

Deep Learning and mmWave Sensor-based Gesture Recognition

Author: Jack McNally (201431464)

Project Supervisor: Dr Junqing Zhang

Project Assessor: Dr Xinping Yi


Declaration of academic integrity

The standard University of Liverpool statement of academic integrity [6] should go here as follows:

I confirm that I have read and understood the University's Academic Integrity Policy.

I confirm that I have acted honestly, ethically and professionally in conduct leading to assessment for the programme of study.

I confirm that I have not copied material from another source nor committed plagiarism nor fabricated, falsified or embellished data when completing the attached piece of work. I confirm that I have not copied material from another source, nor colluded with any other student in the preparation and production of this work.

SIGNATURE.......... DATE.....27/04/22.....

Abstract

Privacy is a growing concern in all areas of the world and as gesture recognition systems become more common place the traditional camera model raises many privacy concerns. Millimetre wave radar poses a good candidate for a replacement to this traditional system due to radars invariance to weather effects, and the advancing CMOS technology allowing for mmWave radar sensors to become cheaper. This project therefore investigated the effectiveness of using mmWave sensors and deep learning to classify hand gestures in order to use them as a potential replacement for traditional camera-based detection. A successful model was implemented using the point cloud and radial velocity (doppler) values that was able to classify 5 non-static gestures with a test accuracy of over 90% in all classes, using a Texas Instruments (TI) IWR6843AOPEVM mmWave sensor and a CNN. Python was used to create and train the CNN and to also handle the interfacing of the mmWave sensor for data capture. The 5 gestures classified were: Left Swipe, Right Swipe, Down Swipe, Up Swipe, and Forward Palm. Around 2500 gestures were performed in order to capture the sensor data, using a custom data streaming script, which formed the input data to a CNN. Using the data streaming script live gesture classification was achieved allowing non-static gestures to be correctly detected and control a UI interface.

Table of Contents

SPECIFICATION TABLE:	4
1. INTRODUCTION	5
2. LITERATURE REVIEW	6
3. INDUSTRIAL, REAL WORLD AND SOCIETAL IMPACT	10
4. BACKGROUND THEORY	11
4.1. Millimetre Wave Radar Sensors	11
4.1.2. Constant False Alarm Rate	13
4.2. Convolutional Neural Networks	13
4 DESIGN AND METHOD	15
5.1. Project Design	15
5.2. Method	17
5 RESULTS	19
6 DISCUSSION	20
7.1. Analysis of Results	20
7.2. Issues that arose	22
7.3. Downsides of the project	22
7.4. Work Packages and Gantt Chart	23
7.5. Reasons for choosing a CNN over other potential options	23
8. CONCLUSION	24
9. REFERENCES	25
APPENDICES	28
a. Original Gantt Chart	29
b. Revised Gantt Chart	30
c. Data Streaming Code	31
d. CNN Code	35

Specification Table:

Phase	Work Package	Task	Deliverable	Milestone
Prep-Work	NA	Pre-Reading	NA	Specification Report - 15th Oct. 2021
		Specification Report	NA	
Configuration and Data Collection	Work Package 1	Configure mmWave Sensor parameters	Complete setup of sensors and have the configured values necessary to extract usable data. Verified with testing the sensor and receiving point-cloud outputs	15th Nov. 2021
		mmWave Sensor Data collection	Complete algorithm design and implementation and have collected data samples from the sensor. Verified by having a data set of point-clouds.	
CNN Development	Work Package 2	Design outline of basic CNN	Completed design of CNN algorithm	Project Presentation – Week before 20th Dec. 2021
		Decide on training and testing data split	Selected an appropriate split of data. Verified by providing an overfitting value of less than 10%	
		Train CNN on the collected data	Have a completed CNN designed and implemented which is verified with it being able to detect a gesture	
Functional System Testing and Optimization	Work Package 3 (Task 3.1)	Perform analysis on system attributes	Obtained data on the performance of the CNN which is verified by having an accuracy of above 80%	Bench Inspection – Week before 4th Apr. 2022
		Check performance against requirements and make basic alterations	Have a CNN with an accuracy of over 80% for 3 gestures	
	Work Package 3 (Task 3.2)	Review options for NN improvements	An outline of possible NN implementations	
		Begin NN redesign	Finish design of new CNN algorithm	
		Test Performance of the Redesign	Implementation of new CNN algorithm design	
Final Work	NA	Prepare for bench inspection	NA	Final Report - 22nd Apr. 2022
		Bench inspection	A working gesture recognition system with accuracy of over 90% on 3 different gestures	
		Final Report	Completed FYP report	

Figure 1: Specification Table showing the work packages and individual tasks.

1. Introduction

Gesture recognition systems are further being utilised more frequently in consumer products where their usefulness has been found to be very welcome in making the user experience more involving. They are also useful for critical safety and monitoring systems in places such as nursing homes where at risk patients can be monitored and any potential issues such as a fall being flagged by a gesture recognition system so that the patient can be aided. These are but the surface for the potential uses of this technology and so research interests have been invigorated in combining gesture recognition with common daily use systems such as home security, fall monitoring, device interfacing, etc [1].

The most common place method of performing this classification is via computer vision with the main sensor being a camera which collects image data and feeds it to a neural network. This method is a relatively cheap and simple way to classify gestures, but it raises a concern that the privacy of the user and bystanders, e.g., passengers in a car, can be compromised if this image data isn't handled correctly [2]. Modern consumers are growing increasingly aware of potential threats to their privacy and so new methods to implement gesture recognition need to be examined. The method proposed by this report is that of mmWave sensors which use RF sensing to gather information about the performed gesture which requires no sensitive information about the user in order to work while also being low cost and non-invasive unlike a wearable device solution [1][3].

Using a RF sensing over another type of radar sensor such as a lidar is beneficial as RF sensing is unaffected by weather and lighting conditions which plague other types of sensors like LiDAR's. With a mmWave sensor gesture recognition in a car for example can work just as well at night as it can in the day as the light in the car won't have any effect on the sensor like it would on a LiDAR [2]. Millimetre wave sensors also have the advantage of higher range and angular precision due to the smaller wavelength; this allows them much better performance than that of a camera which in order to reach near the same level of precision incurs large costs and computational increases. Their ability to penetrate through surfaces gives a unique ability to be unaffected by users wearing gloves or having their hands obfuscated by liquids such as water [4]. Millimetre wave sensors also have much higher accuracies compared to Wi-Fi and sonar signals as they have a higher resolution [5]. Their other advantages over LiDAR are in privacy as high resolution lidars contain enough information to recreate an accurate scene of the user and so present a privacy concern. Another such advantage is that mmWave sensors are becoming cheaper and more compact as advancements in modern CMOS technology have allowed even data processing to be performed on the sensor itself. Allowing them to become very cheap and powerful small-scale devices which can be embedded easily in smaller formfactors like inside of laptops for example [3]. This is compounded again by most modern smart phones including a mmWave solution embedded in the device and so the useability of mmWave radars as sensing solutions is greatly increased [4].

For this project a data streaming algorithm was created in order to be able to generate data from the sensor which was then used as training inputs to a deep neural network in order to classify 5 non-static gestures. These gestures are right hand swipe, left hand swipe, up swipe, down swipe, and forward palm. The model once trained was then integrated with the data streaming algorithm so that live data from the sensor could be used to predict the performed gesture and control a video player. A convolutional neural network (CNN) was employed in

order to classify the gestures with the input data from the mmWave sensor being in the form of a set of point cloud coordinates. The main idea being that the CNN can be used on radar data due to its ability to exploit the structural locality of these signals. The CNN uses a 1-dimensional convolutional layer which is normally called a temporal convolutional due to it sampling the incoming data in given time steps which are the same width as the kernels size.

The structure of the report is as follows, firstly a review of the current and most relevant literature surrounding the topic of mmWave sensors and gesture recognition. In this chapter a general overview of the details of the paper will be given and then any positives or negatives will be discussed, ending with a short conclusion on the papers impact on the field. Following the literature review the third chapter will highlight the relevance of the project to real world applications and industry, providing a discussion on its impact on society should it achieve mass adoption and how industry leaders could implement the technology in their own products. The fourth chapter focuses on the background theory of the project and so will explain the theory behind mmWave sensors and also how convolutional neural network's function. Chapter 5 will show the overall design of the project providing the flowcharts and block diagrams of the system with further explanations. Then finishing the chapter off discussing the method used in designing the system and how the data collection was performed. Chapters 6 and 7 will, in the former, present the results of the project and the latter will give a detailed discussion of the result and analysis the project overall, and finally chapter 8 will conclude the report, giving a brief summation of the project and its successes/failures.

