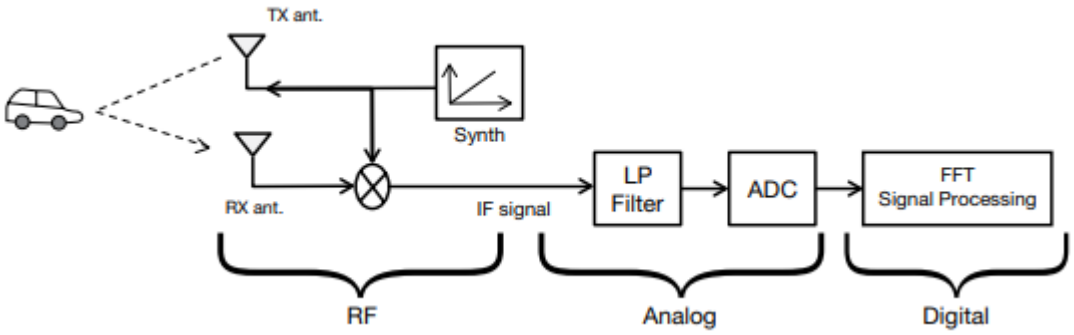



Figure 1.2: Radar Pipeline (Source: Texas Instruments)

by processing the raw data from the ADC. RDI is a 2D image having range along one axis and velocity or Doppler along the other axis. Similarly, RAI is a 2D image having range and angle along the two axes. Any motion in the field of view of the radar generates a sequence of RDIs/RAIs. Hence the data can be expressed as a 3D tensor with one dimension being time and the other two being the RDI/RAI image. Since the radar may have multiple receive antenna, a fourth dimension is also possible in such cases with the data represented as a 4D tensor.

Few hand gestures and the corresponding RDIs are shown in Fig. 1.3. As we can see very little spatial information is present in an RDI, but as we perform a hand gesture (or very fine gestures such as rubbing finger) we obtain RDIs with very high temporal resolution. Our objective is to use this high temporal resolution combined with the spatial resolution to infer about the gesture being performed.

In addition, we need to be aware of the computational complexity and memory requirements of the proposed solution as these radar sensors are proposed to be used in resource constrained electronic devices. Large models or models with high computational requirements may not be very practical.

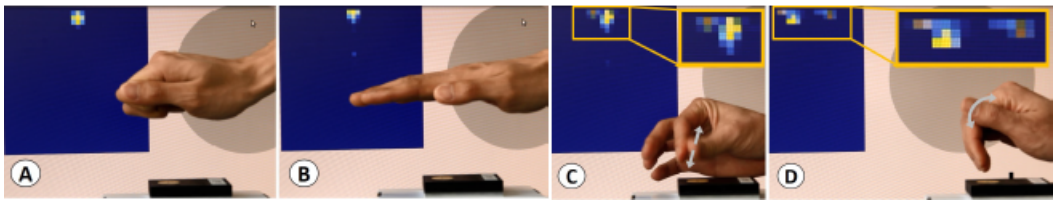



Figure 1.3: Gestures and corresponding RDI (Source: [22]): Pixel intensity corresponds to reflected energy: horizontal axis is velocity and vertical axis is range. (A+B) Sensor produces almost identical response for static objects even of different shapes. (C+D) In contrast, the sensor resolves fine motion with high resolution.

## 1.3 Organization of the Thesis

This thesis has been organised so as to provide an understanding of hand gesture classification using the radar sensor as an input device. The thesis has been organised into the following chapters:

- Chapter 2 presents a survey of the research that has been done in this area. We describe the various methods used in this area and how it enabled us to understand the problem better.

- In Chapter 3, we describe the radar hardware set-up and the operating procedure. The radar set-up is fairly complex and there are a number of software dependencies which must be taken care of.

- Chapter 4 describes the data collection process that was followed while collecting data from 10 persons performing 13 different gestures across multiple sessions. In this chapter we also presents details about the raw data, process for reading the raw binary data, and process it in order to generate RDI and RAI.

- In Chapter 5, we describe the gesture recognition task. As we were not able to collect our own data earlier due to the Covid-19 pandemic (we have collected the data now, but its processing has not been completed yet), we have used two open source data sets from [16] and [22], and a data set provided by Texas

Instruments. This chapter is divided into three sections one for each of these data sets, describing the machine learning models used and their results.

- In Chapter 6 we present concluding remarks and future research directions.

# Chapter 2

# Literature Survey

Since the advent of computers, exploration of novel input devices to allow an operator to control computers goes back decades to the first pen interfaces by Ivan Southerland [20], the first computer mouse by Engelbart [5], and early multi-touch capacitive screens by Lee and Buxton [10]. Despite the significant progress in input devices, the need for developing newer input interfaces/devices has not declined, and remains an active research area in both academia and industry.

In the last two decades we have seen exponential growth in mobile computing. Currently more people use mobile computers than traditional desktop computers (68% vs 29%) [4]. With the development of more powerful mobile devices, newer modes of interaction have emerged such as camera [12, 17], touch screens [10], and voice control [18]. Brain-computer interface is another area of active research [13, 25]. The choice of particular input device depends on factors such as use cases, feasibility, form factor, and power consumption.

There has been a lot of research in the area of camera based hand gesture recognition. Camera based images have very high spatial resolution with relatively low temporal resolution. Camera based gesture recognition such as leap motion sensors [12] and Kinect depth map cameras [17] have shown promising results and are being used for gaming and augmented reality. But in certain applications, cameras may not be the best choice either due to privacy issues, lighting conditions, absence of line of sight, or power consumption (refer Fig. 1.1). Due to these challenges, researchers have been

exploring other sensing modalities. These include infra-red (IR) proximity sensors [2], magnetic field sensing [7], and electric field sensing [9].

Between camera, which works at high frequencies corresponding to visible light ($10^{14}$ Hz), and other low frequency ($10^4$ Hz) methods mentioned above, we have the radio frequency (RF) spectrum. Some work using RF signals for sensing has been explored such as use of GSM signals [27] and leveraging WiFi signals [14]. These use frequencies in the range of 5 GHz or lower which does not give spatial resolution. Google introduced a new radar sensor Soli [6, 11] with a small form factor, ultra low power consumption and operating in the 60 GHz band. This high frequency allowed sensing of fine gestures.

Unlike camera, radars have very high temporal resolution and low spatial resolution. Therefore, existing solutions for gesture recognition using cameras are not suitable for radar based gesture recognition. Our work is inspired by ultra low power radar hardware introduced by Google ATAP [6] and its use for gesture recognition by Lien *et al.* [11]. They describe the Soli radar sensor hardware platform and demonstrate its capabilities. They also describe the working of the hardware, data pre-processing steps, and its use for gesture recognition using classical machine learning methods. Wang *et al.* [22] used the Google ATAP Soli 60 GHz radar and collected data for 11 gestures. They used deep learning models by training CNN and LSTM on range-Doppler images in an end-to-end manner and were able to achieve 87% average gesture recognition accuracy. They used fine gestures such as *finger rub, finger slide, pinch index, pinch pinky* and other relatively large motion gestures such as *push, pull, swipe, circle*. However their proposed models were large in size and not suitable for resource constrained mobile devices. Scherer *et al.* [16] used the same set of 11 gestures (but data was collected using different hardware) and achieved an accuracy of 86% accuracy with a very small model and demonstrated that it is suitable for resource constrained devices. However the performance of their model for fine gestures is not satisfactory.

Range-Doppler images (RDIs) are generated by performing two FFTs (range FFT and Doppler FFT) on the raw ADC data at the radar receiver. Apart from generating RDIs and using it as input we can use different pre-processing algorithms to generate frequency-time spectrogram (micro-Doppler), range-time and Doppler-time plots, and range-angle images.

Kim *et al.* [8], Zhang *et al.* [26] and Amin *et al.* [1] use frequency-time spectrogram (micro-Doppler). In [8], authors used deep convolutional network with frequency-time spectrogram as input data. They use a set of 10 gestures, which includes *left to right and right to left swipe, clockwise and counterclockwise rotation, push, double push, hold, and double hold* and achieve accuracy of 85%. Similarly, in [26] authors use a set of four gestures, *circle, square, tick and cross*, to train a CNN using frequency-time spectrogram and achieved 98% accuracy. Authors in [1] use the positive and negative frequency envelopes of frequency-time spectrogram and use the k-nearest neighbor (kNN) classifier. They used 15 gestures which are grouped into five classes and achieved 96% accuracy. Their results show that micro-Doppler signatures can vary depending on the aspect angle and distance from the radar.

Sakamoto *et al.* [15] use radar echo in-phase/quadrature (I/Q) plot trajectories. Radar echo trajectories are converted to low-resolution images and CNN is used to train and classify gestures. This technique achieved an average accuracy of 91.3% for ten participants. However it uses gestures involving large motion and is not suitable for fine gestures. Suh *et al.* [19] and Choi *et al.* [3] use range-time and Doppler-time plot extracted from RDIs. Authors in [19] use 24 GHz radar to collect data from 7 gestures which includes *left, right, clockwise circle, anti-clockwise circle, push, draw X, draw reverse X*. Range-time and Doppler-time plots were concatenated and used with an LSTM encoder. This method achieved an accuracy of 91%. Similar techniques were used in [3] with 60 GHz Soli radar with 10 gestures. This method successfully distinguishes 10 gestures with a high classification accuracy of 99.10%. However these methods did not employ fine gestures.

Wang *et al.* [23] convert range-Doppler image sequences to range-Doppler feature map similar to the GoogLeNet [21] architecture. These features are then converted to range-time and and Doppler-time data which is passed to LSTM encoder. They achieve 96% accuracy on 10 gestures involing large motion. Yu *et al.* [24] used range-Doppler images with range-angle images on 11 gestures and achieve improved results by fusing the RDIs and RAIs. It is observed that RAIs provide better performance than RDIs.

Most of these research efforts except [16] and [22] use different set of gestures and thus comparing these methods is a difficult task. In addition, most of these techniques (except [16] and [22]) do not use fine gestures. Since classifying gestures involving large

motions is a relatively easier task, direct comparison of these results is not reasonable. Model size and computational complexity are other key aspects which determine their suitability for use on resource constrained mobile devices. Thus we use a 60 GHz radar with a wide range of gestures including fine gestures as well as gestures involving large motion. We also consider the aspects of model size and computational complexity.

# Chapter 3

# Hardware Platform

## 3.1 Hardware Details

We used a Texas Instruments mmWave radar sensor for our work. This hardware platform is based on three electronic boards as listed below and their details are given in the following sections.

i IWR6843ISK-ODS

ii MMWAVEICBOOST

iii DCA 1000 EVM

We used a desktop with Windows 10 OS since the software for interfacing with the radar hardware is available only for Windows.

## 3.2 IWR6843ISK – ODS:

IWR6843ISK–ODS is a millimeter wave (mmWave) radar sensor. It comprises of 3 transmitting units and 4 receiving units. It is to be mounted on the MMWAVEIC-BOOST board using two 60-pin high density pin connectors. It is to be powered with

3.3 V supply which is delivered by the MMWAVEICBOOST board. Fig. 3.1 shows the front and back of the IWR6843ISK-ODS board.

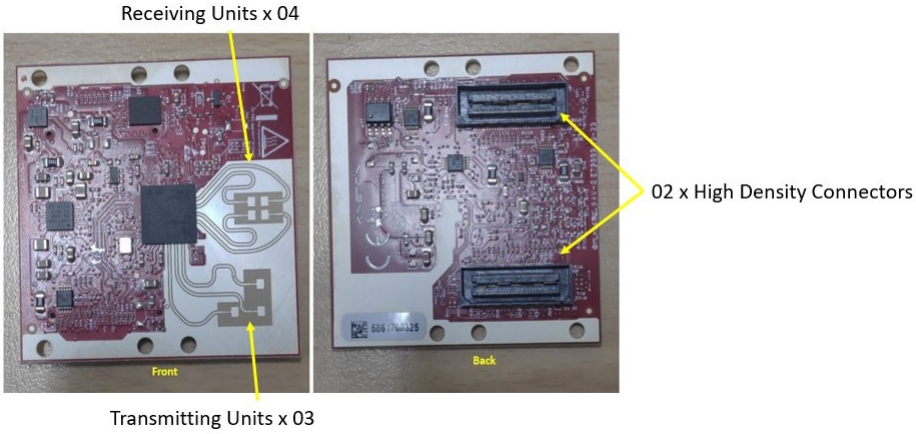



Figure 3.1: Radar sensor board (IWR6843ISK-ODS)

## 3.3 MMWAVEICBOOST

The MMWAVEICBOOST is an add-on board used with the mmWave sensor (IWR6843ISK) starter kit to provide further interfaces and PC connectivity to the sensor. It operates on a 5 V external DC supply and the board can be connected to a PC uing a micro USB cable.

Once connected, the MMWAVEICBOOST board provides an interface between the mmWave Studio tool (software application on a PC) and the radar sensor board. MMWAVEICBOOST board has one analog mux control switch and three sense on power (SOP) connectors. The analog mux setting along with the SOP jumpers control the behaviour of the MMWAVEIC board. Fig. 3.2 shows front and back of the MMWAVEICBoost board.

## 3.4 DCA1000EVM

Data capture adaptor DCA1000EVM is used to capture the raw analog-to-digital converter (ADC) data. The board is powered by a 5 V external DC adaptor and is

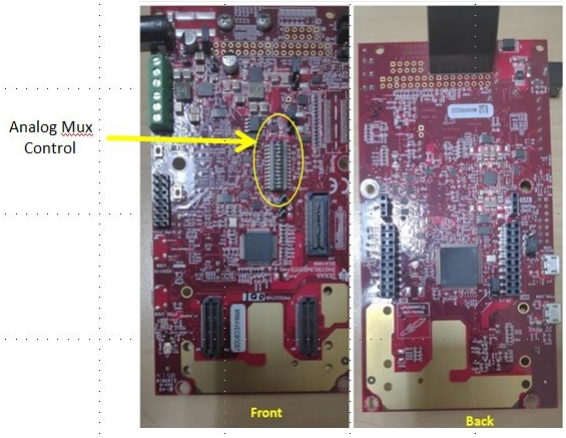



Figure 3.2: MMWAVEICBOOST board

connected to a PC via an ethernet cable and a micro USB cable. The samtec cable connects this board to the MMWAVEICBOOST board. Fig. 3.3 shows the DAC1000EVM board.

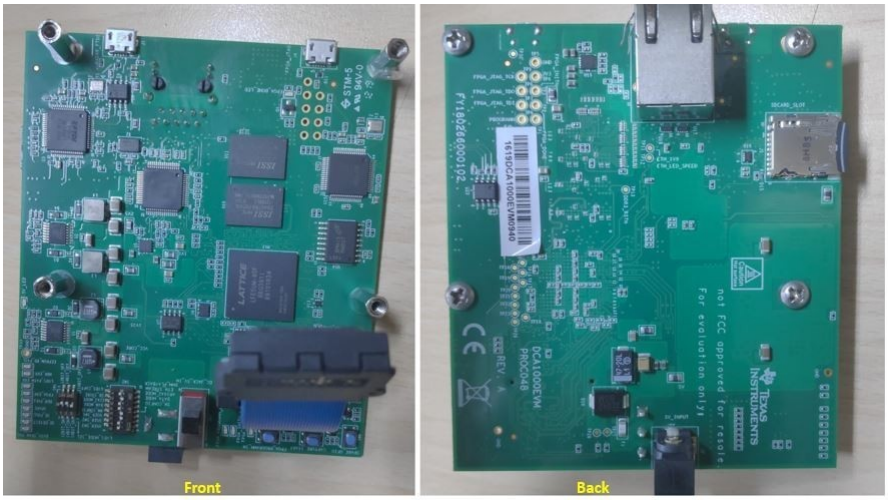



Figure 3.3: DCA1000EVM board

## 3.5 Hardware Setup

Fig. 3.4 shows how the 3 boards connect to each other. The raw data is captured from the DCA1000EVM board. Therefore, we will first set MMWAVEICBOOST board to DCA1000EVM by using the analog mux settings and SOP jumpers as shown in Fig. 3.5.