2. Literature Review

D. Salami, et al. has a large cross over with this project as they developed a lightweight gesture recognition system using the point cloud data from a mmWave radar sensor [2]. They created and implemented an architecture called "Temporal Graph Self Attention Convolution (Tesla)" which is based on message passing graphical neural networks (MPNN) that performed 8 times faster than their current competitor and also had a 21% increase in accuracy, that was tested on 3 datasets. They used a Temporal Graph K-Nearest Neighbour algorithm so they could model the temporal nature of the changing point cloud for each frame of data as a graph structure, and then applied the "Tesla" convolutional layer in order to infer the gestures. They have achieved excellent results and the use of graphical neural networks could be a possible future improvement to this project as has been demonstrated in the paper they are extremely efficient and so would be suitable to allow the project to be ran on embedded devices. The downsides to this implementation are that it is a brand-new architecture and so experience and available information about it are low, this means that to use the architecture there is a learning curve which could be time consuming. Another downside is that the model showed improved performance for gestures that were performed at fast and normal speeds but worse performance for slow gestures. Depending on the planned areas of deployment this could present an issue with accuracy as for example if it was for use with elderly persons whose performance of gestures might be slower than younger persons.

H. Liu et al., created a gesture recognition system to control a UI called "M-Gesture" that was designed to be independent of a person's unique hand features, such as hand size, direction of gesture, heat dispersion, etc. that cause RDI based solutions to lose accuracy [4]. Their solution is to replace RDI with a pseudo representative model (PRM) contains the

useful information extracted from RDI that is supplemented with the azimuth angle information in order to provide spatial context to the gesture. This data is fed into an Equal-volume-learning neural network (EVL-NN) which leverages convolutional layers; this EVL-NN is used in the PRM method as the number of pixels in the “image” is much less than standard image data. They compared their accuracy to that of Google’s Soli sensor for gesture recognition and achieved an accuracy of 99.76% for familiar users which is 0.1% higher than that of model using the Soli sensor. The large improvement comes from the 98.70% accuracy achieved for new users which was around 9% higher than the model using Soli sensors. This research has a lot of cross-over to the project as their implementation of the gestures controlling UI is the same as what this project achieved and could be an effective way to increase the accuracy of new users, giving it an edge when being deployed in a commercial setting.

J -T. Yu, L. Yen, and P. -H. Tseng used range-angle image (RAI) spectrograms as the input to a convolutional neural network and LSTM (CNN-LSTM) neural network in order to classify hand gestures [5]. Their solution differs from other papers who chose to use spectrograms as they normally use range-Doppler images (RDI) spectrograms whereas J -T. Yu, L. Yen, and P. -H. Tseng have chosen RAI. RAI has the benefit of improved horizontal information about the gestures over RDI as the resulting spectrogram from RDI has the same start and end point for every gesture. RAI also allows for extra information such as range changes from the vertical motion of the gesture as RDI only considers the velocity of the gesture. Their results concluded a 9% increase in average accuracy for RAI over RDI solutions and over 11% for the combined solution of using both RAI and RDI images for each gesture. In terms of applicability to the project it is a viable alternative to implementing gesture recognition using mmWave sensors and could allow more differentiability between harder to classify gestures, like finger movement. It does have downsides though as all outputs from the sensor need to be converted into spectrograms which increases the performance overhead by a large margin. In training the data pre-processing is increased massively as a large data set will be needed in order to train a CNN effectively and it will be more time consuming than point cloud methods discussed in D. Salami, et al [2]. Lastly real time application of this method may prove too resource intensive as real time conversion of data to spectrograms will be strenuous on lower-powered systems like laptops and embedded devices.

J. Wu, et al [6] also uses spectrograms like those used in J -T. Yu, L. Yen, and P. -H. Tseng [5] but they instead leverage the use of a unique coordinate system that allows them to convert traditional RDI spectrograms into intrinsic spectrograms. Their method proves to be better than the traditional RDI spectrograms when the locations the gestures are performed in change, as the RDI spectrograms will have noise from the background present causing accuracy losses when the sensor is moved into a different space. They have mainly focused on the data processing aspect of the research and their choice of neural network is a 2D convolutional one with a standard implementation of 4 layers, max pooling, and a LeakyReLU operation for faster training. They were able to achieve a test accuracy of 88.4% of 10 different gestures which is a respectable amount of accuracy as they were limited by their hardware as they only had a mmWave sensor with a single antenna and so weren’t able to leverage the angle of arrival (AoA) which would have increased the accuracy. The downsides of this method are the same as any that use spectrograms and that is the extra processing time, and performance decreases that come with the conversion of the sensor data to spectrogram images. The use of this coordinate system to make more robust RDI spectrograms could be an interesting direction to improve future revisions of the project

though it might be redundant since the RAI method shown in J.-T. Yu, L. Yen, and P.-H. Tseng [5] is more accurate and simpler to achieve.

J. W. Smith, et al. created a unique method for attempting to classify static hand gestures by developing a mechanical scanner that manoeuvres the mmWave sensor around a static hand to create a 2D SAR image of the gesture [7]. Their main concern was with improving the recognition accuracy of these gestures as the human hand has a small radar cross-section which causes a low SNR and therefore the features of the gesture are hard to distinguish. They generated data with the scanner on a real human hand and also on an aluminium model, they then used the data generated from these different materials to train 4 different convolutional neural networks. 2 of which were only trained on data from the real human hand and the other 2 from a combination of the real human hand and aluminium model, in each pair one CNN was using range data and the other range-angle data. Their results showed that the range-angle CNNs performed better than the purely range based data CNN by around 5% for the real human hand dataset and 2% for the combined dataset. They also showed an increase in accuracy of over 5% for the combined dataset compared with the standalone version giving their best accuracy of 95.4% for the range-angle combined dataset. The results provide a possible solution to including static gestures into the project and could be a future potential implementation. Though it has large downsides as they required an aluminium hand model to be made in order to gain the increased accuracy from the larger radar cross-section. Along with this the method for data collection is quite advanced, using a mechanical scanner to create the SAR images, which will increase data collection time and also potential issues that could arise due to malfunctions. Lastly the dataset only consisted of 3 gestures and so an accuracy of around 95% might drop off as more gestures are added, especially if the gestures become more similar and so their differentiation becomes increasingly challenging.

S. M. Kwon et al., use human pose estimation in order to classify human gestures using convolutional neural networks and mmWave sensors [8]. Using a camera in conjunction with the mmWave sensor they leverage OpenPose (A human pose estimation method) to create a matrix of key points from the camera image of the gesture. They take the Doppler-FFT transform data from the mmWave sensor (velocity) and combine it with this key point matrix, this is then used as the training data for their CNN. They trained the model to classify three different human activities, stretching, dumbbell raise, and sitting down. Their implementation has a large strength in that the amount of data needed is low with only 2 images and a total of 150 frames of data needed in order to classify one activity. It is also capable of working with different backgrounds which is a great boon as lots of mmWave implementations struggle with background noise lowering accuracy. While their research focuses on recreating the human skeleton for the activities in order to classify them there is an interesting potential reapplication of this method to hand gestures. As Openpose can also work with pictures of the hand to highlight the hands' skeleton structure. This would be a great way to provide differentiability that static hand gestures need when using mmWave radar.

Y. Ren et al., researched into classifying hand gestures using the built in mmWave sensors in modern smart phones and used the RDI information from the sensor to train a CNN model and an CNN-LSTM model on 5 non-static gestures [9]. The CNN-LSTM model achieved an accuracy of over 95% and the pure CNN model of over 93% with the pure CNN model being developed to run on devices and so a less resource intensive model was used. They use a pattern restoring algorithm in order to lower the effect of lost frames on the classification's accuracy due to the performance constraints of phones. This algorithm improves performance, lowers hardware requirements, and also allowed them to increase the frames

per second from 4fps to 7fps. They also trained the model on the noise inherent in the radar which allowed them to improve the accuracy of detection as RDI methods, discussed above, suffer from background noise. Y. Ren et al., has a large amount in common with the project and so the methods like the pattern restoring algorithm could have useful future implementations in the project. Though there is again the issue of reliance on RDI imaging in order to classify gestures and the inherent problems of new users or new backgrounds causing lower accuracy and while they create solutions to this problem an implementation such as the one in D. Salami, et al. [2] would be more advantageous. G. Zhang, et al., [10] also used a similar comparison with RDI images and two different CNN types, though not focusing on mobile devices, and were able to achieve an accuracy of 98% for the CNN-LSTM model they used. This gives backing to Y. Ren, et al. CNN architecture being a good way to classify gestures.

P. S. Santhalingam, et al., created a model to classify the sign language gestures used in the American Sign Language (ASL) called ExASL, that also considered the non-manual markers of the sign language such as body movement in order to allow for a true implementation of the language [10]. The implementation uses CNNs in conjunction with LSTMs and provides different models that use different types of clustering. They were able to achieve an accuracy of over 92% and a word error rate of 1.25%. The research has great overlap with the project as they also use a point cloud implementation in order to model the top portion of the body with 3 different objects acting as the body, left hand, and right hand.

Research has also gone into some very interesting and novel human user interface (HUI) implementations that use mmWave sensors. One is to detect handwriting in S. D. Regani, et al., [13] and using finger gestures to control a mouse cursor in Z. Li, et al [13]. S., -D., R., et al., created a tracking system that used the mmWaves doppler values to narrow down a power threshold where the pen was mostly likely used and then combining this with azimuth and elevation data to allow the gesture to be tracked. Tests were ran using the resulting detected handwriting images as input to a CNN that classified the handwriting at an accuracy of 96.6%. While this isn't a gesture system per-se the act of handwriting can be considered one and so the methodology applied still holds value for future improvements to the project. Z., Li, et al., were able to achieve a good implementation of the gesture-controlled cursor with the model only having performance issues in the z coordinate plane.

Y. Sun et al., [14] focused on improving the performance of the deep learning networks so that they run more efficiently in edge AI implementations. They were able to reduce the model's computational complexity by 27% and only suffering an accuracy loss of 2%. A. Ninos, J. Hasch, and T. Zwick [15] classified non-static gestures with an accuracy of 94.3% by leveraging the AoA, azimuth angle, and elevation angle. While not a recognition project A. Ninos, J. Hasch, M. E. P. Alvarez, and T. Zwick [16] created a method to artificially create radar gesture data using blender which could reduce the data collection overhead by a large margin when training future gestures for the project. Lastly J. W. Smith, O. Furxhi and M. Torlak. [17], used a FCNN to classify gestures in order to control a theremin by taking the RMA images of gestures performed in front of the sensor as training data. They showed that the mmWave sensor has great tracking capabilities with objects and so proves that it is suitable for HUI tasks.

3. Industrial, Real World and Societal Impact

Classification of gestures is becoming more prevalent in many places in society as users wish to interface with their devices and systems in a more natural manor compared to a mouse or keyboard. This makes the project an interesting and relevant research aim as it explores a section of engineering that is, while not new, constantly evolving and becoming more ubiquitous. Not only that, while the main goal of the project was to design a gesture classification system, research into deep learning and mmWave sensing have also been performed. The project has been able to design a CNN that is able to classify gestures in real time from a mmWave sensor with a relatively small amount of training data and network layers. Research into ways to implement this proves invaluable to further research into the area with any future research into mmWave sensors and deep learning being able to expand upon the successes of this project and also avoid its pitfalls.

In terms of industrial application, the project has a myriad of potential use cases in various industries from car manufacturers, hospitals/care homes, to hardware companies like Microsoft etc. All areas have some use cases for gesture recognition technology whether that be in increasing user safety or experience. Also, with further growing regulation and data protection laws the move to privacy conscious solutions for gesture recognition such as this project removes companies from any potential lawsuits.

In the automotive industry there is a growing focus on the drivers experience inside the car being just as important as how the car actually handles. This can be seen in the increasing amount of entertainment systems being placed into modern motor vehicles. High end car manufacturers have added gesture controls to these entertainment systems where the driver or passenger can control all its entertainment functions via gestures. This gives the occupants the ability to have a more natural interaction experience with their car by controlling for example the station and volume via swipes and finger curls. This gesture control is starting to be seen in the mid-range of cars as well, meaning that over the next few decades this type of technology will become more commonplace and so companies in the automotive industry are looking for other ways to implement this technology. This project fits perfectly in this market as the gestures classified are all non-static gestures which lend perfectly to UI control. Extra gestures can be added such as finger twirls or letter drawing which could be mapped to different functions on the car entertainment system and all that would be needed to add these gestures is a small set of data samples. As was mentioned in the introduction, privacy is an issue if small cameras in the dashboard are used to classify these gestures and if the technology becomes the norm in every car the problem becomes exponentially more common. This is the projects strength in that the mmWave sensor is small enough to be fitted within a standard car dashboard and can perform the classification of all the normal gestures that would be used with a UI and is not affected by light conditions of the car, such as entering a tunnel.

The project has use cases in hardware manufacturers such as Microsoft or Lenovo where gesture control in some products within their hardware line up would be a boon. Tablets and laptops can have gesture control implemented so that users could control their device while watching films etc. while sitting further away from the device. It can become more common place for handicapped people to be able to interface with the device more effectively in conjunction with other accessibility features. While most laptops and tablets come with a camera pre-fitted it again raises a large issue with privacy as to implement gesture control

with a normal camera solution the camera would realistically need to be active at all times which raises a serious privacy concern. Car entertainment systems are locked down proprietary systems of which the security vulnerabilities aren't as pronounced. Whereas a laptop can be easily infected which if the webcam is always on creates a high chance of the privacy of users being violated by bad actors.

Nursing and care homes also benefit from this research as while the gestures trained in this project specifically don't translate to their use case it can be trained with a different set of gesture such full body gestures, falling over etc. Since all of these homes use monitoring systems of some variation, they can benefit from a sensor that can capture the full scene of a room such as a camera but be able to not violate the privacy of the occupant. A mmWave sensor can provide this by allowing detection of a gesture such as falling and be able to inform the staff that a patient has fallen without needing to record the patient via camera. This also reduces the possible blind spots that will be present in the homes such as bathrooms as these are sensitive places where cameras are too invasive to use even in normal circumstances. RF sensing using mmWave sensors would allow for fall detection systems to be used in bathrooms to improve patient care and safety.

4. Background Theory

4.1. Millimetre Wave Radar Sensors

Millimetre wave radar sensors are a special type of frequency modulated continuous wave radar sensors that operate at very high frequencies of over 60GHz which causes them to transmit signals with wavelengths in the millimetre range; lower than 5mm. It is this short wavelength that gives them a great advantage over other types of radar sensors as they can resolve different objects with high accuracy even when many are in close proximity. They detect these objects by continuously transmitting electromagnetic wave signals from the onboard antenna which, when coming into contact with an object, will reflect back to the receiving antenna allowing the velocity, range, etc of it to be found [18]. While the transmitted wave the sensor sends out is continuous it differs from the standard pulsed-radar systems by modulating the outputted signal in frequency and phase. FMCW radar is more advantageous than non-modulated systems as the modulation allows for the range to be determined, while non-modulated systems lack the timing marks necessary in order to calculate the range from the sensor.

Fundamentally radar systems operate on the continuous transmission of an electromagnetic waveform, in FMCW this signal is known as a chirp due to its frequency increasing linearly with time. These chirps are firstly generated using a synthesizer which is used as the transmitted signal in order to detect an object and a chirp(s) will be received back from an object(s) in front of the sensor. These two waves are sinusoidal and are summed by a mixer in order to create an intermittent output frequency [18][19]:

$$x_1 = \sin(\omega_1 t + \varphi_1) \quad (1)$$

$$x_2 = \sin(\omega_2 t + \varphi_2) \quad (2)$$

Where x_1 represents the transmitted chirp and x_2 the received chirp back from the object which create an intermediate output frequency signal:

$$x_{if} = \sin [(\omega_1 - \omega_2)t + (\varphi_1 + \varphi_2)] \quad (3)$$

This intermediate frequency (IF) output signal has an instantaneous frequency that is the same as the instantaneous frequency difference the two input signals, this frequency is also constant and comprises of a signal tone. The resulting final IF equation is as follows:

$$x_{if} = \sin (2\pi f_o t + \varphi_o) \quad (4)$$

Where f_o is the instantaneous output frequency of the new signal and φ_o the initial phase difference of the two chirps, this IF wave is sinusoidal as the frequency is constant. Using the formula for the time taken for the chirp to reach and return from the object:

$$t = \frac{2d}{c} \quad (5)$$

Where t is the time it takes for a chirp to be sent and received, d the distance from the object to the sensor, and c the speed of light. The initial phase difference φ_o is expressed mathematically as:

$$\varphi_o = 2\pi f_c t \quad (6)$$

Here the initial frequency of the chirp f_c represents the starting frequency that the two chirps overlap which is the point at which the IF signal is valid. Inserting the equation for the time taken for a propagate fully (5) into (6) and also the equation for the wavelength of a wave results in:

$$\varphi_o = \frac{4\pi d}{\lambda} \quad (7)$$

This initial phase difference can then be used to find the distance of the object from the sensor. This IF signal can be processed using Fourier transforms in order to resolve multiple tones should the sensor detect multiple objects. Here the peaks of the transformed wave represent the different detected objects, allowing the sensor to also detect their information as well.

This Fourier transform is the basis for most information extraction from the mmWave sensor as fast Fourier transforms (FFT) can be then further performed on the output of the range-FFT in order to gather the velocity of the objects [19]. This second FFT is called the Doppler-FFT and is applied to the range-FFT output which has itself been performed over a chirp frame, which is an N set of equally spaced chirps from the sensor. This chirp frame is used so the velocity of 2 objects at the same distance can be found, as if two objects are at the same distance from the sensor the range-FFT will show these two objects as the same peak and so using the above formulas rearranged for velocity won't work. Using the chirp frame N number of peaks are ranged-FFT'd which will result in a N set of peaks each located on the same point on the frequency axis. The difference being that each peak at this point will have a different phase so when the Doppler-FFT is performed on these peaks they are resolved as two different peaks and so can now be inferred as different objects [18].

The more modern CMOS implementations of mmWave sensors like the one used for this project have all the FFT processing integrated onto the microchip itself which greatly improves the useability of the sensors. **Figure 2** below highlights the processing and operation of the mmWave sensor that is used for this project.