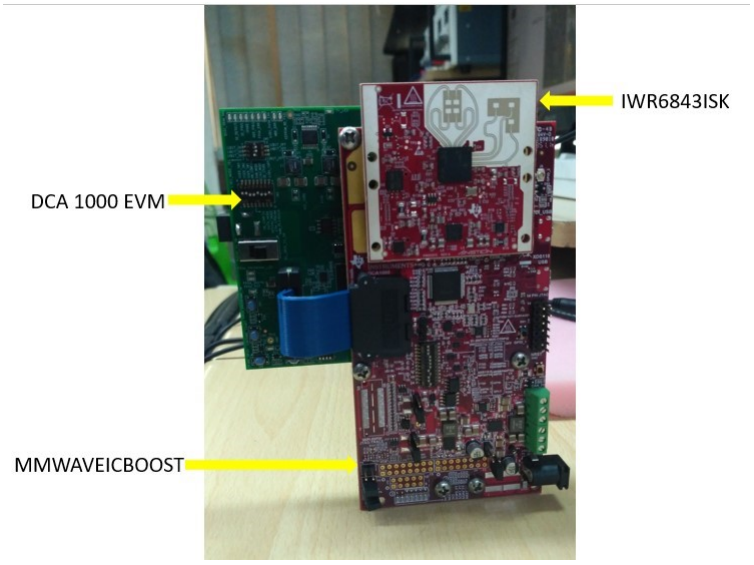
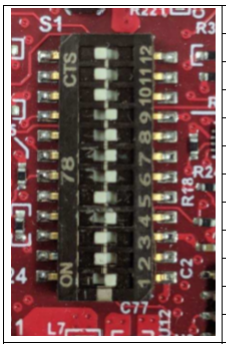



Figure 3.4: Assembly of the three boards connected together.




| | |
|---|---|
| S1.12 | ON |
| S1.11 | ON |
| S1.10 | ON |
| S1.9 | OFF |
| S1.8 | OFF |
| S1.7 | OFF |
| S1.6 | OFF |
| S1.5 | ON |
| S1.4 | OFF |
| S1.3 | ON |
| S1.2 | ON |
| S1.1 | OFF |

| Reference Designator | | Usage | Description |
|---|---|---|---|
| P6 | SOP0 | SOP[2:0] | 011 (SOP mode 2) = development mode |
| P5 | SOP1 | | 001 (SOP mode 4) = functional mode |
| P4 | SOP2 | | 101 (SOP mode 5) = flash programming |

Figure 3.5: Analog MUX and SOP settings

## 3.6 Operating Procedure

1. Connect USB micro cable between PC and MMWAVEICBOOST board's XDS1000 connector.

2. Connect USB micro cable between PC and DCA 1000 EVM's FTDI connector.

3. Connect 5 V external power supply to MMWAVEIC boost.

4. Connect 5 V external power supply to DCA 1000 EVM. Complete setup is shown in Fig. 3.6

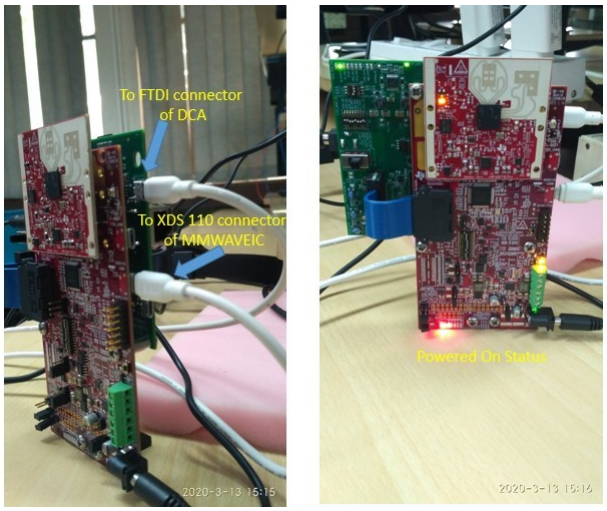



Figure 3.6: USB and power connections.

5. Disable the PC firewall (if any)

6. Set PC's ethernet IP to Static IP address 192.168.33.30 by the following steps:

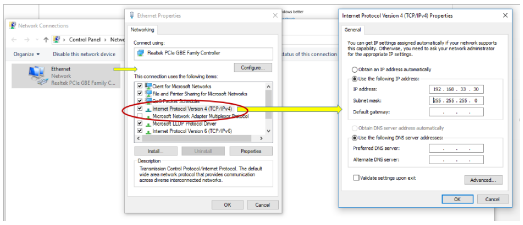
   a. Type Ethernet Settings in Windows Search and click on "Change Ethernet Settings"

   b. On the right side of the screen, you will find "Change Adaptor Settings". Select it.

   c. On the following screen, right click on "Ethernet Network" symbol and select properties. Then double click on "Internet Protocol Version 4" and enter the IP address as shown in Fig. 3.7a.

7. Open the device manager and note down the XDS110 /User UART com port number. In our case, it is "COM3" (refer Fig. 3.7b).

8. Run the MMWAVE studio from
   "C:\ti\mmwave_studio_02_01_00_00\mmWaveStudio\RunTime\mmWaveStudio"
   The window as shown in Fig. 3.8 will come up.



(a) Static IP


(b) COM Port

Figure 3.7: Screenshot of IP address and COM port settings.

i. Our board is of E.2 version and thus we will be using MMWAVE Studio 2.1

ii. Press the buttons marked '1' to '6' on the screen in sequence. At RS232 Operations, select COM Port as the one we noted down in step 7 above. Note: Make sure that at "Load(3)" and "Load(4)" options, we download version 2.1 "radars" and "masters", respectively.

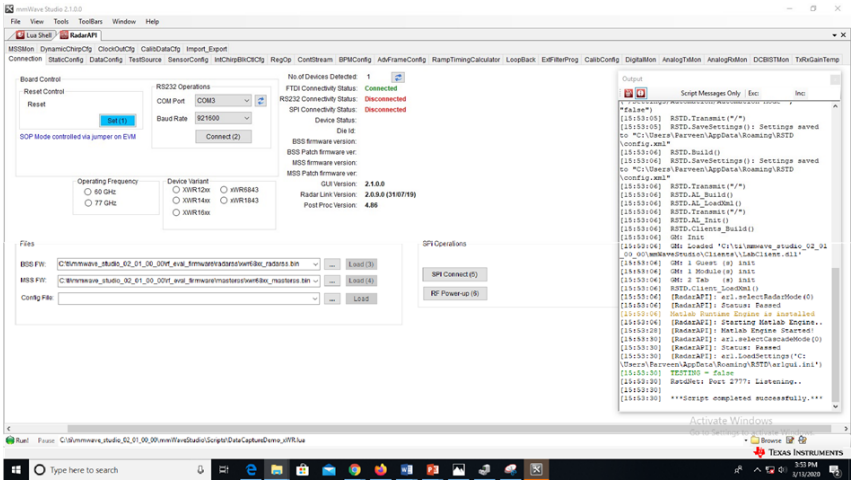


Figure 3.8: Screenshot of mmWaveStudio software

Once buttons "1" to "6" are pressed, the status on MMWAVE Studio should be as shown in Fig. 3.9.

9. Select "Static Config" tab as shown in radar API of MMWAVE studio (refer Fig. 3.10a). This allows us to configure the following sensor parameters:

   a. Basic Configuration

      i. Number of transmitters and receivers to operate.


Figure 3.9: Screenshot upon Successful Connection

ii. ADC configuration such as the number of bits per sample and format.

Press "SET" button for Basic Configuration.

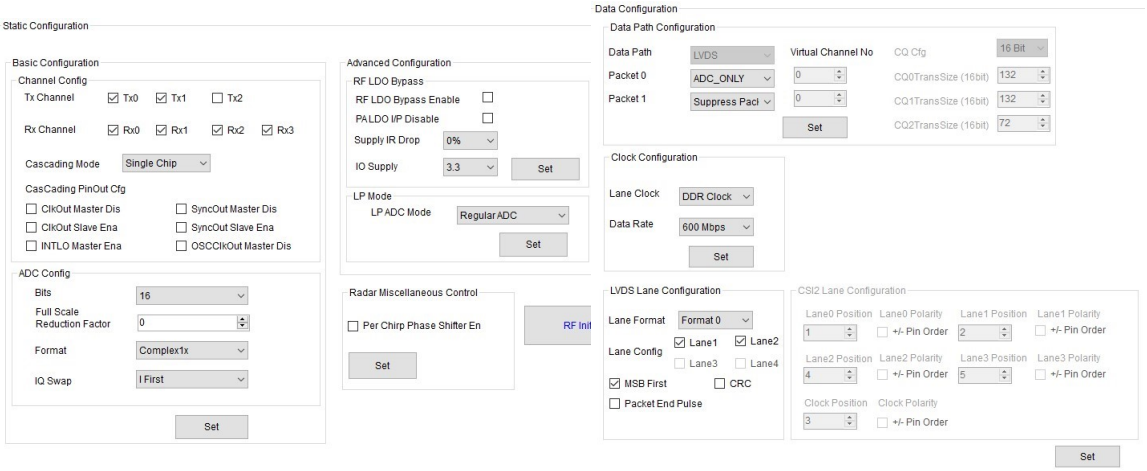
b. Advanced Configuration: Do not change anything here as it may damage the boards.

c. LP Mode

i. Choose Regular ADC option

ii. Press the Set button

d. Then select "RF Init" option.




(a) Static configuration

(b) Data configuration

Figure 3.10: Screenshots of static and data configurations.

10. Now select the "Data Configuration tab" (refer Fig. 3.10b). Here we do not have to change any of the parameters and press the “SET” option.

11. Next select the "Sensor Configuration" which has three subsections (refer Fig. 3.11):

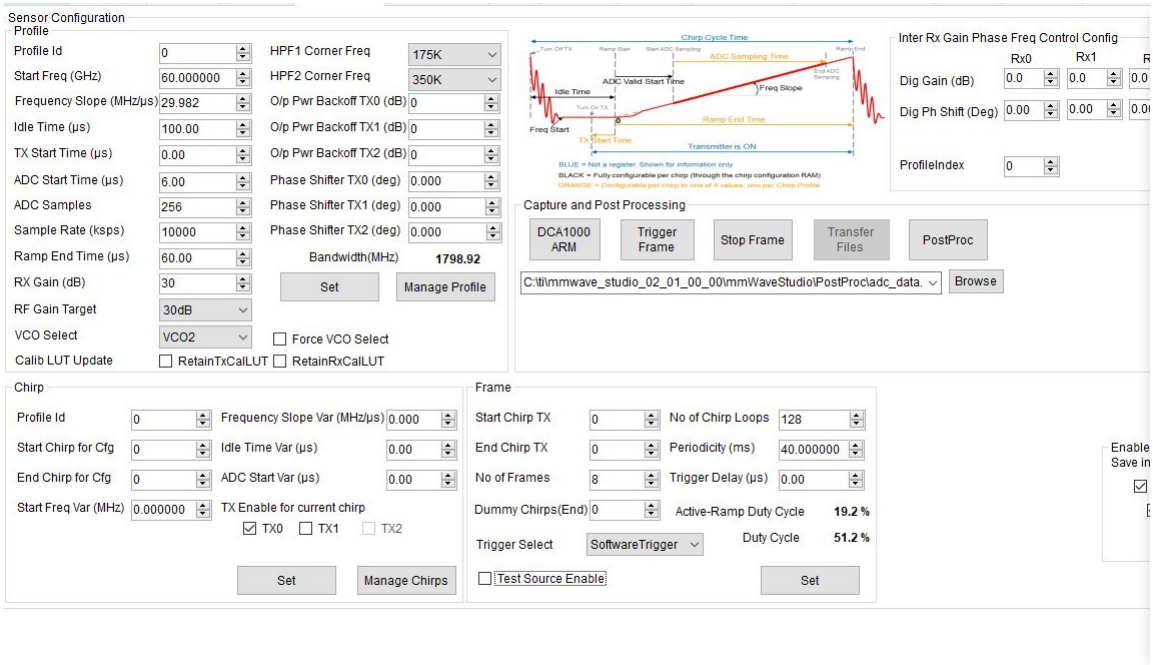



Figure 3.11: Screenshot of the sensor configuration.

i. Profile ID: This subsection allows us to choose radar operating parameters such as the chirp rate, chirp duration, ADC samples per chirp.

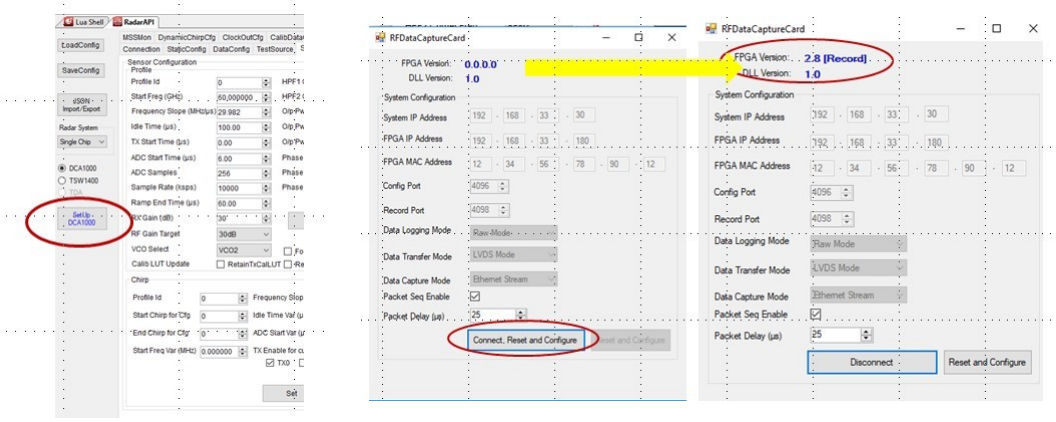
ii. Chirp: Timing parameters related to the ADC start time, idle time per chirp, etc.

iii. Frame: This subsection controls the number of frames to be acquired, frame duration, etc.

After these three subsections have been configured, press their respective “SET” buttons.

12. On the left side of the screen, choose “DCA1000” option and press “Set up DCA1000” (refer Fig. 3.12a)

13. This will open a window as shown in Fig. 3.12b and press “Connect, Reset and Configure”. This will turn FPGA Version: 0.0.0.0 to 2.8 (Record).




(a) Set up DCA1000

(b) RF data capture card

Figure 3.12: Screenshot for setting up the DCA connection.

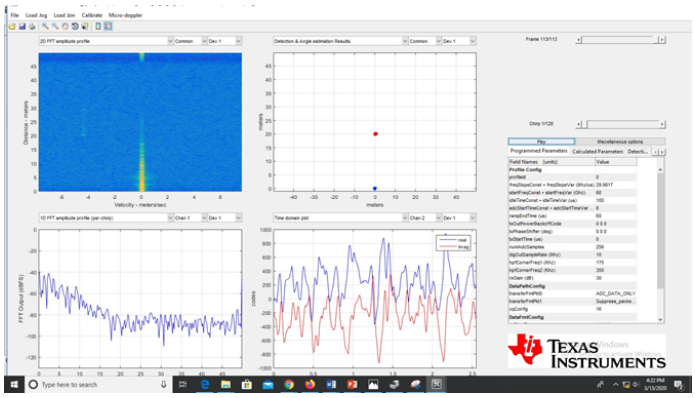



Figure 3.13: Screenshot of post processing of the captured data.

14. Now press DCA1000 ARM and then trigger. "Trigger" is a software control to initiate DCA1000 to start acquiring data. If we have defined a finite number of frames, then it will automatically stop acquiring data, otherwise for infinite duration, we need to press "STOP" button to stop the data acquisition.

15. The acquired data will be stored at location defined under "Capture and Post Processing".

16. MMWAVE Studio has the provision to generate Range–Doppler Profiles via "POST PROC" button. Once pressed, it automatically reads the last acquired binary ADC data file and generates these maps. An example map is shown in Fig. 3.13.

# Chapter 4

# Data Collection and Pre-processing

For the gesture recognition application, we need labeled data and thus we started the radar data collection for a set of gestures. However due to the Covid-19 pandemic we could not complete the data collection. This task was only completed recently. In this chapter, we describe the data collection and the pre-processing steps we have followed.

The first step was to design a set of gestures. During the literature survey we found that researchers use different set of gestures with the number of gestures varying from four to fifteen. Further, the gestures can be performed continuously or only once. We decided to work with non-continuous or single instance gestures. In the absence of our own data set, we started to work with the publicly available data sets from Wang *et al.* [22] and Scherer *et al.* [16] both of which use the same set of 11 gestures. Yu *et al.* [24] uses a set of 11 gestures with some direction dependent gestures (such as clockwise and anti-clockwise circle).

Once we were able to collect our own data, we decided to collect data for 13 gestures from 10 different persons across multiple sessions. The gestures range from fine gestures such as finger rub, pinch to gestures involving large motion such as swipe, push, circle and also include direction dependent gestures. We propose to explore both fine gestures and gestures involving significant motion as these are easy to classify/recognize than the fine gestures.

The gestures were performed in a plane perpendicular to the radar antenna array plane. This results in maximum variance in the RDIs as the perpendicular axis results

in larger changes in the range component as compared to the case when gestures are performed in a plane parallel to antenna. For our data set, each person performed 7 instances of each of the 13 gestures in a session across a total of 5 sessions. Each gesture recording is of a fixed duration given by: number of frames $\times$ periodicity $= 50 \times 35$ ms $= 1.75$ seconds and during this period each user perform a gesture once and then stops. In this manner, each gesture is performed 7 times in each session by each user.

## 4.1 Setup

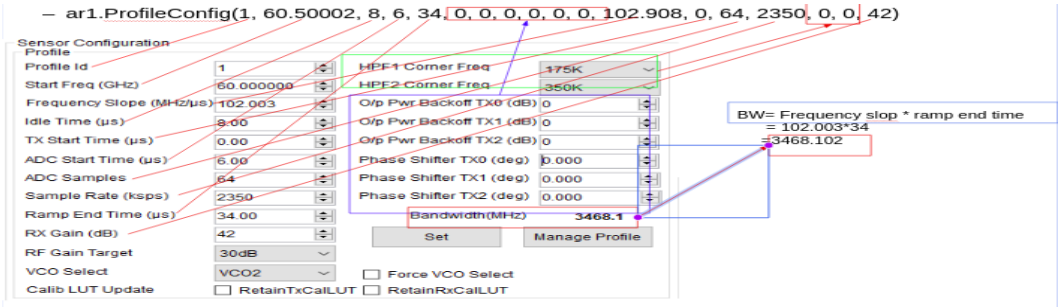
In Chapter 3 we described the hardware setup and software installation. After successful software installation, we perform the following steps each time we connect the radar hardware to the PC for collecting data.