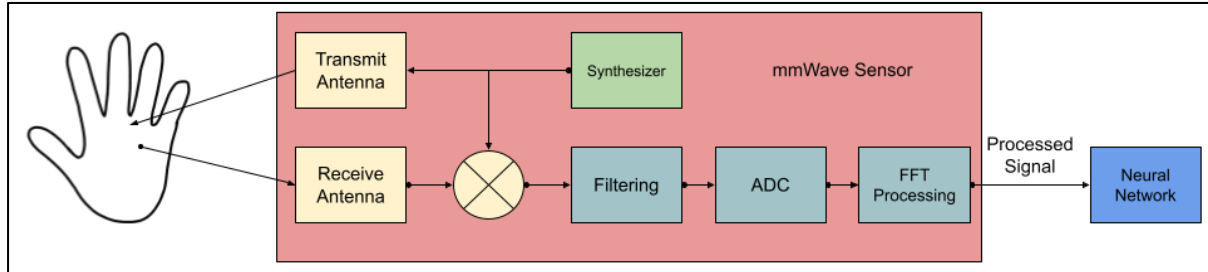


Figure 2: Block diagram showing the operation of the IWR6843AOP mmWave sensor and its data processing pipeline.

4.1.2. Constant False Alarm Rate

As with most sensors noise is an issue whether it is background noise of the area the target is in or inherent noise within the circuitry and so algorithms need to be employed to reduce this noise, improving the SNR. With radar sensors this algorithm is known as Constant False Alarm Rate (CFAR) and is applied to the receiving intermediate frequency signal that is created in the mixer as discussed above. This intermediate frequency signal undergoes envelope extraction where this envelope is proportional to the power of the received signal. By using this envelope, usually called a video signal, a threshold on the magnitude of the power can be set to specify a point in which the magnitude can be seen to be too high and so effected by noise to the point of being useless. This threshold in CFAR is set to be variable with the algorithm setting the threshold value depending on the changing background that is in view of the sensor. This is done as the background greatly effects the noise in the signal and a specific value cannot be chosen as when the sensor for example moves from a cluttered room to a spacious one the number of detections my decrease greatly as the threshold would be set high to remove unneeded clutter in the other location. CFAR's strength is that is can be adjusted as needed to the changing environment and improves the accuracy of sensor which is of great importance in the field of gesture recognition [20].

4.2. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of deep learning network that are most commonly used in computer vision and image classification. CNNs can have a lot of variation in the types and amounts of layers used but the convolutional layer is the backbone of the architecture, which uses a kernel of a pre-determined size to convolve over the input matrices in order to extract features; volume convolutions are the most common type. This kernel is what allows such a complex set of data like an image be connected to the hidden layer neurons without a myriad of connections, as the kernel can form a local connection in the image to the next layers neurons. This is combined with keeping the local connection weights fixed for all the neurons in the next layer which reduces the number of parameters again by a large margin [21]. Due to the weight sharing CNNs are invariant to the features positions in the images and so can classify different types of the same image without loss of

accuracy. Feature learning is also greatly helped by having all of the kernel values being set to random values to force the network to learn more effectively. All these factors reduce the amount of processing performed for a large complex task like image classification giving CNNs a great advantages over a more basic deep learning model like Artificial Neural Networks (ANNs).

CNNs also have great flexibility in the possible tweaking that can be done to its parameters in order to improve performance in terms of classification accuracy and also training and classification speed. These options might have trade-offs, but the flexibility allows for any needed changes to fit the classification problem presented. An option to improve the performance is changing the stride length of the kernel which controls how much the kernel moves for each connection. This controls the overlap of the local regions where having a low stride value will mean more parameters are being used and some even used many times if the image size is large. Having a larger stride size causes the number of parameters to be reduced and thereby increasing the performance of the network at the expense of potentially losing accuracy due to lost information. This loss of information is overcome by using padding normally in CNNs a typical choice is zero padding which makes the input a power of two so that the border information in the image isn't lost [22]. Along with padding a non-linearity function is applied to the output to allow the CNN to approximate functions that don't follow linearity. This gradient function is normally the Rectified Linear Unit (ReLU) as it is simpler to understand and also, so the network does not suffer from the vanishing gradient problem when the network becomes too deep. Finally, CNNs usually encompass some form of pooling in order to again reduce the complexity of the outputs to make it easier for other layers. There are a few different types of pooling, max pooling, min pooling, and average pooling and each type of pooling partitions the output matrix into sub regions and then applies the needed function. The function relates to which type of pooling is need so in the case of max pooling the output will be the max value present within that subdivision, min pooling the smallest value, and average will take the average number between the subdivision. The resulting output matrix will be the same size as the chosen pooling size.

The result of these layers is finally passed as input into a fully connected layer which consists of a large number of artificial neurons which are connected to each other in layers to act as computational units that perform sum-of-products operations between given weights [23]. Artificial neuron's use non-linear functions like ReLU discussed above rather than fixed thresholds like in a perceptron. It is the output of this fully connected layer that can be measured to give an indication if the model is learning correctly, and the rest of the layers remain hidden and act as a black box. As mentioned before these neurons have weights and it is the values of these weights that generates a useful output and the method for estimating these weights is through the use of back propagation. Backpropagation adjusts the weights of the neurons depending on the error present in the output when compared to its actual value. This works by feedforwarding a pattern vector through the network and then backpropagating again through the network adjusting the weights of the neurons as needed. This error is calculated using a cost function which normally takes the mean-squared error between the networks output and the desired output. Mathematically CNNs and fully connected layers perform the same computations on the neurons with the only major difference being that CNNs operate with volumes and fully connected layers operate with vectors [23].

4 Design and Method

5.1. Project Design

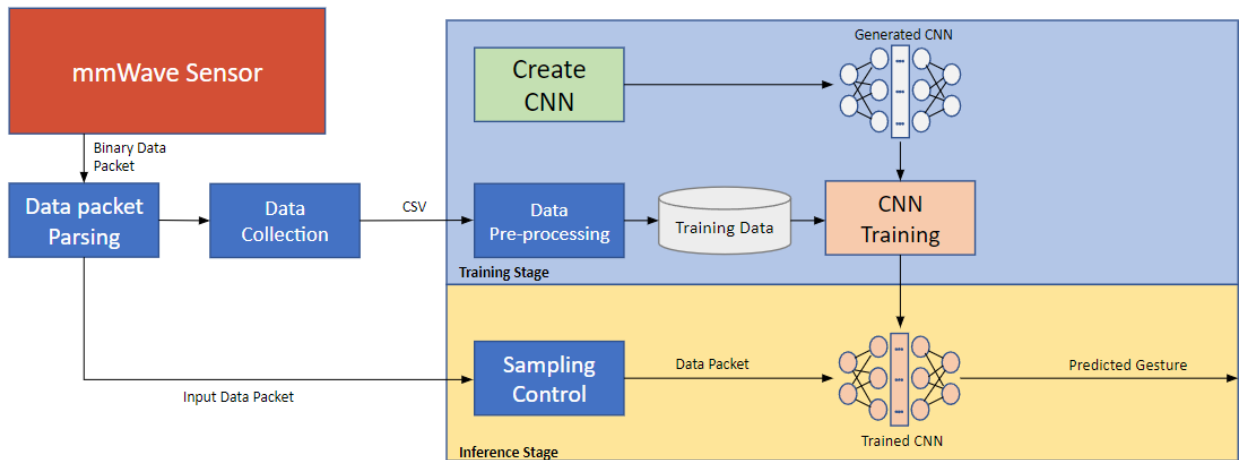


Figure 3: Block diagram overviewing the project operation as a whole.

Figure 3 above shows the overall block diagram for the project which highlights how the various pieces of the project work together. The mmWave sensor is used to collect data for either training or for classification, the former storing the captured data in csv files which are then used as inputs to a convolutional neural network for training. In the case of the latter the incoming data packet is parsed to convert the outputted binary data into decimal point cloud data, this data is pooled by the sampling control until it meets the input size requirements of the CNN upon which it is passed through allowing the neural network to predict a classification. This predicted gesture is then shown to the user via controlling the state of a video player of which the state change corresponds logically to the gesture that was performed, a right swipe moves the video forward for example.

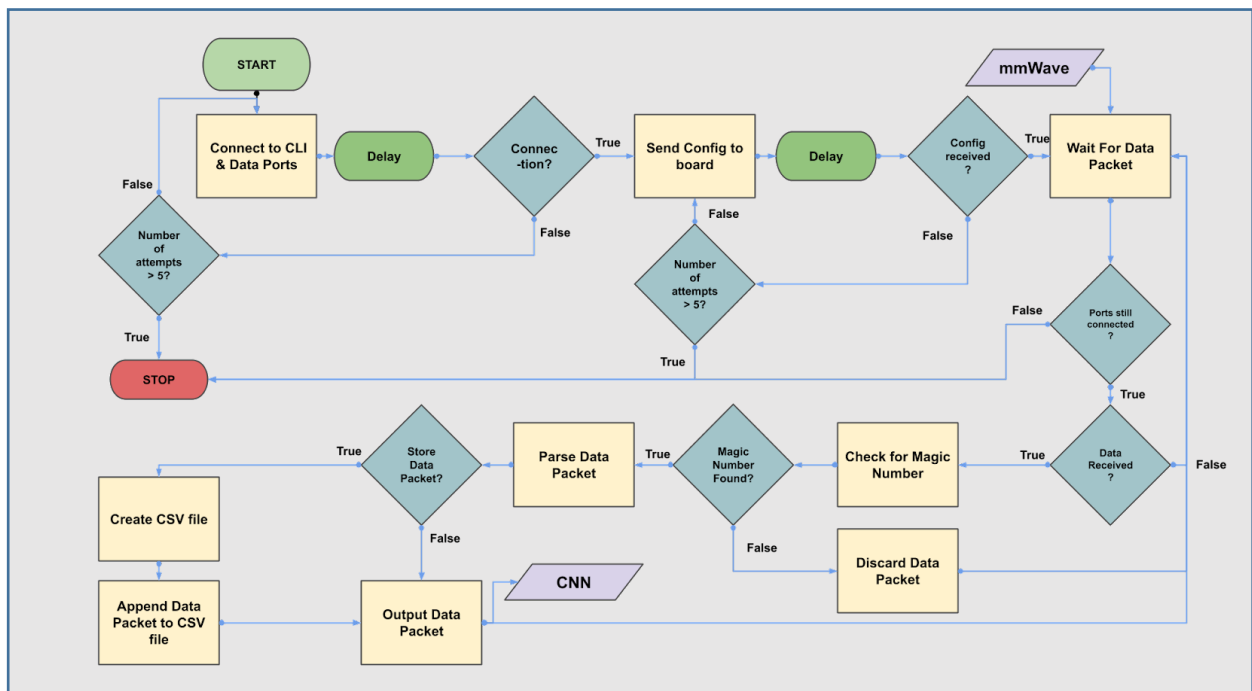


Figure 4: Flowchart showing the data extraction algorithm.

The flowchart in **figure 4** shows the logic behind the data extraction algorithm that is employed to gain useful data from the sensor. The algorithm works by first making a connection between the two needed ports on the mmWave sensor and has a failsafe to stop attempting the connection if it fails after a set number of tries. When a successful connection is made the configuration file is then sent to the sensor which sets the parameters that will be used for that session. These chosen parameters are shown in the table in **figure 5** below. Both the previous two steps have a short delay placed on them after completion in order to avoid the sensor missing the timing window of sending the config file which would cause the sensor to hang. From here the main logic is implemented as this will be the termination point if connection to the sensor is lost. When an output from the sensor is given to the system the algorithm parses the data packet and finds the magic number which is a binary packet header that signifies that the packet has successfully propagated and contains no errors, if it is not found the packet is discarded. This packet is then parsed and depending on the input mode is either stored into a csv file or sent directly to the CNN for classification. This parsed packet contains a large amount of useful data such as range, point cloud, azimuth angle etc., the data chosen to be used as the input to the CNN is the point cloud data and radial velocity (doppler). This is because this data can be effectively used by a **1D convolutional** layer, shown in **figure 5**, to extract the information about these changing point clouds with respect to a sampling window. This in practice can be seen to act like accelerometer data where the coordinates change with respect to the sampling window. These point clouds and doppler values then allow for an easy way to apply this same model to hand gestures as the movement of the hand will cause changing point cloud and doppler values with respect to the chosen sampling window, chosen as 3 in this case due to empirical testing.

Option	Value Set to
Azimuth Resolution	30 + 30 degrees
Range Resolution	0.044m
Maximum Unambiguous Range	9.02m
Maximum Radial Velocity	1m/s
Radial Velocity Resolution	0.13 m/s
Range Detection Threshold	15dB
Doppler Detection Threshold	15dB
Range Peak Grouping	Enabled
Doppler Peak Grouping	Enabled
Static Clutter Removal	Disabled
Angle of Arrival (AoA)	Full FoV
Range FoV	Full FoV
Doppler FoV	Full Fov

Figure 5: Parameters used for the mmWave config file.

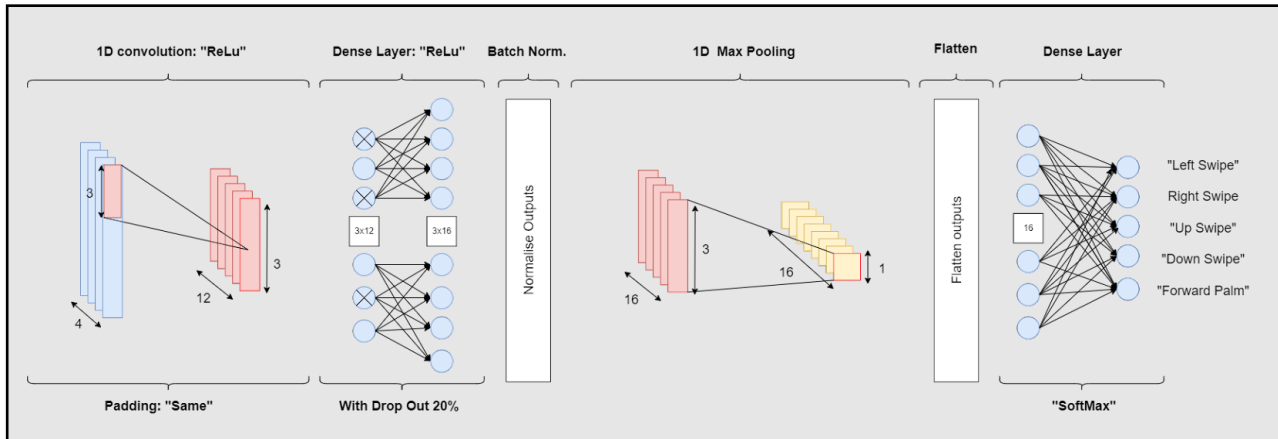


Figure 6: Overview of the CNN architecture.

In order to make use of the data that has been extracted from the mmWave sensors data packet a deep learning network has been created and is shown in **figure 6** above. The network employs a single convolutional layer which was chosen to be one dimensional to allow the network to learn the temporal features of the incoming data. This choice makes logical sense since the chosen extracted data is the point cloud coordinates and radial velocity so when the gesture moves across the detection window these values will change based on how the gesture moves through the window with respect to time. Two fully connected layers are used, one after the convolutional layers and the other connected directly to the output. The final dense layer uses a SoftMax activation function which limits the output between 0 and 1 and so will “fire” one of the output neurons whose probability is the highest. SoftMax was chosen as it provides the probability of each class label which gives a useful probability distribution allowing for an easy way to tell if the network is performing correctly. In the first dense layer a ReLU function is used which reduces the computational costs as ReLU is simple to calculate since the comparison is only between the input and 0 [21]. Drop out is also used in this layer after the convolutional layer as here it has the greatest effect, and the value of dropout was set to 20% causing 20% of the neurons to have their input value set to zero. This dropout is included in order to reduce overfitting of the model on the training data and thereby improve the accuracy of the model on live data. Batch normalisation is utilised after the first dense layer in order to speed up the training of the network by normalising the contributions to that layer. In the centre of the network, between the two dense layers, a max pooling layer is used to create a down sampled feature map which reduces the number of parameters the network needs to learn thereby decreasing the amount of computation. Finally, the output of the pooling layer is flattened before being given to the final dense layer.

5.2. Method

Firstly, the data streaming algorithm outlined above in **figure 4** was coded in Python to create a script using the pymmWave library as a basis for controlling the parsing and the completed code can be seen in **appendix c**. This script was created to allow for either data collection or classification. The script works by converting and storing the incoming data array into a NumPy array which itself, when set for data collection mode, is stored in a CSV file using the Pandas library when the NumPy array reaches a set size; controlled by the user. Empirical tests showed that for one full movement of a gesture a 4x3 NumPy array was generated, this is 3 detections over a time frame of which each detection is an array containing x, y, z, and radial velocity. Tests were then made to check the validity of the collected information by

using measurement points set out across a room where the distance from the sensor was measured in 50cm break points up to 2 metres. Reviewing the detected objects point cloud coordinates showed that the sensor was returning the correct values for the distance the object was from the sensor, this test was then repeated in the z, and y planes as well. As the point cloud and doppler data that is returned from the sensor is related to each object in the scene sometimes the amount of data returned is very large and mostly irrelevant, therefore averages are taken when this occurs. Through empirical testing the parameters in the table in **figure 5** were found to be the most effective and so where the ones used in the data collection and gesture classification, the use of the Range Detection Threshold allows for detections to happen only in 30cm of the sensor. This greatly reduces potential background noise lowering the accuracy, and also setting the Doppler Detection Threshold to 15dB causing detection only of moving objects within this distance further improves the classification.