- Load the lua script
- MasterSS (MSS) and RadarSS (RSS) firmware will load automatically to the hardware and other setup will also be done automatically
- Record the gesture

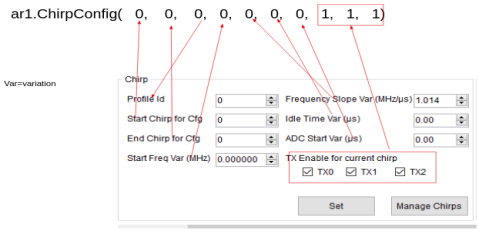
### Lua Script

We need to setup the software and load many of the parameters to hardware each time we start the hardware or software. To automate this setup process we can define these steps in a lua script which will setup the required configuration automatically. These lua commands and their response can be seen in the output window in mmWave Studio when we click on each setup button. The lua setup file which we used for data recordings is given in Appendix A.1

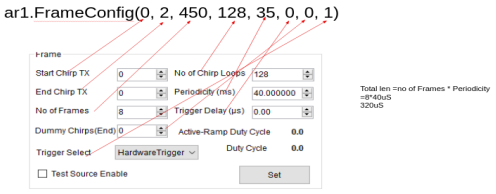
When this lua file is executed, it selects "SOP control" and then connects to the listed COM port. The next two lines load the MSS and RSS firmware to the hardware. The path to these files has to be mentioned in the lua commands. The remaining commands are similar to the hardware setup procedure and can be cross checked with the manual setup by looking at the output tab in mmWave studio. Profile, chirp, and frame configurations are as shown in Fig. 4.1




(a) Profile configuration




(b) Chirp configuration



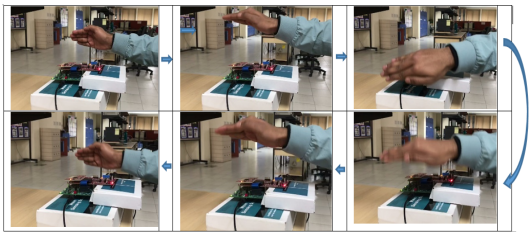

(c) Frame configuration

Figure 4.1: Configurations and the corresponding commands in the Lua script.

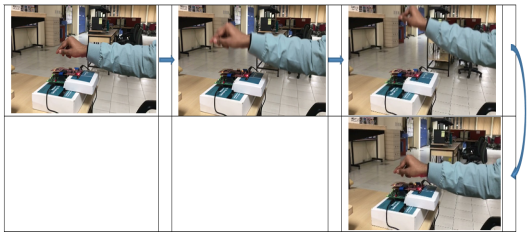
## 4.2 Gesture Set

We designed a set of 13 gestures consisting of fine gestures, normal or wide gestures, and direction dependent gestures. These 13 gestures are shown in Fig. 4.2

**Data Storage Format:** Our data storage format is inspired from the TinyRadarNN data set file format [16]. Each person is named as P1, P2, ... and within each person's folder there are 13 gesture folders. In each gesture folder there are 5 session folders which contains 7 instances of that gesture.

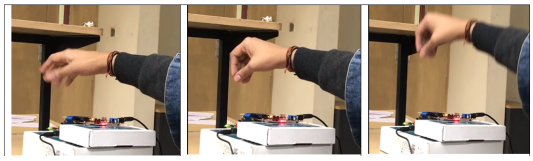


(a) Arc

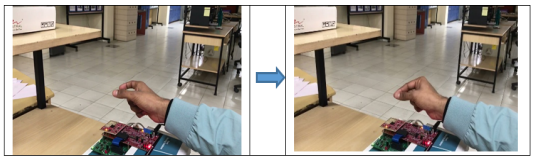


(b) Circle (clockwise)

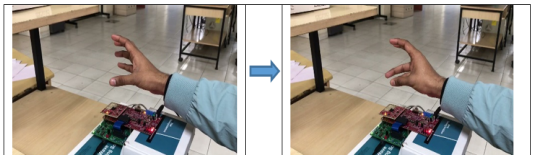
(c) Clap



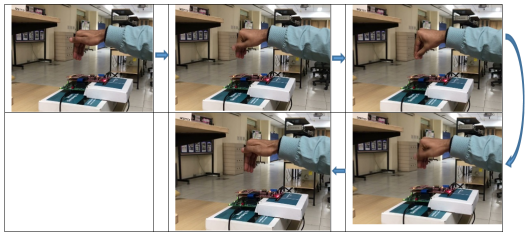
(d) Grab



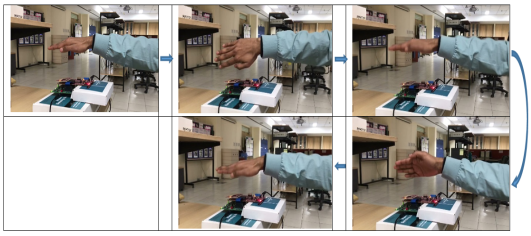
(e) Pinch Index



(f) Pinch Pinky

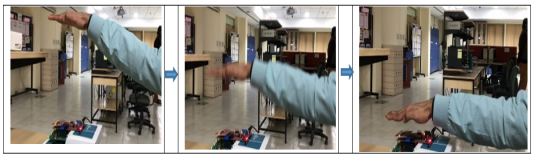


(g) Finger Slide

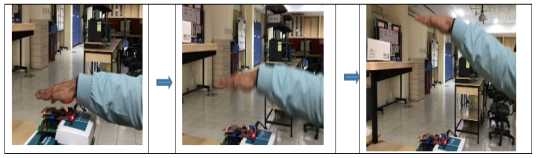


(h) Palm Tilt

(i) Swipe



(j) Push



(k) Pull

Figure 4.2: Gesture Set

## 4.3 Signal Properties

The raw signal captured by the radar receiver is a superposition of reflections from different parts of the hand and other objects in the field of view. This signal is then pre-processed by transforming it to range-Doppler images (RDI). Each pixel value of the RDI represents the reflection intensity from the scattering centers and the pixel position represents the velocity and range. RDI does not provide spatial/skeleton representation of the hand (as we see in camera images) but instead provides high temporal representation of the hand movements. During the time a gesture is performed we obtain a sequence of 2D RDIs (refer Fig. 1.3 C+D).

## 4.4 Data Pre-processing

Raw I/Q data captured by the radar is arranged in a 3D cube commonly known as the radar data cube and shown in Fig. 4.3. We have a total of 12 channels (4 Rx antenna and 3 Tx antenna), 64 range bins, and number of chirps is 128. The raw data stored as a binary file is converted into a data cube, where each data cube correspond to one frame of signal recording. The details of this data conversion are given in the mmWave Studio user guide and our implementation in Python is given in Appendix A.2.

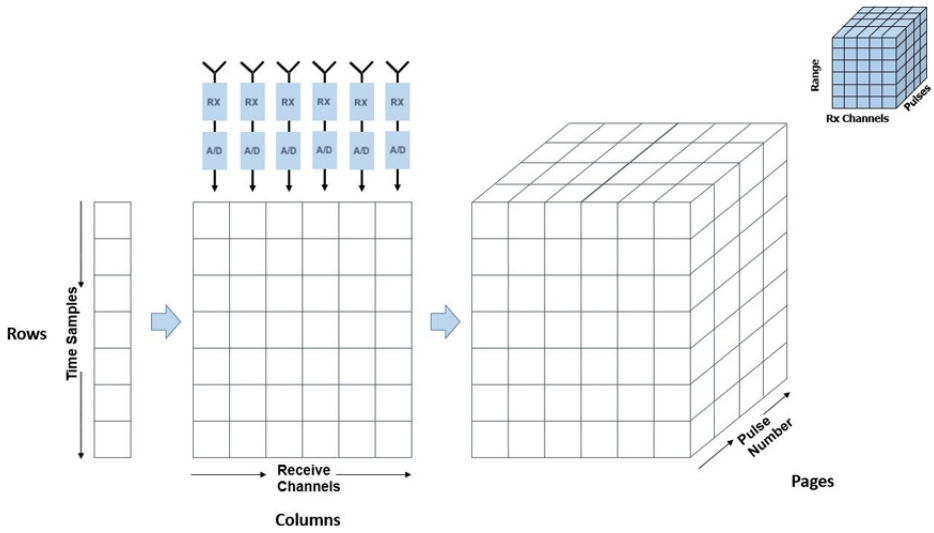



Figure 4.3: Radar Data Cube (Source: Matlab)

After converting the binary data to a sequence of data cubes, we extract range, Doppler, and angle information from each data cube. To obtain range information we perform FFT along time/sampling axis (64 ADC samples per chirp) and performing a second FFT along the pulse/chirp axis gives the RDI. In this manner we obtain a sequence of RDIs from the sequence of data cubes. Angle information is obtained by performing a third FFT along the virtual antenna dimension. The data is now in the form of a cube with 3 axes corresponding to angle, Doppler, and range. We take maximum of the pixel intensities along the Doppler dimension to obtain the range-Angle image (RAI). This complete process is shown in Fig. 4.4.

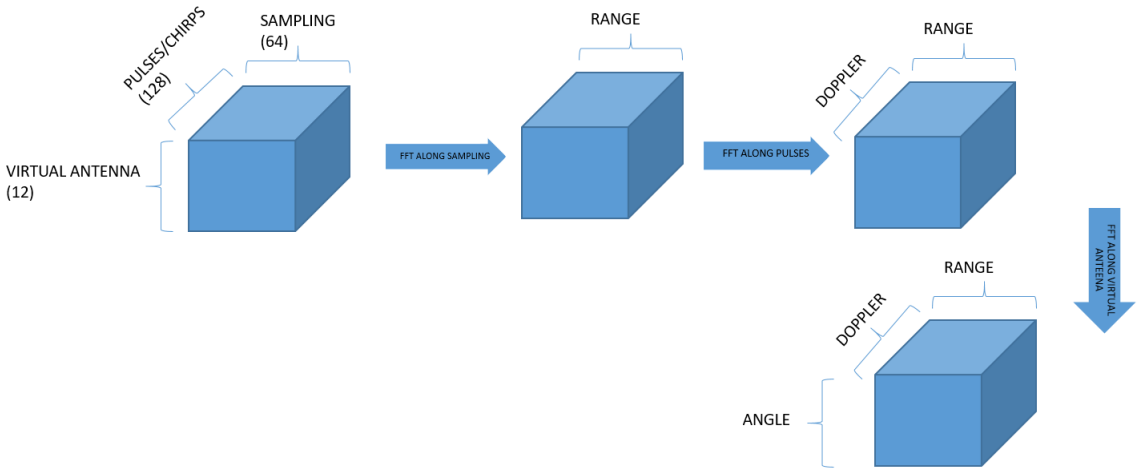



Figure 4.4: RDIs and RAIs from Data Cube

The binary file is read through biraryread.m and before reading the binary file we define few parameters in dataextraction.m which is provided in Appendix A.3.

# Chapter 5

# Gesture Recognition

Our objective is to use the radar data and apply supervised machine learning algorithms to recognize the gestures. We use the labeled time series data set after the pre-processing described in Chapter 4 to train the machine learning models. Each data point is a 3D tensor where the first dimension represents time, second and third dimensions represent the 2D RDI/RAI (refer Fig. 5.1). This data set is similar in structure to a grey-scale video which consists of a (time) sequence of frames of 2D images. The RDI/RAI have very limited spatial resolution but have high temporal resolution. In other words, based a particular RDI/RAI it is difficult to identify the gesture, however with the time sequence of RDIs, the gesture can be recognized. Thus, we will focus on the temporal aspect of the data.

In this chapter, we have used two publicly available data sets Soli [16] and TinyRadar [22]), and a data set provided by Texas Instruments as we were not able to complete our data collection due to the Covid-19 pandemic. Soli and TinyRadar data sets were collected using different radar hardware but with a similar set of gestures which differ only in the sequence length and the resolution of RDI/RAI.

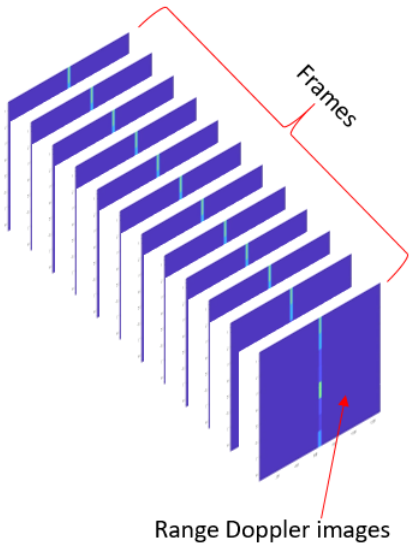



Figure 5.1: RDI time sequence data.

## 5.1 Soli Data Set

The radar data set consists of a time sequence of pre-possessed range-Doppler images. Each sequence is saved as a single HDF5 format data file. Range-Doppler image data array has the dimensions *number of frames* $\times 1024$ (which can be reshaped into 2D range-Doppler image of dimension $32 \times 32$). Each gesture sequence may contain different number of frames as some gestures are of shorter duration than others. There are a total of 5500 such HDF5 files in the Soli data set.

The 11 gestures that are part of the Soli data set are shown in Fig. 5.2. Each column represents a gesture and the snapshots show three key positions for each gesture. The gesture label is indicated by the number in the circle above each column. Sequences with gesture ID 11 are background signals where the hand was not present in the radar field of view.

As radar RDI has very limited spatial information such as the shape of the hand and pose, most techniques that are used for camera-based images and video cannot be used for the radar data. Some of the techniques applied on non-camera based inputs rely on the Doppler effect and have used frame-level classification. Based on this, we will use convolutional neural network (CNN), recurrent neural network (RNN), as well as end-to-end training with CNN and LSTM. In particular, we perform our analysis

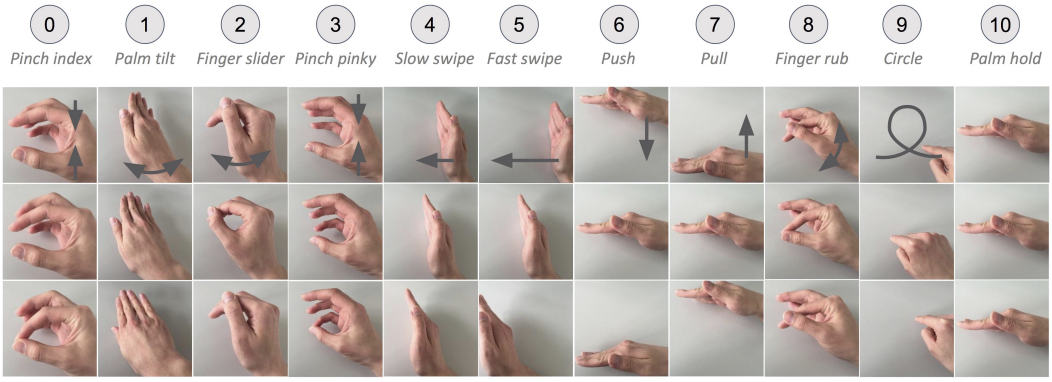



Figure 5.2: Soli data gesture set. (Source [22])

on the following architectures:

- **Frame level classification**: We use standalone CNNs as the baseline model for representation learning. We would like to understand how frame-level classification works and we will compare the performance with and without pooling.
- **Standalone RNN**: Since the radar data has high temporal resolution, we will use RNN with the sequence of RDIs as the input.
- **CNN and LSTM**: CNN and LSTM are jointly trained in an end-to-end manner with the sequential input data.
- **Range-time and Doppler-time from RDI**: We generated range-time and Doppler-time data from the RDI sequences.

### 5.1.1 Frame level Classification

Frame level classification uses each frame of the gesture across the training set as an independent input for the training model. Three variants of the CNN architecture are shown in Table 5.1. The first two architectures represent shallow CNNs with and without pooling layer, and the third is a deep CNN network as given in [22].

**Implementation:** The training, validation and test data consists of 1250, 625, and 625 gesture instances, respectively. We use the Adam Optimizer with a learning rate of 0.001 and categorical cross-entropy as the loss function. We use Tensor-Flow callback to save the best model by monitoring the validation loss.

| Layer | Shallow CNN with pool | Shallow CNN without pool | Deep CNN |
|---|---|---|---|
| 1 | conv1 $3 \times 3 \times 32$ - pool1 $2 \times 2$ | conv1 $3 \times 3 \times 32$ | conv1 $7 \times 7 \times 96$ - pool1 $3 \times 3$ |
| 2 | conv2 $3 \times 3 \times 64$ - pool2 $2 \times 2$ | conv1 $3 \times 3 \times 64$ | conv2 $5 \times 5 \times 256$ - pool2 $3 \times 3$ |
| 3 | conv3 $3 \times 3 \times 128$ - pool3 $2 \times 2$ | conv1 $3 \times 3 \times 128$ | conv3 $3 \times 3 \times 512$ |
| 4 | fc4 512 | fc4 512 | conv4 $3 \times 3 \times 512$ |
| 5 | fc5 512 | fc5 512 | conv5 $3 \times 3 \times 512$ - pool5 $3 \times 3$ |
| 6 | fc6 - softmax 11 | fc6 - softmax 11 | fc6 4096 |
| 7 | | | fc7 2048 |
| 8 | | | fc8 - softmax 11 |

Table 5.1: Frame level classification architectures.

| Model | Accuracy |
|---|---|
| CNN with pooling | 55.4 (91.3*) |
| CNN without pooling | 57.7 (87.97*) |
| Deep CNN | 53.8 (92.8*) |

Table 5.2: Accuracy on frame level classification (* after mode operation).

**Evaluation**: To evaluate the models we first classify each frame which yields an accuracy in the range of 55%. During testing, we notice that if a gesture set contains all the frames of a particular gesture (rather than the frame being shuffled across different gestures) many frames are classified correctly. Further analysis reveals that the mode of the frame sequence corresponding to a particular gesture represents the true class of gesture. For example, if gesture A has label X and contains 50 frames, and assuming 28 frames are classified as X, 10 frames as Y, 5 frames as Z, and so on. Then we classify gesture A as X. This "trick" significantly improved the classification accuracy to 93% whereas standard frame level classification has an accuracy of $55 \pm 2\%$. The results are presented in Table 5.2.