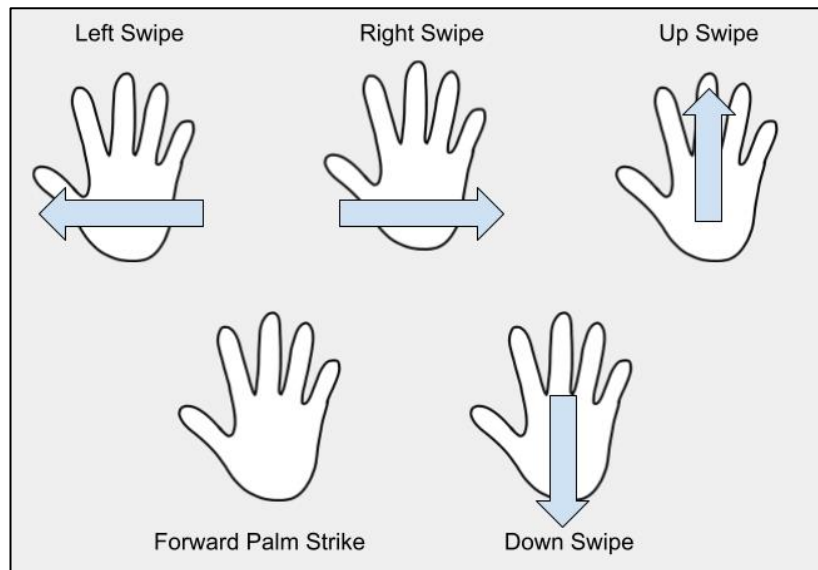


Figure 7: The gestures that the neural network had been trained to classify.

After the data collection algorithm was proven to work, data was then gathered for 5 non-static gestures, these gestures are shown in **figure 7** and are as follows: left swipe, right swipe, up swipe, down swipe, down swipe, and forward palm strike. 1502 pieces of data were gathered for each gesture which with a sampling rate of 3 gives approximately 500 complete gesture movements across the sensor for each gesture. The collected data for each gesture was collected in the same environment and in the same sensor position. The convolutional neural network shown in **figure 6** was then created in Python using the Tensorflow API and the data collected from the sensor used as input data to train the neural network, the results of this can be seen in the results section and code in **appendix d**. The trained model was then imported into the data streaming algorithm and then tested with live input data using the gestures it had been trained on. Once the main functionality was shown to be working another set of data was collected in another environment to test the accuracy of the system on an unseen dataset. This comprised of 835 pieces of data being collected in total resulting in approximately 167 swipes per gesture. Lastly, the data streaming algorithm was then expanded to give the extra functionality to control a video player by taking the highest absolute value of the prediction and having a key be pressed on the keyboard in order to operate the video player.

5 Results

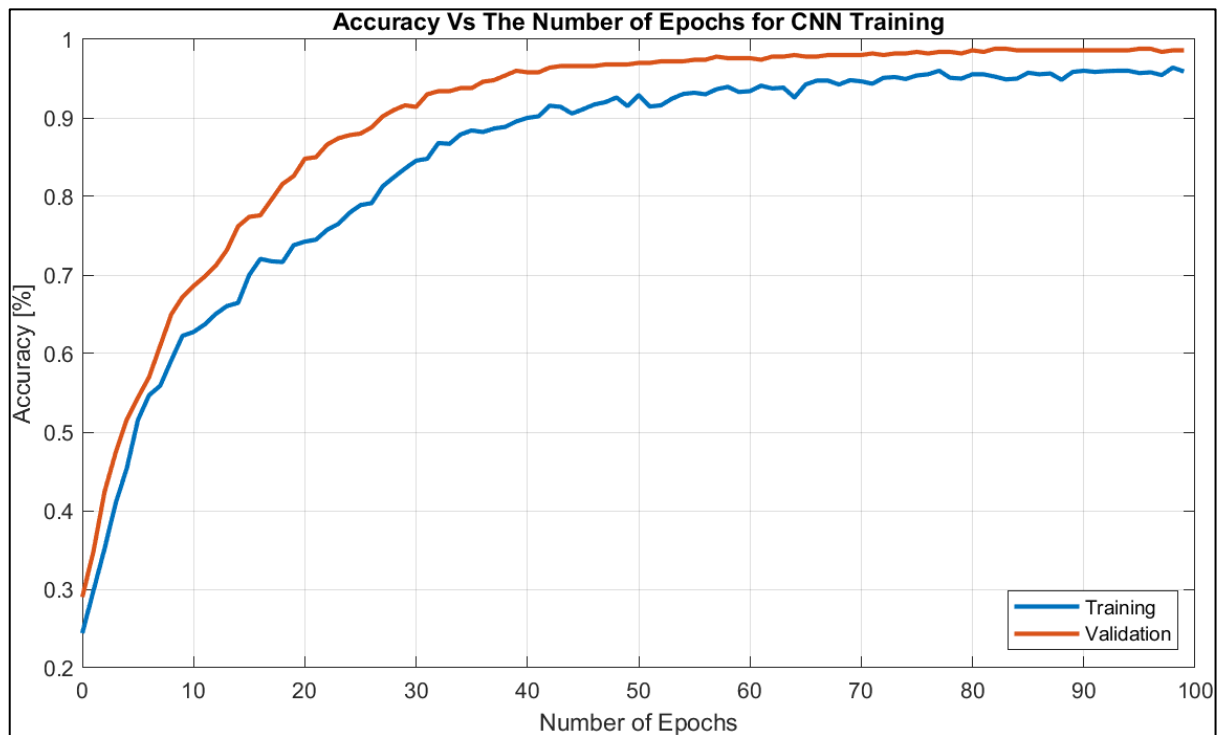


Figure 8: The convolutional neural network training and validation accuracy compared to the number of epochs.

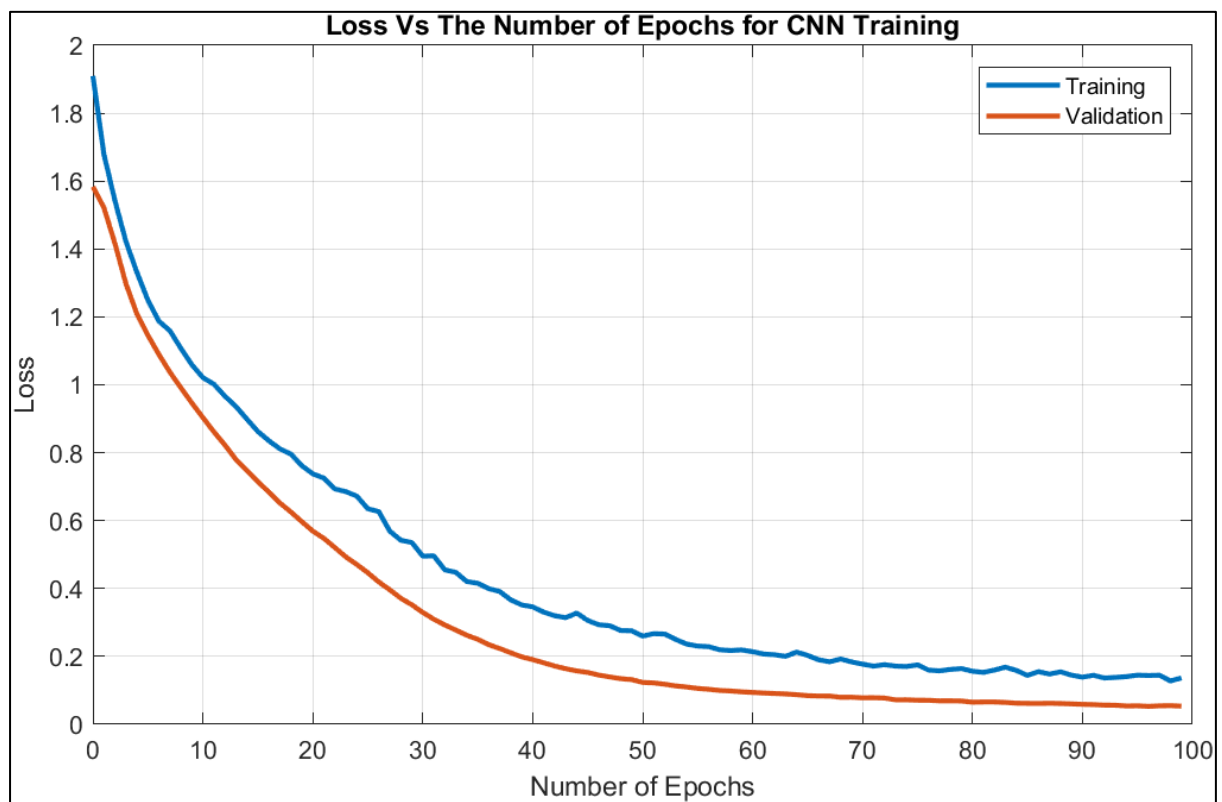


Figure 9: The convolutional neural network training and validation loss compared to the number of epochs.

Figures 8 & 9 show the accuracy and loss for the neural network compared to the number of epochs it is trained over which was 100 using a batch size of 8. The input data was 4 values which comprised of the x, y, and z point cloud coordinates and the radial velocity (doppler) which was sampled at a time step of 3. 2502 pieces of input data were used (and the split between the training and validation data was chosen at 80/20 respectively).