### 5.1.2 Sequence Model

To fully utilize the high temporal resolution provided by the radar, we use two models which accept frame sequences as input rather than only individual frames. The first is a recurrent neural network (RNN) and the second is a 3D CNN (refer Table 5.3). We use long short term memory (LSTM) cells and trained them on time sequence data. Similarly, we trained the 3D CNN model.

**Implementation detail**: We use 1386 gesture instances for training and 1364

| Layer | RNN | 3D CNN |
|---|---|---|
| 1 | fc1 512 | Conv1 3x3x3x32-pool 2x2x2 |
| 2 | Istm 512 | Conv2 3x3x3x64 - pool 2x2x2 |
| 3 | fc3-softmax 11 | Fc1 256 |
| | | Fc2-softmax-12 |

Table 5.3: Sequence model architecture

instances for testing. We use the Adam optimizer with a learning rate of 0.002 and categorical cross-entropy as the loss function. Since each gesture is of different duration, it will consist of different number of frames. However in preparing the time sequence data as input for LSTM, we provide a fixed window of 100 frames and pad it with zero frames if the sequence contains less than 100 frames and the frame sequence will be clipped if the frame sequence length is greater than 100. The data is reshaped to $32 \times 32$ for being fed into the CNN model.

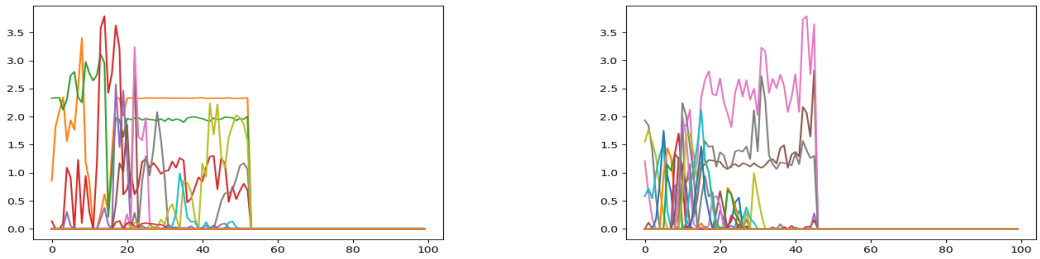
These sequence models provide good results with accuracy of 97.4% using LSTM and 97% using a 3D CNN model (refer Appendix B.1 and B.3 for the confusion matrices using LSTM and 3D CNN models). It is seen that the two gestures pinch index (tapping thumb and index finger) and pinch pinky (tapping thumb and little finger) show large mis-classification with classification accuracy in the range of 90-92%. Gestures such as palm tilt, finger rub, finger slide, push, pull, circle, fast swipe, slow swipe have accuracy in the range of 99-100%.

### 5.1.3 CNN and LSTM

We trained CNN and LSTM jointly in an end-to-end manner. The first part of the model is CNN which is used for feature extraction and the second part is LSTM which uses the temporal resolution of the gestures. These two models are trained jointly (rather than individually) and error is back-propagated through the entire network. Model architecture is shown in Table 5.4. We use the Adam optimizer with a learning rate of 0.0001 and categorical cross-entropy as the loss function. Using this model we achieve an accuracy of 97.3%. By analyzing the confusion matrix (Fig. B.2), we see that the model performance is similar to 3D CNN and LSTM, and is able to classify the gestures with high accuracy except the pinch index and pinch pinky gestures.

| Layer | CNN and LSTM model |
|---|---|
| 1 | cov1 3x3x32 |
| 2 | cov2 3x3x64 |
| 3 | cov3 3x3x128 |
| 4 | fc4 512 |
| 5 | fc5 512 |
| 6 | lstm 512 |
| 7 | fc7-softmax 12 |

Table 5.4: CNN and LSTM model.



(a) Range-time plot

(b) Doppler-time plot

Figure 5.3: Range-time and Doppler-time plots.

### 5.1.4 Range-time and Doppler-time data from RDI

RDI is a 2D plot of range and velocity, and CNN extracts features from the RDI. We can instead use hand-crafted features and eliminate the 2D-CNN so that only sequence models such as RNN and TCN can learn the relevant classification model. We extract range-time and Doppler-time data (refer Fig. 5.3) from the sequence of RDIs (the flat region is due to padding). A similar approach has been used by [3, 19, 23].

$$RT_{time} = \sum_{velocity} RDI_{time}(range, velocity) \tag{5.1}$$

$$DT_{time} = \sum_{range} RDI_{time}(range, velocity) \tag{5.2}$$

Since we have four receiver channels we obtain four range-time and four Doppler-time data sequences which are then concatenated together.

We use a single layer of LSTM with 256 inputs and 128 outputs followed by a fully

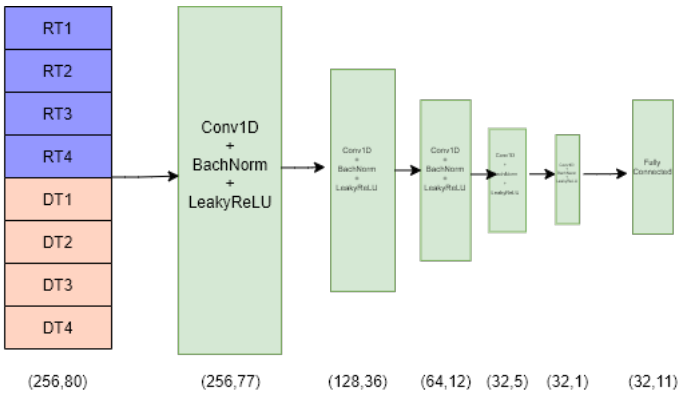



Figure 5.4: 1D Convolution

connected layer as shown in Fig. 5.5. The model contains 331,147 parameters and achieves 96.8% accuracy. We then use 1D CNN with 441,195 parameters as shown in Fig. 5.4. For this architecture, we use a 2D tensor as input where the range-time and Doppler-time of size 256 ($(32 + 32) \times 8$) are stacked as a function of time. This input tensor is passed to a dilated 1D-CNN (dilation captures long temporal dependencies). The full model architecture is given in Appendix B.1. This model achieves an accuracy of 98.2%. The classification results and confusion matrix are shown in Table 5.5 and Fig. 5.6. It is seen that for several gestures such as palm tilt, slow swipe, fast swipe, push, pull, finger rub, and circle, the model achieves perfect classification. From the confusion matrix, it is seen that pinch index and pinch pinky gestures do not achieve good performance which is consistent with the results of the other models. 1D CNN achieves the best classification and surpasses the state of the art models in [16, 22]. Apart from the improvement in accuracy, the model does not contain LSTM layer. LSTM cannot utilize parallelism as each cell's output acts as the input for the next cell. On the other hand, CNN can be computed in parallel and this can provide fast inference results. In addition, the proposed model requires 441,195 parameters as compared to 46,763,532 parameters in Soli's model [22] which is over two orders of magnitude reduction in the number of parameters (refer Table 5.6).

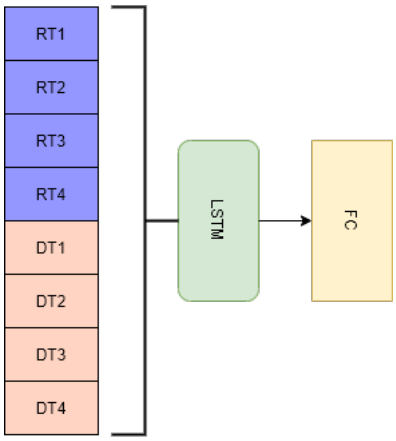



Figure 5.5: LSTM network

| **Gesture** | **Precision** | **Recall** | **F1-Score** |
|---|---|---|---|
| pinch index | 0.915 | 0.960 | 0.937 |
| palm tilt | 1.000 | 1.000 | 1.000 |
| finger slide | 0.976 | 0.976 | 0.976 |
| pinch pinky | 0.941 | 0.903 | 0.922 |
| slow swip | 1.000 | 0.984 | 0.992 |
| fast swip | 1.000 | 1.000 | 1.000 |
| push | 1.000 | 1.000 | 1.000 |
| pull | 1.000 | 1.000 | 1.000 |
| finger rub | 1.000 | 0.992 | 0.996 |
| circle | 1.000 | 0.992 | 0.996 |
| hold | 0.976 | 1.000 | 0.988 |
| **Overall accuracy** | 0.982 | | |

Table 5.5: Classification results using 1D CNN.

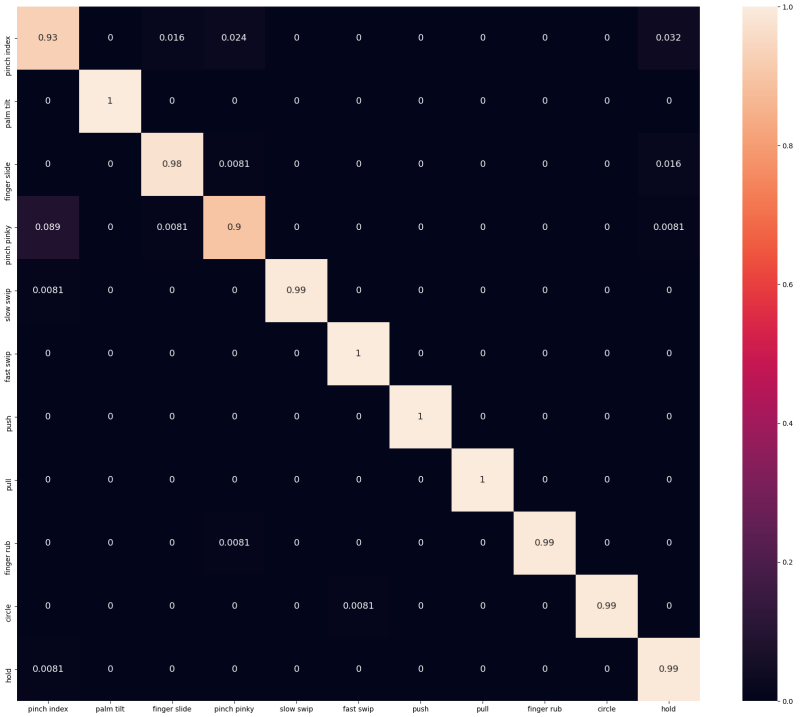



Figure 5.6: Confusion matrix based on the 1D CNN results.

### 5.1.5 Model Comparison

The models we have considered such as the CNN+LSTM (end-to-end), LSTM, 3D CNN, and 1D CNN give similar classification accuracy (97-98%). We also observe that two of the gestures, pinch pinky and pinch index, are not recognized accurately resulting in poor classification performance. This is because pinch pinky and pinch index gestures are quite similar. If these two gestures are removed from the gesture set, an overall accuracy of 99.5% is achieved with 1D CNN. The models along with their parameter size and classification accuracy are listed in Table 5.6. 1D CNN has the fewest number of parameters and can utilize parallel computation which makes it ideal for resource constrained mobile devices.

| Model | Parameters | Accuracy |
|---|---|---|
| CNN+ LSTM | 46,763,532 | 97.3% |
| 3D CNN | 13,625,548 | 97.4% |
| LSTM | 2,629,643 | 97% |
| 1D CNN | 441,195 | 98% |

Table 5.6: Model comparison

## 5.2 TinyRadarNN Data Set

The TinyRadarNN [16] data set is a collection of radar data from hand gestures, collected at the Integrated Systems Lab at ETH Zurich using the Acconeer XR111 sensor. The data sets are divided into one data set with 5 gesture classes and a second data set with 11 gesture classes. For our analysis, we use the 11 gesture classes data set. Each gesture instance is captured as 5 frames with 32 velocity bins and 492 range bins. The Soli data set, we had 32 range and 32 velocity bins with frame lengths varying from 40 to 100. Thus, the TinyRadar data set has high resolution along the range axis compared to velocity. The Soli data set was collected using hardware with 4 channels while TinyRadar hardware has 2 channels. The TinyRadar data set is in raw binary format which we process to generate RDIs/RAIs. The following sections describe the various methods and algorithms used.

### 5.2.1 CNN and TCN

We use the temporal convolutional network (TCN) as a sequence model and the model architecture is shown in Fig. 5.7. The model contains 64,000 parameters. We divide the data set into training set (2/3), test set (1/3), and validation set (1/5 of training). The input dimension is $5 \times 32 \times 492 \times 2$ where the first dimension is the frame length (5), second and third dimensions are the RDI dimensions, and the last dimension is the number of channels or receive antenna (2).

The TCN model achieves an accuracy of 85.5% and the confusion matrix is shown in Fig. 5.8. We observe that fine gestures such as pinch index, pinch pinky, finger slide, and finger rub have significant misclassifications between them compared to the other gestures. Thus, these fine gestures appear "hard" to classify though similar fine gestures

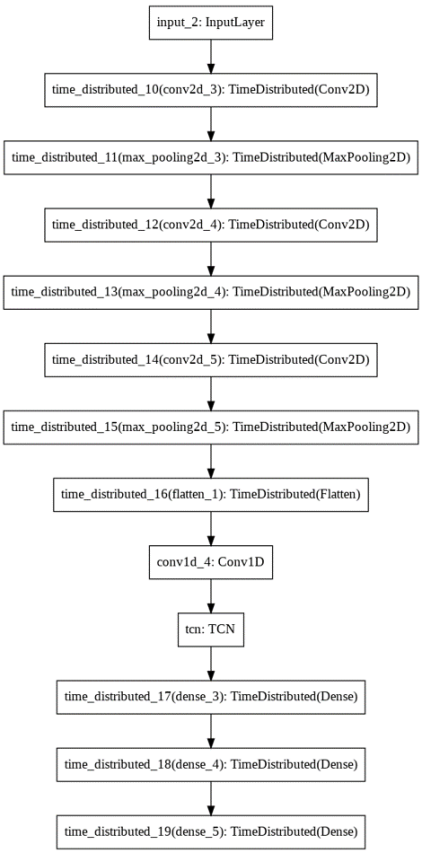



Figure 5.7: TCN model architecture.

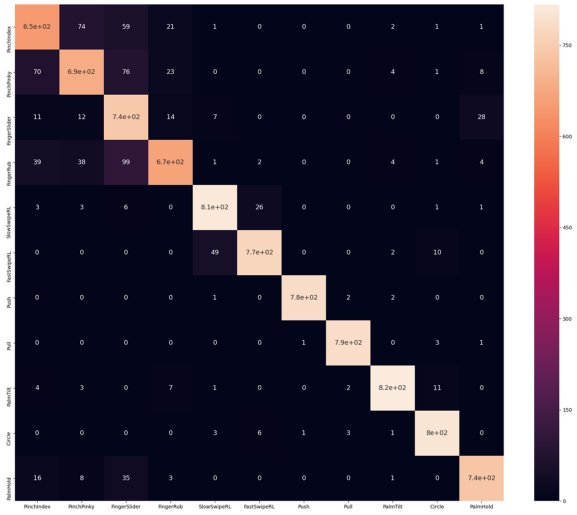



Figure 5.8: Confusion matrix for the results using the TCN model.

in the Soli data set were classified much more accurately. This is due to the larger number of frames per gesture instance in the Soli data set compare to the TinyRadar data set. We can reduce the number of model parameters without compromising with performance by using the reduced TCN model [16] shown in Fig. 5.9.

### Single and Multiple Users

We have observed that if we train the model on a single user's data and test it on the same user's data, about 97% classification accuracy can be achieved. However when we use multiple user's data during training and testing, the classification accuracy achieved is only 85%. Single user model performs well on both large motion gestures as well as fine gestures. In the case of model trained on multiple users, most of the misclassifications are due to the fine gestures. This is due to variation in the data when different people perform the same gestures. For example, while performing the swipe gesture some people use only the palm while other users use the entire hand. The confusion matrix is shown in Fig. **??**. We would like to achieve single user's accuracy and precision on the multiple user's data set.

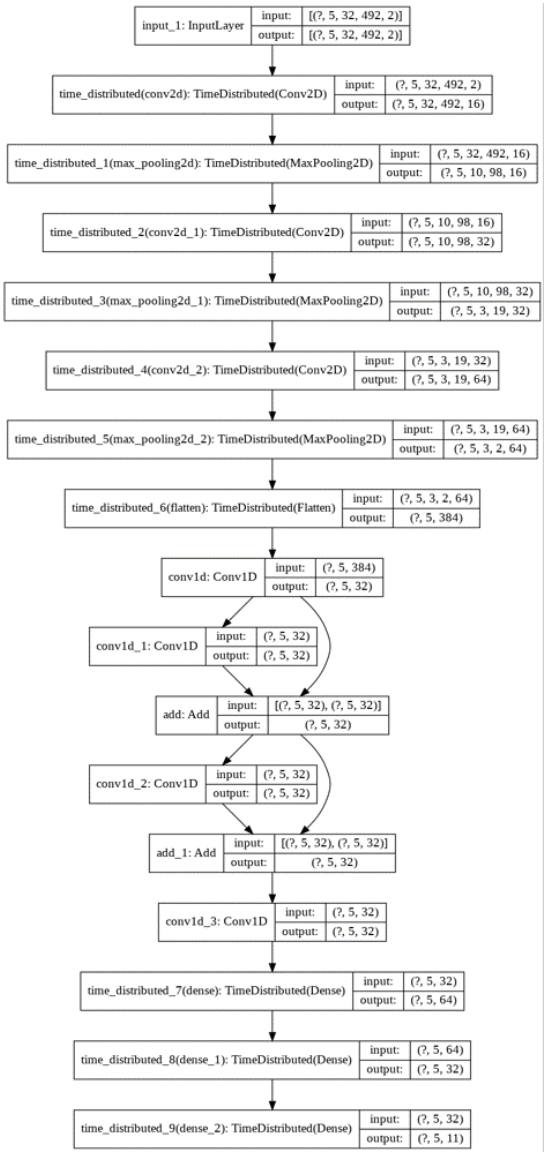



Figure 5.9: Reduced TCN model architecture.

| Fold | Accuracy on entire data set | Accuracy on multiple user data set |
|---|---|---|
| 1 | 90.71 | 87.77 |
| 2 | 91.19 | 87.06 |
| 3 | 91.06 | 86.82 |
| 4 | 91.17 | 85.28 |
| 5 | 90.20 | 87.02 |
| Average | 90.86 | 86.79 |

Table 5.7: Results of five-fold cross validation.