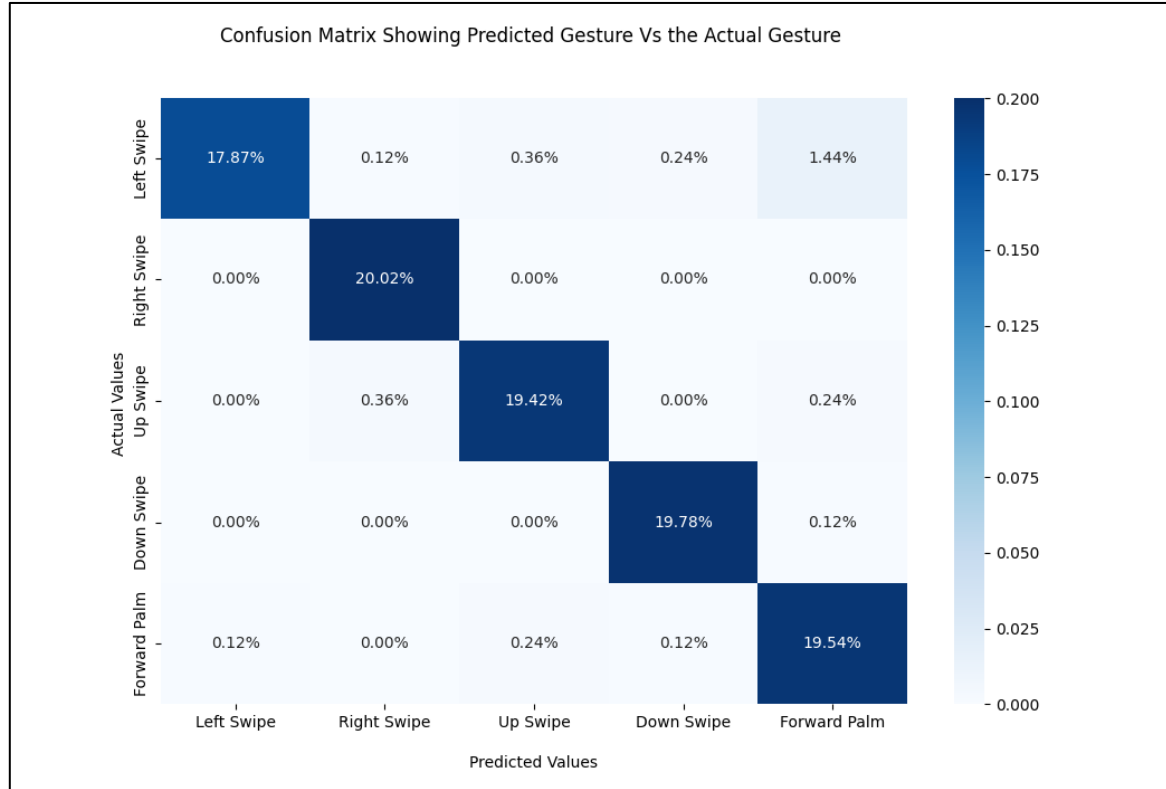


Figure 10: Confusion Matrix showing the test data set's prediction results.

Figure 10 shows the confusion matrix of the neural networks prediction results in terms of percentages which shows how many true positive, false positives, true negative, or true positives there were during the testing of the unseen data set. The test set data that was collected during the method phase equal 2505 pieces of data which corresponds to 835 total gestures performed when sampled at 3 following the empirical testing done before. This gives 167 gestures performed for each class and the average test accuracy to be 96.63% for the 5 gestures.

6 Discussion

7.1. Analysis of Results

The network was trained using the gathered gesture data and the results of the training displayed in **figures 8 & 9** above in the results section. **Figure 8** shows the training and validation accuracy for the network which managed to reach over 95% for both these measures. Achieving such a high training accuracy shows the model has adapted to the complexity of the data it is being fed and so has managed to successfully extract features from the training, showing positive potential for good predictive performance on unseen data; training accuracy does not necessarily correlate with high test accuracy. The validation accuracy is slightly higher than the training accuracy which is most likely due to the use of

drop out in the model which will cause the training set to have a harder time training as when the neurons are disabled the subsequent layers have to operate on incomplete representations of the following layers data. This is also further aided by the fact that the validation data set does not see the drop out layers. There could also be issues with the splitting of the data not being statistically matched and the validation set being comprised of easier to classify examples which would lead to it having a higher accuracy overall. **Figure 9** shows the loss over the epochs for both the training and validation set which is under 0.2 for both, with the validation set having a lower value than the training set. This small loss value shows that the model has a low number of errors and therefore that the model performs well on the data it has been provided. Again, the higher training loss than that of the validation does pose a question of the possibility that the model is overfitting to the data but the difference between the 2 is marginal and with the included dropout doesn't pose a large issue for the current implementation. Its effect could be felt greater if a larger amount of data was used as that could reinforce any overfitting that has already occurred. The data sets all being collected in the same scenario could have also caused this issue as any noise or common errors could be extracted as features and so would cause the model to only be able to classify known sets of data. The chosen number of epochs would also be a possible cause as the data set is relatively small and so having a large number of epochs could cause the model to be pushed into overfitting, the batch size was made smaller to counter act this and to also increase the accuracy as the learning rate of the model is small (0.0001).

Figure 10 shows the more interesting and important data which is a confusion matrix that shows how the model performs with an independent test set that was gathered in a new location and setup compared to the training and validation set. It can be seen that the model predicts the gestures effectively with the lowest percentage being the left swipe which has a total true positive rate 17.87%, which since each gesture is a fifth of the total gives an accuracy of around 89%, $(17.87/20.03) * 100$, with its highest false positive being with the forward palm gesture at 1.44%. This means that out of the 835 total gestures around 11% of the left swipes present in the test data set were incorrectly classified as other gestures, approximately 18. Right swipe has a 100% accuracy score with all gestures in the set being classified correctly as true positives. Up and down swipe have an accuracy above 90% with both having some of their false positives belonging to the forward palm strike. Lastly forward palm strike prediction has the highest false positive with 7.2% of left swipe gestures being wrongly classified as this gesture. This behaviour is exhibited due to the similar features that the data of forward palm strike has with that in left swipe where when performing a left swipe for example sometimes the user uses a slight forward momentum when performing the gesture which causes the x point cloud coordinate to change in the same pattern as the forward palm. Right swipes 100% accuracy is most likely due to the data relating to this gesture being the most information rich allowing for the network to easily discern this gesture from the others. Another thing to note about the confusion matrix in **figure 10**, is that some gestures totals are slightly higher than the ideal split of a 1/5 of the data set for each gesture, this is because there was an issue with reshaping the data to be the needed size of the CNN input that caused the last label and its data to be lost. This means that the 835 total gestures is actually 834 and it becomes fractional when trying to split it amongst 5 different classes in the table and so some values are higher than 20% like left swipe which totals to 20.03%.

Overall, all the results prove positive for this implementation having achieved high test accuracies as shown in **figure 10** which allowed for the project to classify unseen data at a high true positive rate. It even managed to achieve higher accuracies than J. Wu, et al [6] and Y. Ren et al. [9] with a less performance intensive method as the project uses point cloud data

rather than spectrograms. This gives further credence to using the point clouds as this project and D. Salami, et al. [2] showed that higher accuracies with lower overheads are achieved with point cloud data when compared to RDI and RAI spectrograms, which are the common method. Further, the full implementation with the video player was also realised with the project being able to effectively control the video player in real time. As shown in **figure 5** the model itself is relatively simple which shows that mmWave sensors can implement gesture recognition without the need for extremely complex deep learning architectures and can also be trained on relatively small amounts of data. This gives a great merit to mmWave sensors in that they provide very rich sensor data that can easily be exploited in order to improve gesture recognition systems. Improvements can certainly be made but the results achieved in this first implementation provide a proof of concept and platform to build upon.

7.2. Issues that arose

During the course of the projects development there were some minor and large setbacks that caused changes in the design and development of it. Firstly, the initial progress was slow to develop a working algorithm to stream data from the mmWave sensor using Python due to the lack of support from the manufacturer. This was compounded due to the sensor chosen being the newest that Texas Instruments offered at the time and so community efforts in making the previous sensors work with Python could not be directly ported to act as a base to build upon. Not only this but the structure of the configuration files that need to be sent to the board before it can work where changed, and so extra research was needed to study the data packet structure in order to be able to edit this file.

The largest and most significant setback that happened was over two thirds of the way into the project the code base had to be rewritten and the collected gesture data thrown out. This was because the additional Python interface library that was original used for earlier models of the mmWave sensor was causing artifacts to appear in the recorded data. This meant, detections when nothing was in front of the sensor and also the same data values reappearing multiple times even when the objects had changed their states. Attempts to fix the issue went in vain and a few weeks were lost attempting to find a solution to it. During the time that a solution was trying to be found a Python library called pymmWave was discovered that implemented these functions more elegantly and so the project was remade from the ground up using this new library. Progress was much faster this time as the experience gained from the creation of the original code base made it easier to port or remake the old logic and code. Due to this set back the implementation of more complex deep neural networks couldn't be finished and so that would be a future improvement to the project to see how performance can be increased by using more advanced neural networks like CNN-LSTM etc.