### Five-Fold Cross Validation

This is a technique for assessing how the results of a statistical analysis will generalize to an independent data set. The Soli data set contains a larger single user data set to test the capability of the single user model. The results of five-fold cross validation are shown in Table 5.7 where two accuracy figures are listed. One is when we use the single user and multiple user data sets together, and the second one is using only the multiple user data set. As mentioned previously the single user data set gives much better results than the multiple user data set, and if the data of single user and multiple users are combined in equal proportion we observe an increase of 4% in the classification accuracy, which is the average of single user and multiple user accuracy.

### Leave One Out Cross Validation

Next, we train the model on the entire data set leaving out one user's data and using this user's data as the test data. This indicates how the model would perform when it encounters data from a user which was not contained in the training set. In this analysis, an average accuracy of 71% was achieved for 22 users. Table 5.8 shows the leave one out cross validation (LOOCV) results (Accuracy 1 and Accuracy 2 refers to two independent runs). We observe high variance in the accuracy across users. LOOCV accuracy when tested with user 8 which was excluded during training gives only 58% accuracy while user 19 gives 82% accuracy. As we have seen in the case of single user versus multiple users, the accuracy of the model is highly dependent on the choice of data for training and testing. We have seen that if only the single user data set is used in training and testing, 97% accuracy is achieved, whereas with multiple users data

| User | Accuracy 1 | Accuracy 2 |
|---|---|---|
| 1 | 63.33 | 63.33 |
| 2 | 67.73 | 67.58 |
| 3 | 77.90 | 70.84 |
| 4 | 72.55 | 70.70 |
| 5 | 72.97 | 72.34 |
| 6 | 73.99 | 68.28 |
| 7 | 75.18 | 72.32 |
| 8 | 57.64 | 63.01 |
| 9 | 76.79 | 73.57 |
| 10 | 66.34 | 64.89 |
| 11 | 71.91 | 68.19 |
| 12 | 72.93 | 73.26 |
| 13 | 63.96 | 64.95 |
| 14 | 66.86 | 67.42 |
| 15 | 70.55 | 66.76 |
| 16 | 75.92 | 75.66 |
| 17 | 66.50 | 71.05 |
| 18 | 72.57 | 73.77 |
| 19 | 81.68 | 81.35 |
| 20 | 64.58 | 63.89 |
| 21 | 78.38 | 75.77 |
| 22 | 76.50 | 73.09 |
| **Average** | **71.22** | **70.09** |

Table 5.8: Results of the leave one out cross validation.

shuffled between training and testing, an accuracy of 85% is obtained, and if both the single user and multiple user data sets are combined in equal proportion we obtain an accuracy of 91% which is the average of the single user and multiple users. This analysis give a measure of robustness of the model.

### 5.2.2 3D CNN Model

We use the 3D convolutional neural network model (total parameters: 2,450,151) with a dropout of 0.2 and 0.5 as shown in Fig. 5.10. Without a dropout layer the model achieves an accuracy of 90.6% and with the dropout layer accuracy of 92.2% (dropout=0.2) and 93.78% (dropout=0.5) are achieved. Upon analyzing the confusion matrix (shown in Fig. B.6), we observe that there are 4 gestures (pinch index, pinch

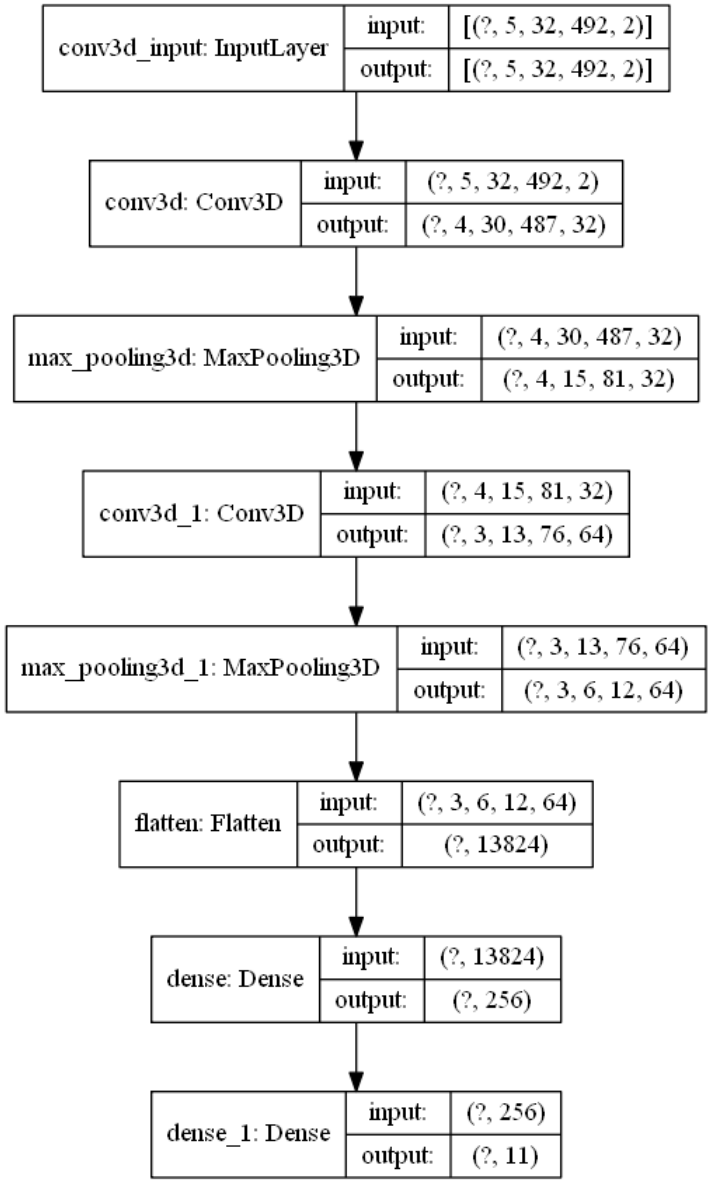



Figure 5.10: 3D CNN model architecture.

pinky, finger rub, finger slide) which have poor performance in this model as well in the earlier CNN+TCN model. In the Soli data set, reasonably good classification accuracy was achieved for these 4 gestures as compared to that in the TinyRadar data set. One possible reason is the larger number of frames (40 to 100) per gesture instance in the Soli data set as compared to only 5 in the TinyRadar data set. Due to the smaller number of frames, TinyRadar data set is not able to record the fine gestures with good temporal resolution.

Next we eliminate the above mentioned 4 gestures from the gesture set and use only the remaining 7 gestures. By training the model on this reduced set of gestures results in an average accuracy of 98.56% and the confusion matrix is shown in Fig. B.7. We then perform LOOCV analysis and the results are shown in Table B.2 for 11, 7, and 4 gestures. The 3D CNN model results in an average classification accuracy of 78% which is 7% more than that using the CNN+TCN model (based on 11 gestures). However, the CNN+TCN model uses 38 times fewer parameters than the 3D CNN model. In LOOCV based on 11 gestures, there is a 15.5% reduction in average accuracy (93.78% to 78.28%), while based on 7 gestures the reduction is 5.5% (98% to 92.5%) which indicates that these 7 gestures will perform well across users.

We will restrict to using only 7 gestures due to consistently lower classification accuracy for the remaining 4 gestures. Upon analyzing the LOOCV for different users on 7 gestures, we note that most of the misclassifications are between slow swipe and fast swipe with the slow swipe being misclassified as fast swipe. This is due to the fact that slow and fast are relative terms and in many cases a user's "slow" can be similar to another user's "fast" swipe. We note that user_10 and user_17 have the least classification accuracy while user_19 has the best generalization accuracy. We also note that for user_10, 90% of the time slow swipe is misclassified as fast swipe while for user_17, 88% of the time fast swipe is misclassified as slow swipe. For detailed results refer to Table B.2 and Fig. B.8.

### 5.2.3 Raw I and Q Data

The RDIs are obtained by applying FFT twice to the raw complex I and Q data. Next we use the raw I and Q data directly for gesture recognition. The TinyRadarNN data

Table 5.9: Single user

| User | Accuracy using RDI | Accuracy using I\Q data (Magnitude values) | Accuracy using I\Q data (Two channels) |
|---|---|---|---|
| 0 | 96.93 | 95.11 | 97.00 |

Table 5.10: Five fold cross validation

| Fold | Accuracy using RDI | Accuracy using I\Q data (Magnitude values) | Accuracy using I\Q data (Two channels) |
|---|---|---|---|
| 1 | 87.77 | 85.00 | 86.50 |
| 2 | 87.00 | 85.20 | 86.20 |
| 3 | 86.82 | 83.00 | 86.20 |
| 4 | 85.82 | 83.48 | 86.50 |
| 5 | 87.00 | 83.00 | 84.00 |
| Average | 87.08 | 83.93 | 85.88 |

set provides the raw data and since this data is complex, we use the magnitude values. In addition, we consider the real and imaginary parts as two channels and perform analysis on this data. The results are shown in Table 5.9) for a single user, Table 5.10 for 5-fold cross-validation, and Table 5.11 for LOOCV analysis. We use the CNN+TCN model as described earlier. We note that that using the raw data doesn't give any significant advantage and results in similar average classification accuracy of 86% on the multiple user data set. Thus, we can remove the pre-processing steps without compromising on the accuracy.

Table 5.11: Leave one out cross validation

| User | Accuracy using RDI | Accuracy using I\Q data (Magnitude values) | Accuracy using I\Q data (Two channels) |
|---|---|---|---|
| 1 | 63.33 | 69.44 | 64.46 |
| 2 | 67.73 | 66.49 | 71.77 |
| 3 | 77.90 | 74.13 | 68.20 |
| 4 | 72.55 | 75.06 | 68.95 |
| 5 | 72.97 | 66.78 | 66.17 |
| 6 | 73.99 | 66.89 | 64.49 |
| 7 | 75.18 | 72.99 | 71.41 |

### 5.2.4 Range Angle Images

Using the TinyRadar raw data, we generate range angle images (RAIs) as described in Chapter 4. Similar to the RDIs, we can use RAIs for gesture recognition. A similar approach has been used by Yu *et al.* [24]. We train the CNN+LSTM (refer Fig. 5.11) and CNN+TCN (refer Fig. 5.12) models with RDIs and RAIs separately as well as both combined. The model architectures are inspired from image caption technique where

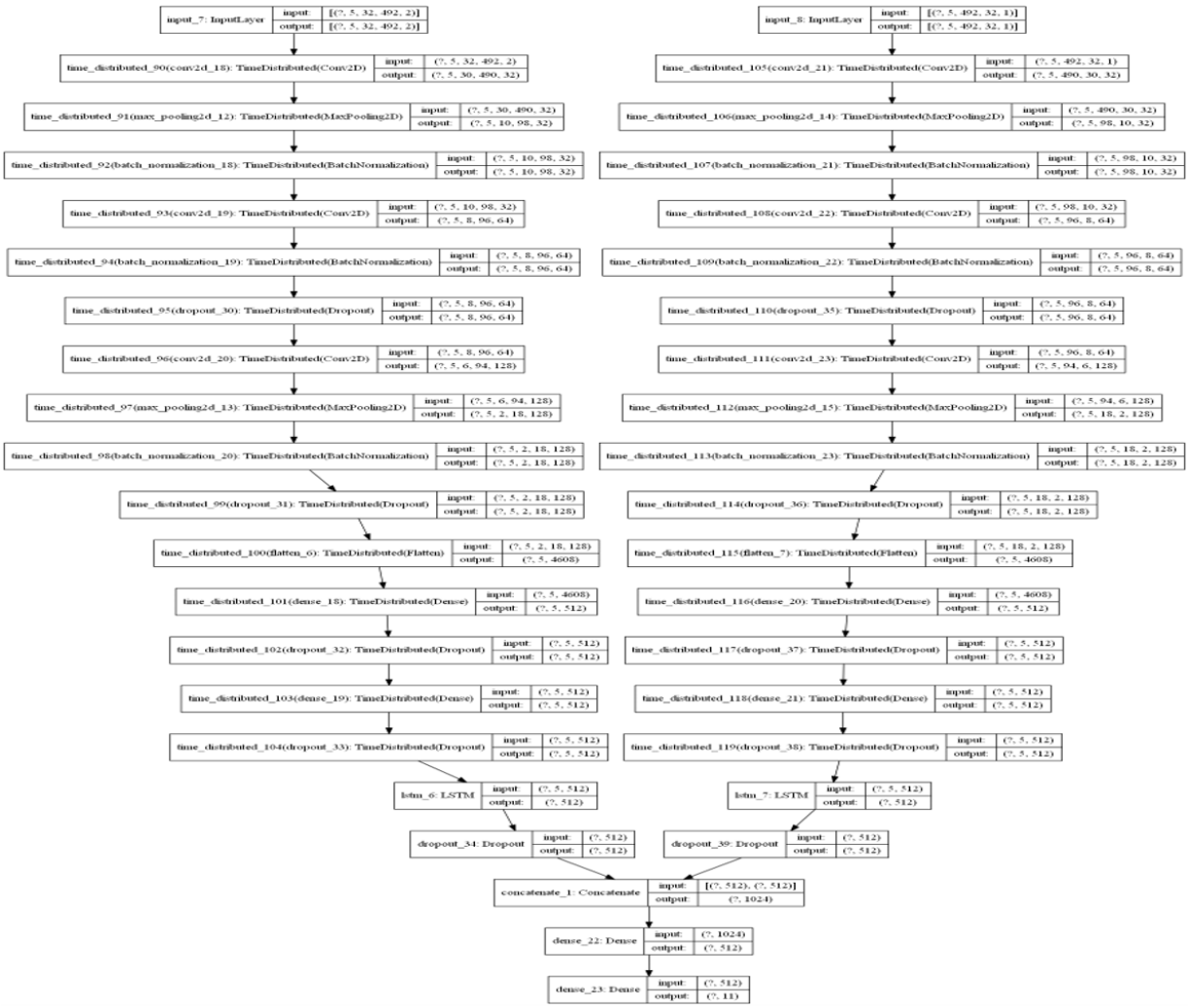



Figure 5.11: CNN+LSTM model architecture.

image is feed in one input channel and the caption is given to the other input channel. Features from both the input channels are combined and pass through a simple fully connected layer. In our case, the architecture is simpler as both the input channels take similar type of input data.

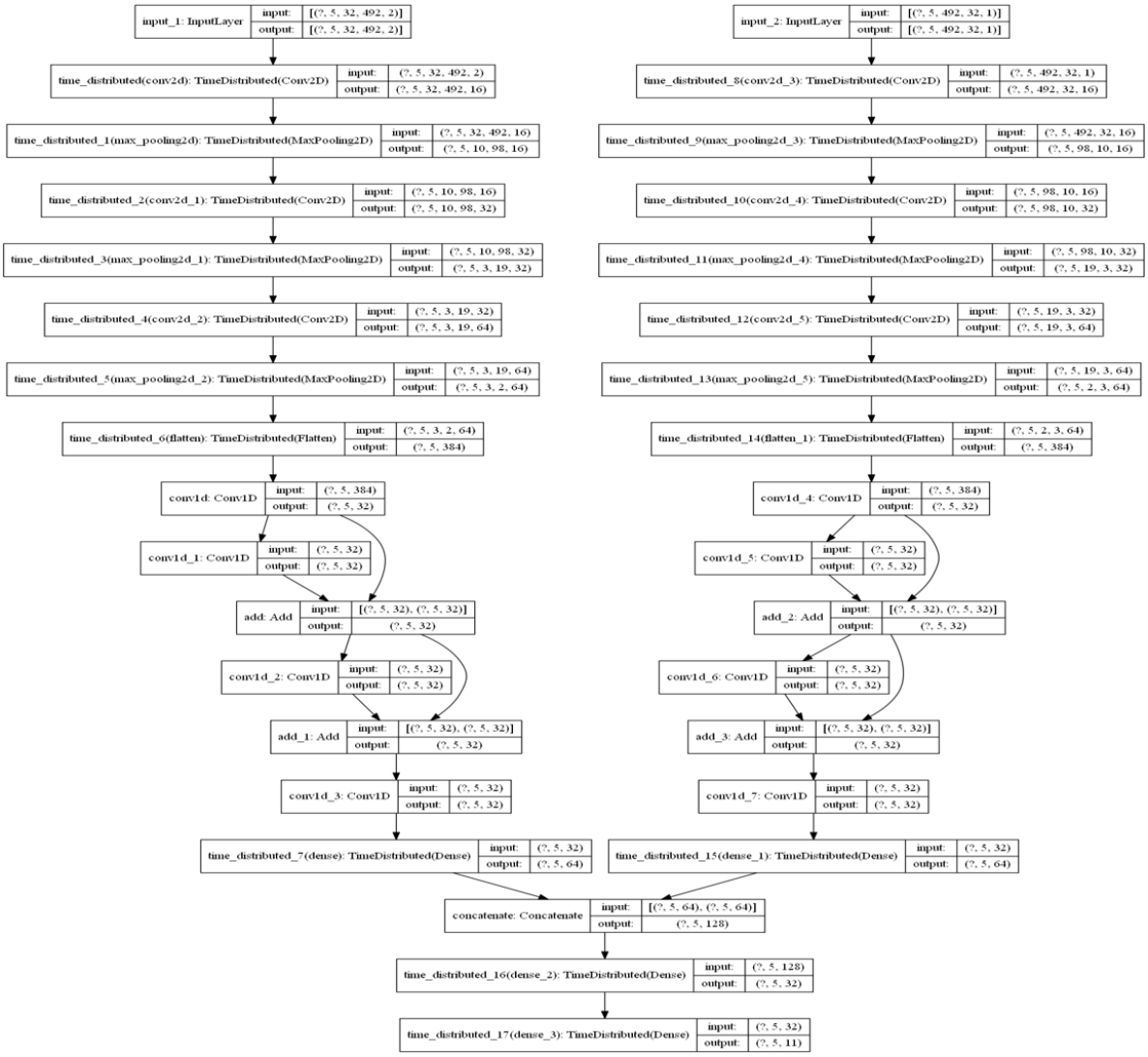



Figure 5.12: CNN+TCN model architecture.

Thus, we remove the second input channel and train using only one input channel with the RDIs as input. Similarly, we remove the first input channel and train using only the second channel with the RAIs as input. Finally, we use both the input channels with RDIs and RAIs and concatenate their outputs, which is then passed through a fully connected layer. The results are summarised in Table 5.12.

We do not observe any significant improvement in performance using RAIs. It is noted that RAIs are only useful in recognising direction-dependent gestures such as clockwise circle and anticlockwise circle. For direction-dependent gestures such as clockwise and anticlockwise circle the range-Doppler profiles are very similar whereas

| CNN+LSTM model | | | CNN+TCN model | | |
|---|---|---|---|---|---|
| RDI | RAI | RDI+RAI | RDI | RAI | RDI+RAI |
| 72.85% | 55% | 71.50% | 82.65% | 72.00% | 84.8% |

Table 5.12: Results of CNN+LSTM and CNN+TCN models using RDI, RAI, and both combined.

their range-angle profiles are very different. Since the TinyRadar data set does not contain directional gestures we were not able to get any benefit from using RAIs. Thus, in designing a gesture set it is good to include directional gestures to use the additional information provided by multiple receive antenna and the RAIs.

### 5.2.5 CNN and LSTM

Similar to the end-to-end training used with the Soli data set, we train CNN+LSTM for the TinyRadar data set. Similar classification accuracy of 98.6% is achieved using 3D CNN model with the RDIs. We also carry out LOOCV analysis and obtain similar results which are shown in Table B.3.

## 5.3 Texas Instruments Data Set

A third data set was provided by Texas Instruments, India for research purposes. This data set consists of continuous gestures which are different from the non-continuous gestures in the other data sets. An example of a continuous gesture is drawing circle continuously. Thus, in this data set, the gestures do not have a start or end point. This introduces new challenges in gesture recognition and thus we explore the window length (or the number of frames) used while making inference. The data set contains 4 gestures twirl, fine-tuning, light-up and come (refer Fig. 5.13).

### 5.3.1 Data Set Description

This data set consists of two files x_training.csv and y_training.csv.

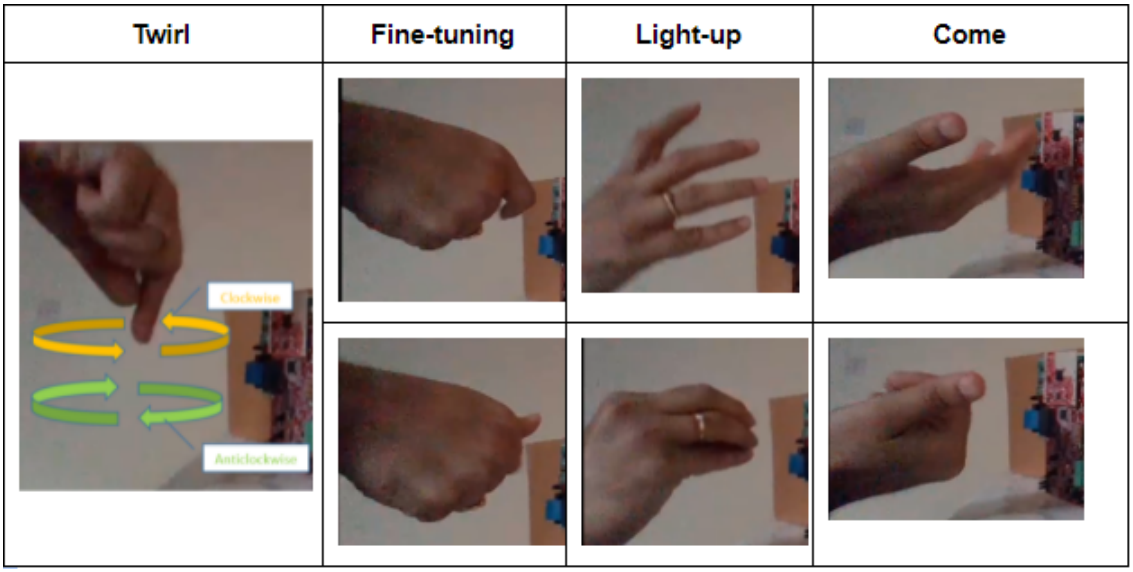



Figure 5.13: Gesture data set provided by Texas Instruments.

- Each range-Doppler image is read out column-wise (column1→ column2→ etc.) to create a vector of length 8×128 = 1024.
- Each row of the file x_training.csv contains one such image.
- The frame rate of the radar is 35 ms and thus one range-Doppler image is generated every 35 ms. Each range-Doppler image contains 8 range bins and 128 Doppler bins.
- Each test was done as follows:
  - A specific gesture was continuously performed in front of the radar for about 10-15 seconds.
  - During this period a range-Doppler image is generated once every 35 ms.
  - The RDI is stored in x_training.csv. Each row of length 1024 corresponds to one image (8x128 numbers stored in a single row of 1024 elements).
  - For each row in x_training.csv, the gesture label is given in the corresponding row of y_training.csv. There are 4 gestures labelled 0,1,2,and 3.
- Multiple such tests are done and each test has a testid.
  - The file testid.csv records the testid for the entire data set.

| Model 1 | Model 2 | Model 3 | Model 4 |
|---|---|---|---|
| LSTM 128 | LSTM 256 | LSTM 256 | Conv3D 32(3× 3× 3) |
| Dense 4 | Dense 128, Relu | LSTM 128 | Conv3D 64(3× 3× 3) |
| - | Dense 4, Softmax | Dense 4, Softmax | Dense 256, Relu |
| - | - | - | Dense 4, Softmax |

Table 5.13: Model architectures

– Thus, consecutive rows correspond to the evolution of the gesture in time – as long as they belong to the same test.

### 5.3.2 Methodology

We analyze the effect of frame length using simple RNN and 3D CNN models. As we increase the frame length we capture a larger part of the gesture. For example, if the gesture "come" is performed in 350 ms then it would require 10 frames to capture the entire gesture. Similarly, different gesture take different amount of time to perform and hence require different number of frames to capture the entire gesture. The other concern is the start point of a continuous gesture. Being continuous, it is difficult to define a start point for these gestures.

To study the effect of window size, we started with a window size of 1 frame and consider up to 20 frames in steps of 1. The model architectures are listed in Table 5.13. Model 1 is an LSTM layer followed by a dense layer. With a window size of 1 i.e, frame-level classification an accuracy of 70% is achieved which goes up to 92% as we increase the window size (refer Fig. 5.14).

Model 2 has an LSTM layer with two dense layers which results in an improvement in classification accuracy by 3-4% and shows a similar trend as Model 1 (refer Fig. 5.15). This indicates that the performance can be improved up to a certain level by increasing the model complexity.

Model 3 uses two LSTM layers followed by a dense layer. There is a significant improvement in performance by using a deeper LSTM layer as compared to an additional dense layer in Model 2. With Model 3, classification accuracy of more than 98% is achieved with a frame length of 10 (refer Fig. 5.16). It is noted that increasing the frame length does not result in any improvement in performance. This may be due

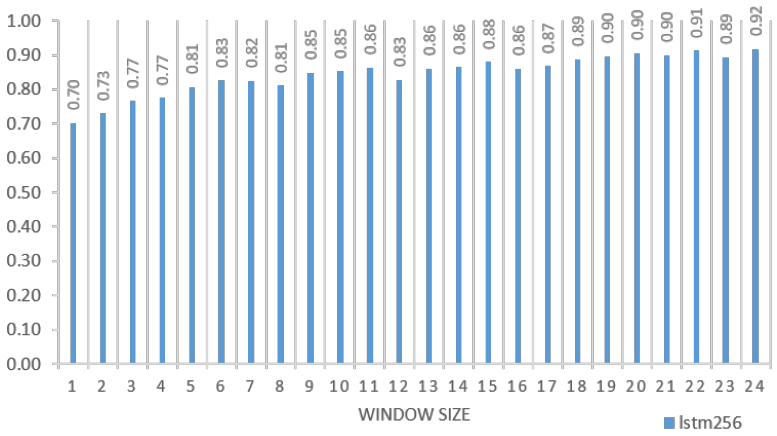



Figure 5.14: Classification accuracy versus window length for Model 1.

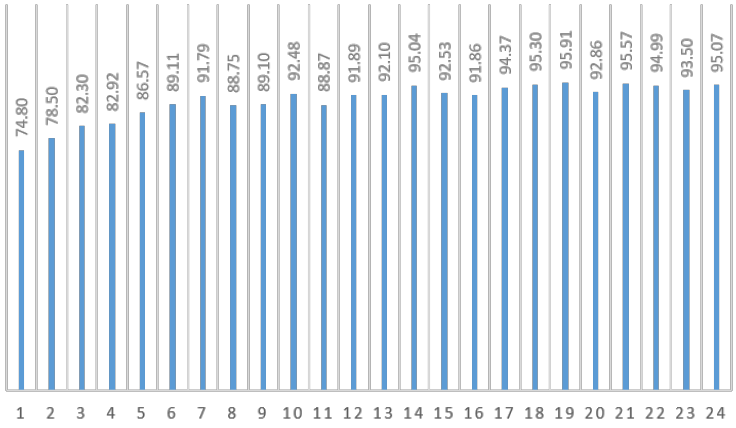



Figure 5.15: Classification accuracy Versus window length for Model 2.

to multiple instances of the gesture within the window length which does not provide additional information. The confusion matrix for the case with window length of 13 is shown in Fig. 5.17a and the average classification accuracy is 98.76%.

Model 4 is a 3D CNN model which has two layers of CNN followed by two dense layers. It uses 2D RDIs stacked in sequence of length determined by the window size. Accuracy of 90.57% is achieved with a window size of 1 and a rapid increase in accuracy is observed as the window size is increased till 8 resulting in 99.5% accuracy. The accuracy versus window size for Model 4 is shown in Fig. 5.18 and the confusion matrix is shown in Fig. 5.17b.

For Models 1-3, the classification accuracy increases as the window length increases. A larger window results in a longer inference time. Model 4 (3D CNN model) gives

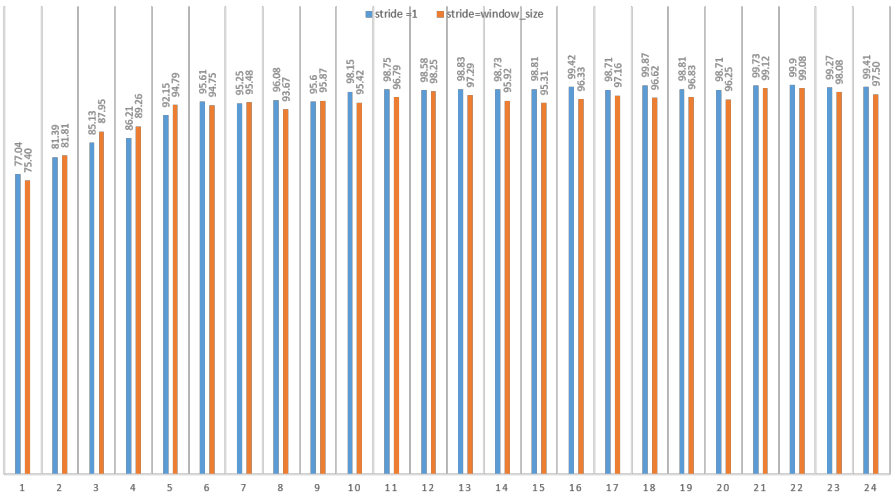
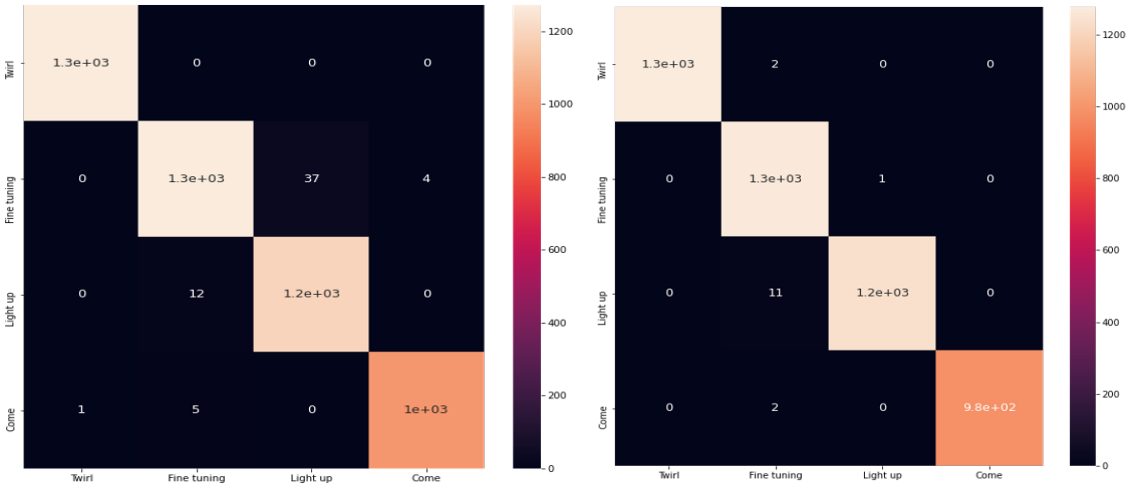



Figure 5.16: Classification accuracy versus window length for Model 3.




(a) Model 3

(b) Model 4

Figure 5.17: Confusion matrices

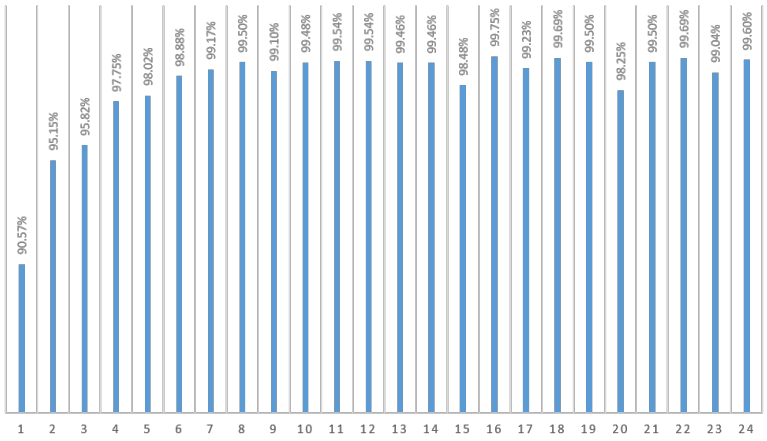



Figure 5.18: Classification accuracy versus window length for Model 4.

better accuracy with a shorter window length. The shorter window length and accuracy of 99.5% makes the 3D CNN model ideal for the real-time continuous gesture recognition task.

# Chapter 6

# Conclusion and Future Work

## 6.1 Conclusion

In this thesis we presented techniques for hand gesture recognition using radar data. We have developed the models using publicly available data sets such as the Soli and TinyRadar data sets. We found that 1D CNN with hand-crafted features result in 98% accuracy on 11 gestures and perform very well for fine gestures such as pinch pinky, finger rub, finger slide, pinch index as compared to easily classifiable gestures such as push, pull, swipe. As the model is based only on CNN (and not LSTM) it has 106 times fewer parameters and the inference can be carried out using parallel computation and real time results can be obtained. A small model size is necessary for implementation on resource constrained devices. TinyRadar data set contains very few frames per gesture instance and the performance for fine gestures is not very good. Due to the fewer number of frames a smaller model size can be achieved which translates to faster inference and low memory requirement. There is a trade-off between model size and complexity of the gestures we would like to classify. Fewer frames result in a smaller model size but do not work well with fine gestures. Raw ADC data gives similar result as RDIs, which eliminates the need of pre-processing steps for the TinyRadar data set. We also demonstrate that the model performs well on single user compared to the multiple user data set (97% vs 85% accuracy).

## 6.2 Future Research Directions

As our data set collection task is now completed, we can apply the methods described in this thesis for our data set. We can explore the trade-off between model size , computational complexity, and the number of gestures including fine gestures. We have noted that RAIs are beneficial for directional gestures and we can study this further as our data set contains directional gestures. With the high frame rate (50 frames per gesture instance) used for this data set, we can also study the use of raw data to understand if we can obtain improvement not only in performance but in terms of computational power and inference time on resource constrained devices for real time gesture recognition. In addition, we can study the data collected from left handed users.

# Appendix A

# Code

## A.1 Lua Script

```
reloadCfg = 1;

if reloadCfg == 1 then
    ar1.SOPControl(2)
    ar1.Connect(22,921600,1000)
    ar1.DownloadBSSFw("C:\\ti\\mmwave_sdk_03_03_00_03\\
                    firmware\\radarss\\iwr6xxx_radarss_rprc.bin")
    ar1.DownloadMSSFw("C:\\ti\\mmwave_studio_02_01_00_00\\
                     rf_eval_firmware\\masterss\\xwr68xx_masterss.bin")
    ar1.PowerOn(0, 1000, 0, 0)
    ar1.RfEnable()
    ar1.ChanNAdcConfig(1, 1, 1, 1, 1, 1, 1, 2, 1, 0)
    ar1.RfLdoBypassConfig(0x1)
    ar1.LPModConfig(0, 1)
    ar1.RfInit()
    ar1.DataPathConfig(513, 1216644097, 0)
    ar1.LvdsClkConfig(1, 1)
```

```
    ar1.LVDSLaneConfig(0, 1, 1, 0, 0, 1, 0, 0)
    RSTD.Sleep(500)
    ar1.SelectCaptureDevice("DCA1000")
    ar1.CaptureCardConfig_EthInit("192.168.33.30", "192.168.33.180",
                    "12:34:56:78:90:12", 4096, 4098)
    ar1.CaptureCardConfig_Mode(1, 2, 1, 2, 3, 0)
    ar1.CaptureCardConfig_PacketDelay(125)
end

ar1.ProfileConfig(0, 60.50002, 82, 6, 34, 0, 0, 0, 0, 0, 0,
                    102.908, 0, 64, 2350, 0, 0, 42)

RSTD.Sleep(500)
ar1.ProfileConfig(1, 60.50002, 8, 6, 34, 0, 0, 0, 0, 0, 0,
102.908, 0, 64, 2350, 0, 0, 42)
RSTD.Sleep(500)
ar1.ChirpConfig(0, 0, 0, 0, 0, 0, 0, 1, 0, 0)
RSTD.Sleep(500)
ar1.ChirpConfig(1, 1, 1, 0, 0, 0, 0, 0, 1, 0)
RSTD.Sleep(500)
ar1.ChirpConfig(2, 2, 1, 0, 0, 0, 0, 0, 0, 1)
RSTD.Sleep(500)
ar1.FrameConfig(0, 2, 450, 128, 35, 0, 0, 1)
RSTD.Sleep(500)
```

## A.2 Python Code to read raw binary data

```
# define some constant parameter
num_of_frames=50
num_of_chirp_per_frame=128
num_adc_sample_per_chirp=64
```

```
num_range_bin=64
num_virtual_antennas=12
chirp_cofig_per_chrip=3
num_real_channel=8
total_sample_per_frame=
                num_real_channel*
                num_adc_sample_per_chirp*
                chirp_cofig_per_chrip*
                num_of_chirp_per_frame

import numpy as np

path="C:\\Users\\Chandragupta\\Desktop
        \\gesture_samples\\P2\\Arc\\Inst1\\1.bin"
def read_bin_file(path):
    data_file = np.fromfile(path, dtype=np.int16)
    data = data_file.reshape(
                num_of_frames,
                total_sample_per_frame // 4,
                4
                )
    rearrange_data = np.copy(data)
    #convert the data to  Rx0I0, Rx0Q0, Rx0I1, Rx0Q1, ...
    rearrange_data[:, :, 2] = data[:, :, 1]
    rearrange_data[:, :, 1] = data[:, :, 2]
    del (data)
    rearrange_data =
    rearrange_data.reshape(
                num_of_frames, 2 * num_adc_sample_per_chirp,
                num_real_channel // 2 * chirp_cofig_per_chrip,
                num_of_chirp_per_frame
                )
```

```
    rearrange_data = np.swapaxes(rearrange_data, 1, 3)
    #covert to Complex no
    radar_data_real = rearrange_data[:, ::2, :, :]
    radar_data_imag = rearrange_data[:, 1::2, :, :]
    radar_data = 1j * radar_data_imag
    radar_data += radar_data_real
    radar_data = np.moveaxis(radar_data, [0, 1, 2, 3], [0, 2, 1, 3])
    return radar_data
end=0
```

## A.3 Parameters

| **Parameters** | **Value** | **Remark** |
|---|---|---|
| Number of frames | 50 | |
| Number of chirps per frame | 128 | |
| Number of ADC samples per chirp | 64 | |
| Number of range bins | 64 | |
| Number of virtual antenna | 12 | 3 Tx and 4 Rx |
| ChirpConfigs per chirp | 3 | 3 Tx used |
| Number of real channels | 8 | 4 Rx with real and imaginary parts |
| MAX_RANGE_BIN | Up to number of range bins | Variable for range bin in final output |

# Appendix B

# Experimental Results

## B.1 Soli Data Set

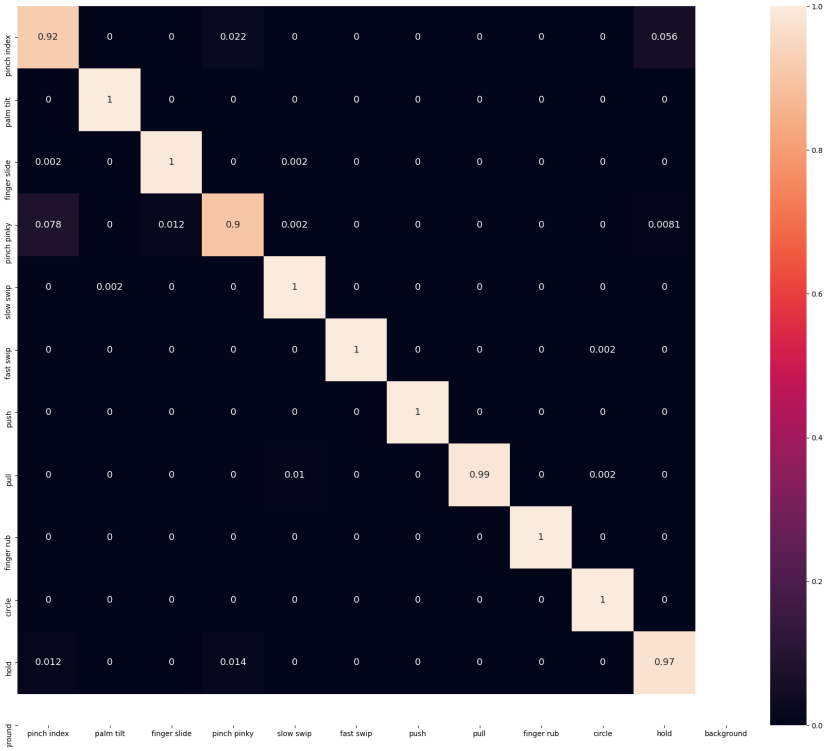



Figure B.1: Confusion matrix for the LSTM model.

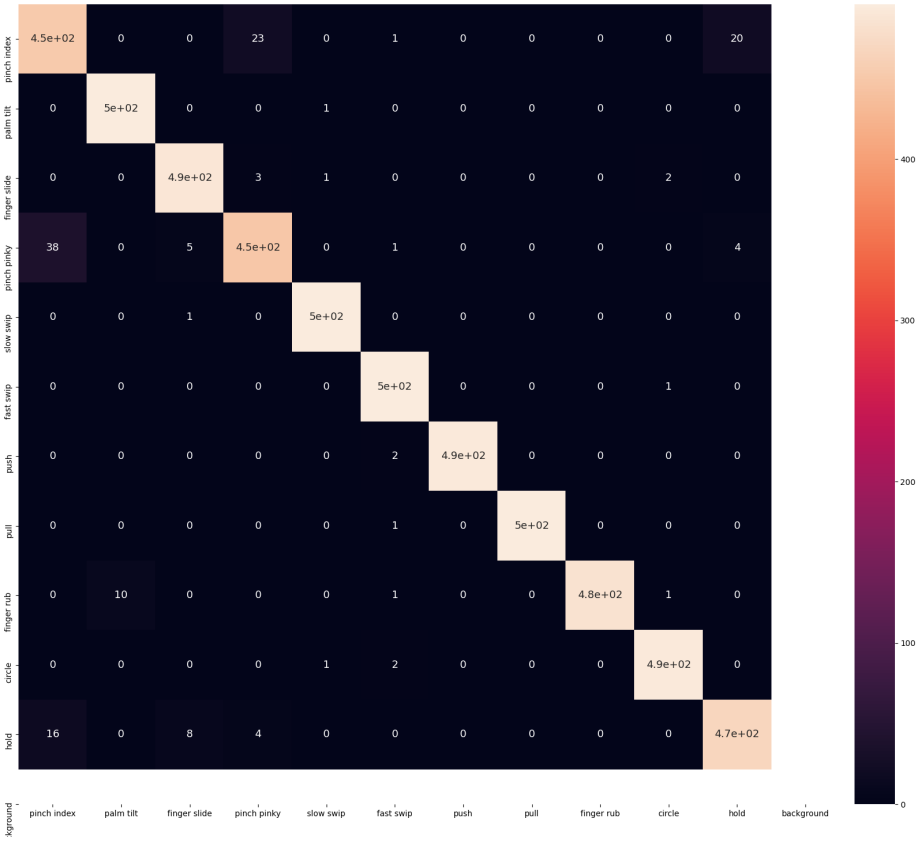



Figure B.2: Confusion matrix for the CNN+LSTM model.

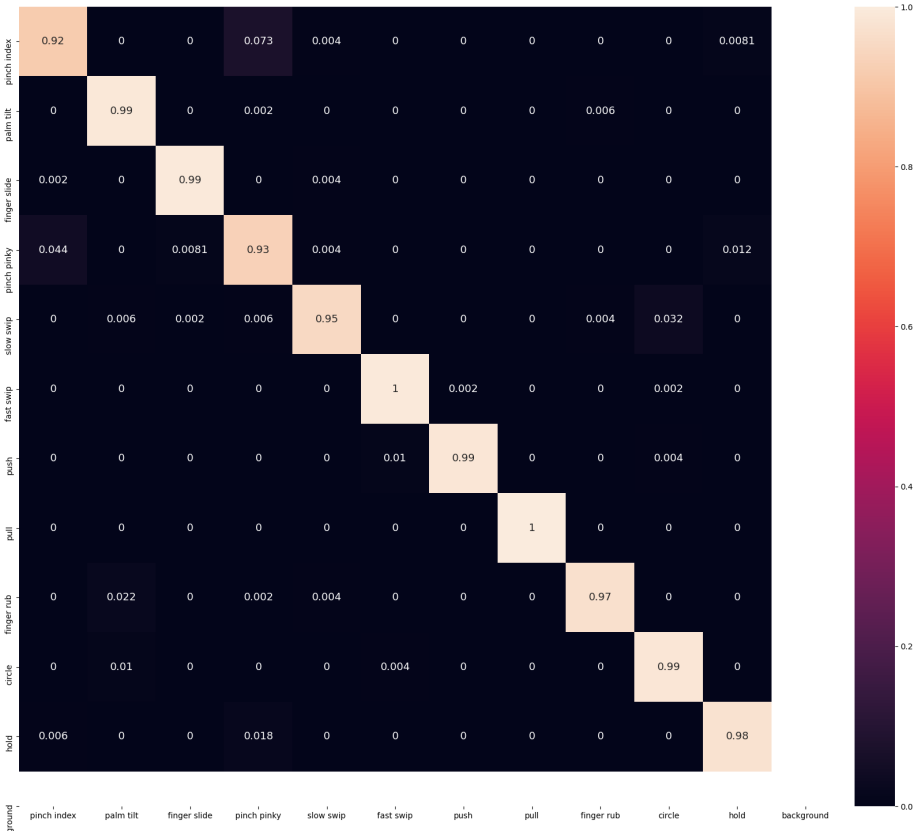



Figure B.3: Confusion matrix for the 3D CNN model.

### B.1.1 1D CNN Model

The architecture of 1D CNN model is given in Table B.1. It takes an input of shape (256,80). Layers 1, 2, and 3 use dilated convolution with dilation of 1, 2, and 4, respectively. After the 3rd layer we narrow down the output to (32,1) which is then passed to a fully connected layer with 11 output classes.

| Layer no. | Parameters | Output shape |
|---|---|---|
| 1 | kernel_size=4, stride=1, dilation=1 | (256,77) |
| 2 | kernel_size=4,stride=2, dilation=2 | (128,36) |
| 3 | kernel_size=4, stride=2, dilation=4 | (64,12) |
| 4 | kernel_size=4, stride=2, dilation=1 | (32,5) |
| 5 | kernel_size=5, stride=1, dilation=1 | (32,1) |
| 6 | fully connected, 11 | 11 |

Table B.1: 1D CNN model architecture

## B.2 TinyRadar Data Set

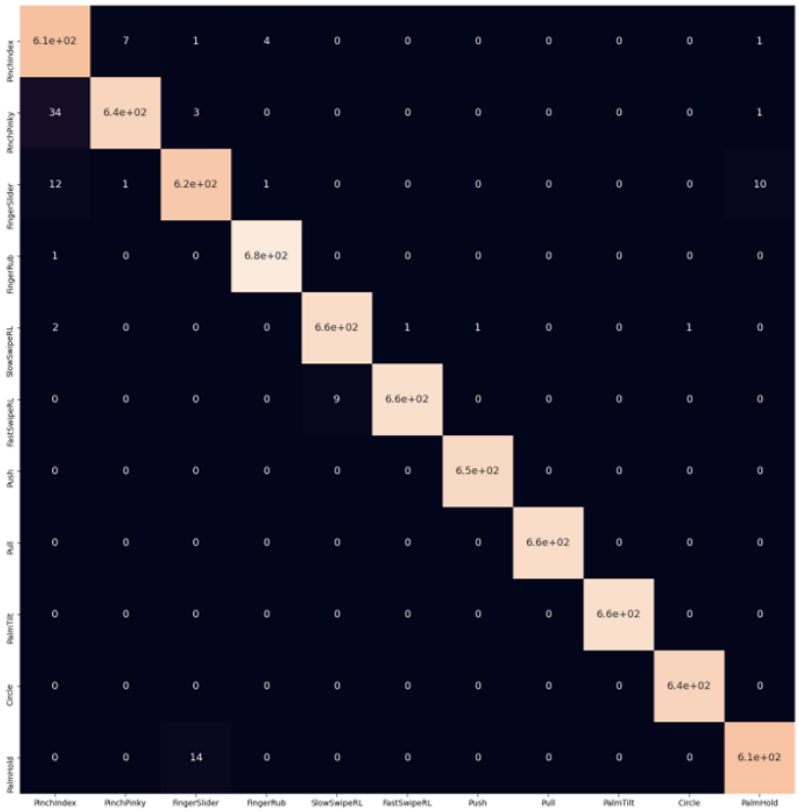



Figure B.4: Confusion matrix for single user (accuracy = 96.93%).

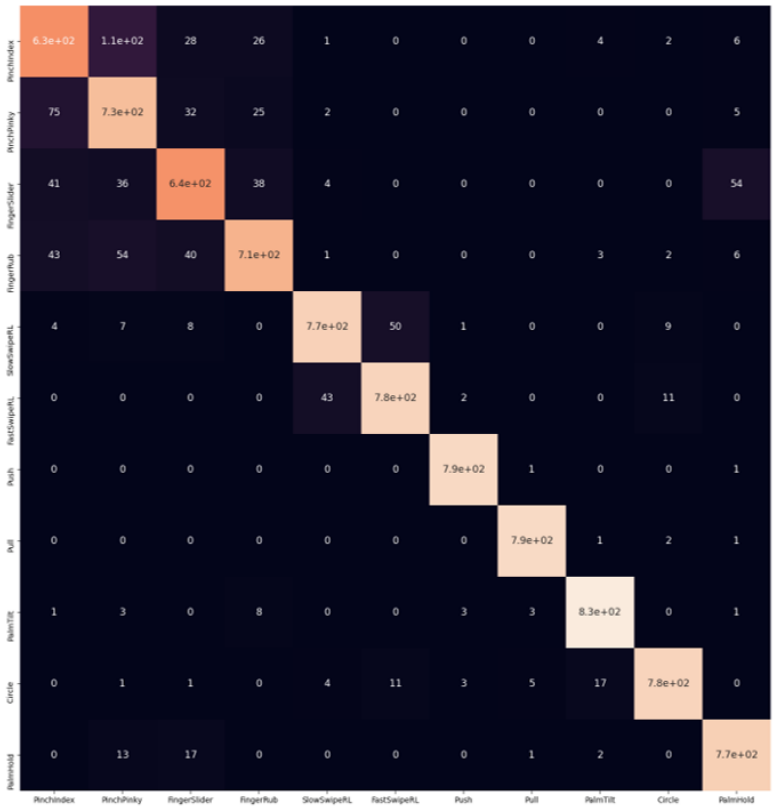



Figure B.5: Confusion matrix for multiple users (accuracy = 85%).

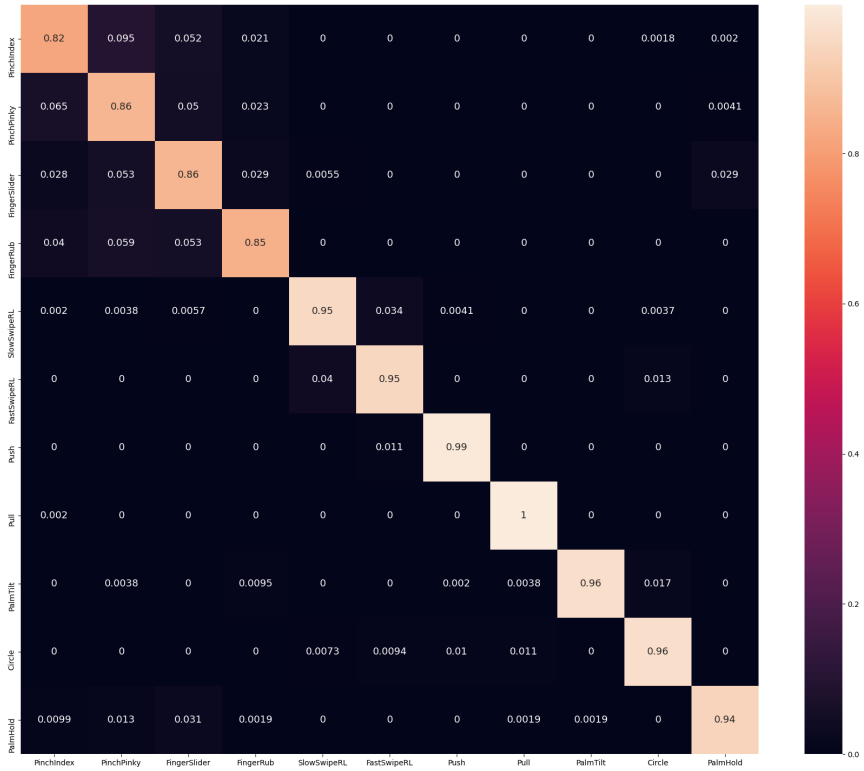



Figure B.6: Confusion matrix for the 3D CNN model.

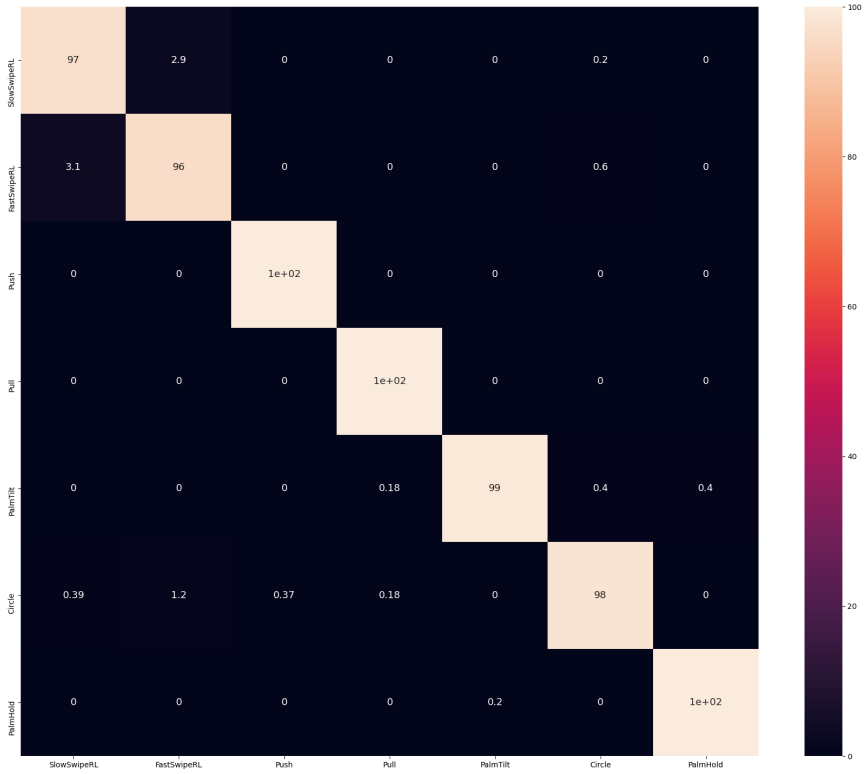
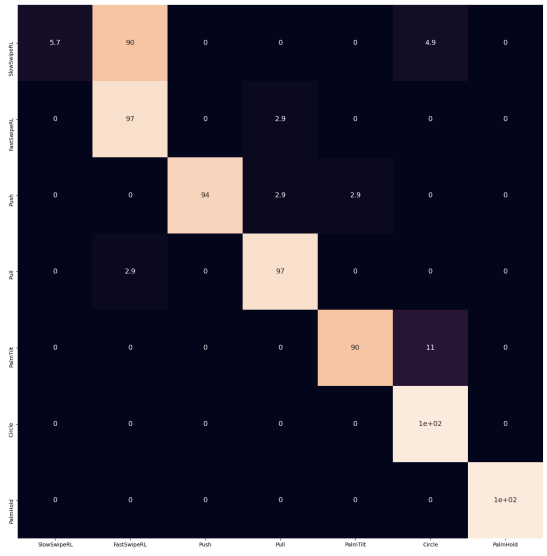


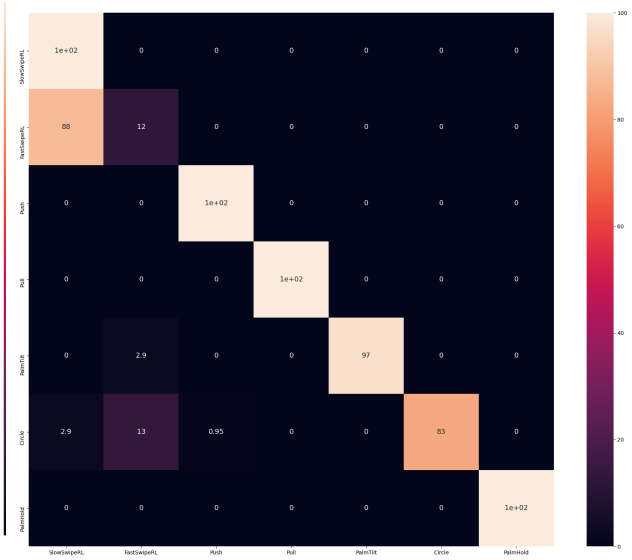

Figure B.7: Confusion matrix for the set of 7 gestures.

| User | 11 Gestures | 7 Gestures | 4 Gestures |
|---|---|---|---|
| 1 | 76.56% | 84.56% | 71.67% |
| 2 | 80.00% | 96.05% | 56.90% |
| 3 | 81.90% | 95.10% | 64.52% |
| 4 | 80.26% | 92.79% | 60.00% |
| 5 | 72.65% | 94.84% | 52.38% |
| 6 | 75.13% | 90.20% | 54.05% |
| 7 | 79.13% | 95.10% | 56.19% |
| 8 | 66.67% | 82.86% | 33.33% |
| 9 | 81.59% | 97.85% | 62.86% |
| 10 | 69.36% | 83.26% | 50.95% |
| 11 | 80.58% | 99.31% | 63.33% |
| 12 | 83.29% | 98.10% | 62.14% |
| 13 | 75.24% | 88.57% | 47.14% |
| 14 | 70.82% | 93.70% | 35.73% |
| 15 | 79.91% | 96.19% | 54.05% |
| 16 | 81.99% | 94.29% | 64.52% |
| 17 | 80.43% | 84.63% | 74.52% |
| 18 | 79.57% | 94.29% | 57.14% |
| 19 | 90.39% | 99.73% | 74.76% |
| 20 | 73.91% | 80.44% | 58.57% |
| 21 | 81.90% | 90.61% | 67.14% |
| 22 | 71.69% | 91.70% | 47.38% |
| 23 | 84.16% | 99.59% | 59.29% |
| 24 | 83.55% | 95.10% | 60.71% |
| 25 | 76.36% | 94.29% | 44.29% |
| **Average** | 78.28% | 92.53% | 57.34% |

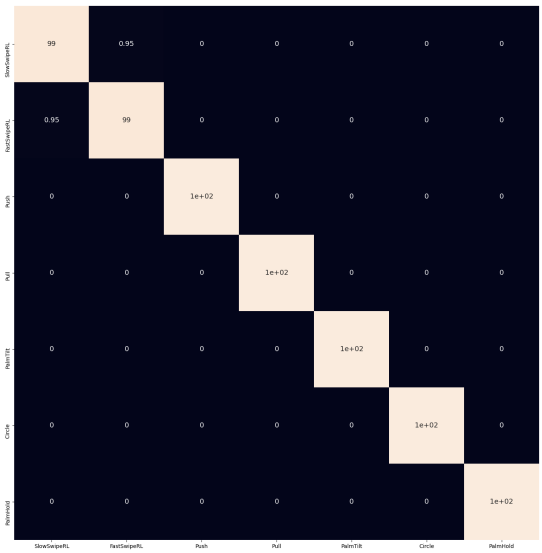
Table B.2: Leave one out cross validation for the 3D CNN model.



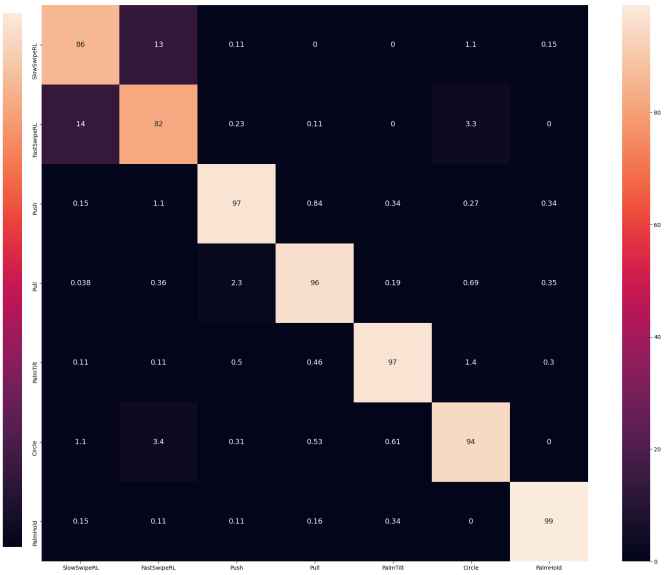

(a) User 10




(b) User 17




(c) User 19




(d) Average over all users.

Figure B.8: Confusion matrix for LOOCV

| User | 11 Gestures | 7 Gestures | 4 Gestures |
|---|---|---|---|
| 1 | 77.08% | 86.48% | 76.19% |
| 2 | 82.08% | 97.82% | 60.71% |
| 3 | 83.20% | 94.15% | 69.52% |
| 4 | 80.52% | 94.29% | 63.57% |
| 5 | 75.20% | 93.86 % | 56.19 % |
| 6 | 75.04 % | 90.76 % | 51.90 % |
| 7 | 79.13 % | 97.28 % | 53.10 % |
| 8 | 65.45 % | 88.30 % | 44.05 % |
| 9 | 82.48 % | 99.14 % | 57.86 % |
| 10 | 68.41 % | 81.48 % | 55.71 % |
| 11 | 77.34 % | 99.59% | 64.52% |
| 12 | 79.22 % | 95.51% | 59.52 % |
| 13 | 71.17 % | 79.73 % | 49.29 % |
| 14 | 73.12 % | 91.18 % | 34.77 % |
| 15 | 76.19 % | 96.05 % | 51.90 % |
| 16 | 77.84 % | 87.76 % | 55.24 % |
| 17 | 73.25 % | 85.17 % | 77.86 % |
| 18 | 78.79 % | 89.39 % | 47.86 % |
| 19 | 87.79 % | 93.74 % | 72.86 % |
| 20 | 71.20 % | 81.40 % | 54.52 % |
| 21 | 80.78 % | 92.52 % | 61.19 % |
| 22 | 68.40 % | 89.6 | 43.33 % |
| 23 | 81.82 % | 99.40 | 60.00 % |
| 24 | 80.09 % | 97.01 % | 65.48 % |
| 25 | 73.85 % | 90.61 % | 47.86 % |
| **Average** | 76.78 % | 91.70 % | 57.40 % |

Table B.3: LOOCV for the CNN+LSTM model

# Bibliography

[1] M. Amin, Z. Zeng, and T. Shan. Hand gesture recognition based on radar micro-doppler signature envelopes. pages 1–6, 04 2019.

[2] A. Butler, S. Izadi, and S. Hodges. Sidesight: Multi-"touch" interaction around small devices. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology*, UIST '08, page 201–204, New York, NY, USA, 2008. Association for Computing Machinery.

[3] J.-W. Choi, S.-J. Ryu, and J.-H. Kim. Short-range radar based real-time hand gesture recognition using lstm encoder. *IEEE Access*, 7:33610–33618, 2019.

[4] E. Enge. Mobile vs. desktop usage in 2020.

[5] D. C. Engelbart and W. K. English. A research center for augmenting human intellect. In *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, AFIPS '68 (Fall, part I), page 395–410, New York, NY, USA, 1968. Association for Computing Machinery.

[6] R. Ford. Google soli technology.

[7] S. Hwang, M. Ahn, and K.-y. Wohn. Maggetz: Customizable passive tangible controllers on and around conventional mobile devices. In *Proceedings of the 26th Annual ACM*

*Symposium on User Interface Software and Technology*, UIST '13, page 411–416, New York, NY, USA, 2013. Association for Computing Machinery.

[8] Y. Kim and B. Toomajian. Hand gesture recognition using micro-doppler signatures with convolutional neural network. *IEEE Access*, 4:7125–7130, 2016.

[9] M. Le Goc, S. Taylor, S. Izadi, and C. Keskin. A low-cost transparent electric field sensor for 3d interaction on mobile devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, page 3167–3170, New York, NY, USA, 2014. Association for Computing Machinery.

[10] S. Lee, W. Buxton, and K. C. Smith. A multi-touch three dimensional touch-sensitive tablet. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '85, page 21–25, New York, NY, USA, 1985. Association for Computing Machinery.

[11] J. Lien, N. Gillian, M. E. Karagozler, P. Amihood, C. Schwesig, E. Olson, H. Raja, and I. Poupyrev. Soli: Ubiquitous gesture sensing with millimeter wave radar. *ACM Trans. Graph.*, 35(4), July 2016.

[12] T. Mantecón, C. R. del Blanco, F. Jaureguizar, and N. García. Hand gesture recognition using infrared imagery provided by leap motion controller. In J. Blanc-Talon, C. Distante, W. Philips, D. Popescu, and P. Scheunders, editors, *Advanced Concepts for Intelligent Vision Systems*, pages 47–57, Cham, 2016. Springer International Publishing.

[13] A. N. Pisarchik, V. A. Maksimenko, and A. E. Hramov. From novel technology to novel applications: Comment on "an integrated brain-machine interface platform with thousands of channels" by elon musk and neuralink. *J Med Internet Res*, 21(10):e16356, Oct 2019.

[14] Q. Pu, S. Jiang, and S. Gollakota. Whole-home gesture recognition using wireless signals (demo). *SIGCOMM Comput. Commun. Rev.*, 43(4):485–486, Aug. 2013.

[15] T. Sakamoto, X. Gao, E. Yavari, A. Rahman, O. Boric-Lubecke, and V. Lubecke. Hand gesture recognition using a radar echo iq plot and a convolutional neural network. *IEEE Sensors Letters*, 2:1–4, 2018.

[16] M. Scherer, M. Magno, J. Erb, P. Mayer, M. Eggimann, and L. Benini. Tinyradarnn: Combining spatial and temporal convolutional neural networks for embedded gesture recognition with short range radars, 2020.

[17] J. Stowers, M. Hayes, and A. Bainbridge-Smith. Altitude control of a quadrotor helicopter using depth map from microsoft kinect sensor. In *2011 IEEE International Conference on Mechatronics*, pages 358–362, 2011.

[18] M.-C. Su and M.-T. Chung. Voice-controlled human-computer interface for the disabled. *Computing & Control Engineering Journal*, 12(5):225–230, 2001.

[19] J. S. Suh, S. Ryu, B. Han, J. Choi, J.-H. Kim, and S. Hong. 24 ghz fmcw radar system for real-time hand gesture recognition using lstm. In *2018 Asia-Pacific Microwave Conference (APMC)*, pages 860–862, 2018.

[20] I. E. Sutherland. Sketchpad a man-machine graphical communication system. *SIMULATION*, 2(5):R–3–R–20, 1964.

[21] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[22] S. Wang, J. Song, J. Lien, I. Poupyrev, and O. Hilliges. Interacting with soli: Exploring fine-grained dynamic gesture recognition in the radio-frequency spectrum. In *Proceedings*

*of the 29th Annual Symposium on User Interface Software and Technology*, pages 851–860. ACM, 2016.

[23] Y. Wang, S. Wang, M. Zhou, Q. Jiang, and Z. Tian. Ts-i3d based hand gesture recognition method with radar sensor. *IEEE Access*, 7:22902–22913, 2019.

[24] J.-T. Yu, L. Yen, and P.-H. Tseng. mmwave radar-based hand gesture recognition using range-angle image. In *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, pages 1–5, 2020.

[25] T. O. Zander and C. Kothe. Towards passive brain–computer interfaces: applying brain–computer interface technology to human–machine systems in general. *Journal of Neural Engineering*, 8(2):025005, mar 2011.

[26] J. Zhang, J. Tao, J. Huangfu, and Z. Shi. Doppler-radar based hand gesture recognition system using convolutional neural networks, 2017.

[27] C. Zhao, K.-Y. Chen, M. T. I. Aumi, S. Patel, and M. S. Reynolds. Sideswipe: Detecting in-air gestures around mobile devices using actual gsm signal. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, pages 527–534, New York, NY, USA, 2014. ACM.