7.3. Downsides of the project

While the project worked successfully and implemented the aims that it set out to do there were some problems that could not be fully corrected in the time frame given for the project. One of these issues was the design decision to place a threshold on the radial velocity (doppler) shown in the table in **figure 5**, which is capped to 15dB. The decision was taken in order to reduce the possible noise in the received data and to also remove any potential detections that could arise from people around the sensor, which would cause false classifications. This leaves out a large section of possible gestures that could be classified like hand signs and so in future versions of the system this could be altered. Though the downside is more complex than just altering the velocity threshold as the mmWave sensor is not able

accurately differentiate between two different static gestures easily, as explained in Z. Li, et al [13]. For example, if two fingers were held up and then three fingers the difference between these two gestures is very small with the only difference being an extra finger. Not only that as shown in H. Liu et al. [4] the radar cross section of the human hand is small and so the resolution of the human hand isn't as clear as other materials. This would mean more advanced methods of design would need to be employed if static-gestures were to be classified using this system. Further, mmWave sensors resolve objects and so in the point cloud method the entire hand will most likely be resolved as one object meaning that there will be no possible way to differentiate between two different static gestures. The maximum possible differentiation between objects that the mmWave sensor model that is used for the project can detect is approximately 5mm. This is assuming ideal conditions and in order to achieve near to this accuracy sacrifices must be made in other areas such as the velocity and range resolution.

There is also an issue with performance of the live classification as due to an issue with the implementation of asynchronous behaviour in the code there is sometimes extra data points that get added to the array that is the input to the CNN. This causes the project to sometimes have lower confidence in the gesture it is classifying but the issue is rather inconsistent and a work around was implemented by creating a cache flush with the downside being it is quite resource intensive and so performance suffers on lower powered devices.

7.4. Work Packages and Gantt Chart

The **Appendix a** and **b** show the original Gantt chart and the revised Gantt chart respectively. As can be seen comparing the two charts together the projects plan had to be revised in the second week of February as that is when the issue of the data artifacts discussed above arose. **Appendix b** shows the new plan where the first week was spent researching the new library and also porting over salvageable code into the new Python script. As soon as that was completed data training and testing data sets for the gestures were recollected and the following week the CNN was retrained using these new data sets. At this point in the revised Gantt chart the work packages 1 and 2 in **figure 1** are completed with the data streaming being reimplemented and receiving point cloud outputs, data sets collected, and the CNN being designed with the network returning high accuracies showing the potential for gesture classification. Work package 3 task 3.1 was successfully completed with the system achieving an accuracy of over 95% on 5 gestures which is 15% higher in terms of accuracy than aimed for and even contains more gestures. Task 3.2 however was not realised due to time constraints and the unfortunate time that the issue was found and so was replaced with optimisations to the performance of the live classification which was partially achieved but could still use improvements. The other part of the replacement was the development of a demo to show the usability of the project which was achieved and was then successfully demonstrated at the bench inspection.

7.5. Reasons for choosing a CNN over other potential options

A CNN was chosen over other possible models of deep learning networks as the majority of the literature shows that CNNs have performed the best with the data produced for the mmWave sensor. Not only that but since the project is about classifying gestures there is potential for invariance in the data which could cause incorrect predictions, but a CNN is invariant to this transformation and so is a good candidate for radar data classification. Along with this the CNN can exploit the structural locality of the radar signal which allows for it to

learn the features more effectively. Other choices of network could have been recurrent neural networks (RNN) which work with temporal data which would be a good fit for the project as the point cloud works with temporal data. But RNNs are slow to train and have large memory bandwidths, not only that but they aren't invariant to transformations in the input data and so have the potential to miss classify more often than its convolutional counterpart. Generative adversarial Network's (GAN) are another potential pick, but they have very little literature made about them and also present unstable behaviour.

8. Conclusion

A real time mmWave based gesture classification system using the point cloud coordinate and radial velocity data was developed to classify 5 non-static gestures. A unique data collection algorithm was developed that allowed for easy collection of training and test data, and also provided easy switching to live classification. A CNN architecture was also developed to train a model which was integrated successfully into the data collection algorithm to create a fully functioning system which was able to implement control of a video players UI. The system achieved accuracies above 90% for all classes except one and also achieved an overall average test accuracy of 96.63% on an unseen data set. Overall, the project was a success with majority of the aims being achieved and mmWave sensors being shown to have a great ability to classify gestures. The drawbacks of the project were highlighted, and possible reasons/solutions were given so that future developments can have a path to follow.

9. References

- [1] M. A. Alanazi, A. K. Alhazmi, C. Yakopcic and V. P. Chodavarapu, "Machine Learning Models for Human Fall Detection using Millimeter Wave Sensor," 2021 55th Annual Conference on Information Sciences and Systems (CISS), 2021, pp. 1-5, doi: 10.1109/CISS50987.2021.9400259.
- [2] D. Salami, R. Hasibi, S. Palipana, P. Popovski, T. Michoel and S. Sigg, "Tesla-Rapture: A Lightweight Gesture Recognition System from mmWave Radar Sparse Point Clouds," in IEEE Transactions on Mobile Computing, doi: 10.1109/TMC.2022.3153717.
- [3] T. Yoo, V. L. Le, J. Eon Kim, N. L. Ba, K. -H. Baek and T. T. Kim, "A 137- μ W Area-Efficient Real-Time Gesture Recognition System for Smart Wearable Devices," 2018 IEEE Asian Solid-State Circuits Conference (A-SSCC), 2018, pp. 277-280, doi: 10.1109/ASSCC.2018.8579256.
- [4] H. Liu et al., "M-Gesture: Person-Independent Real-Time In-Air Gesture Recognition Using Commodity Millimeter Wave Radar," in IEEE Internet of Things Journal, vol. 9, no. 5, pp. 3397-3415, 1 March1, 2022, doi: 10.1109/JIOT.2021.3098338.
- [5] J. -T. Yu, L. Yen and P. -H. Tseng, "mmWave Radar-based Hand Gesture Recognition using Range-Angle Image," 2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring), 2020, pp. 1-5, doi: 10.1109/VTC2020-Spring48590.2020.9128573.
- [6] J. Wu, J. Wang, Q. Gao, M. Cheng, M. Pan and H. Zhang, "Toward Robust Device-Free Gesture Recognition Based on Intrinsic Spectrogram of mmWave Signals," in IEEE Internet of Things Journal, doi: 10.1109/JIOT.2022.3165196.
- [7] J. W. Smith, S. Thiagarajan, R. Willis, Y. Makris and M. Torlak, "Improved Static Hand Gesture Classification on Deep Convolutional Neural Networks Using Novel Sterile Training Technique," in IEEE Access, vol. 9, pp. 10893-10902, 2021, doi: 10.1109/ACCESS.2021.3051454.
- [8] S. M. Kwon et al., "Demo: Hands-Free Human Activity Recognition Using Millimeter-Wave Sensors," 2019 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN), 2019, pp. 1-2, doi: 10.1109/DySPAN.2019.8935665.
- [9] Y. Ren et al., "Hand gesture recognition using 802.11ad mmWave sensor in the mobile device," 2021 IEEE Wireless Communications and Networking Conference Workshops (WCNCW), 2021, pp. 1-6, doi: 10.1109/WCNCW49093.2021.9419978.
- [10] G. Zhang, S. Lan, K. Zhang and L. Ye, "Temporal-Range-Doppler Features Interpretation and Recognition of Hand Gestures Using mmW FMCW Radar Sensors," 2020 14th European Conference on Antennas and Propagation (EuCAP), 2020, pp. 1-4, doi: 10.23919/EuCAP48036.2020.9135694.
- [11] P. S. Santhalingam et al., "Expressive ASL Recognition using Millimeter-wave Wireless Signals," 2020 17th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON), 2020,

pp. 1-9, doi: 10.1109/SECON48991.2020.9158441.

[12] S. D. Regani, C. Wu, B. Wang, M. Wu and K. J. R. Liu, "mmWrite: Passive Handwriting Tracking Using a Single Millimeter-Wave Radio," in *IEEE Internet of Things Journal*, vol. 8, no. 17, pp. 13291-13305, 1 Sept.1, 2021, doi: 10.1109/JIOT.2021.3066507.

[13] Z. Li, Z. Lei, A. Yan, E. Solovey and K. Pahlavan, "ThuMouse: A Micro-gesture Cursor Input through mmWave Radar-based Interaction," 2020 *IEEE International Conference on Consumer Electronics (ICCE)*, 2020, pp. 1-9, doi: 10.1109/ICCE46568.2020.9043082.

[14] Y. Sun et al., "Enabling Lightweight Device-Free Wireless Sensing With Network Pruning and Quantization," in *IEEE Sensors Journal*, vol. 22, no. 1, pp. 969-979, 1 Jan.1, 2022, doi: 10.1109/JSEN.2021.3129466.

[15] A. Ninos, J. Hasch and T. Zwick, "Real-Time Macro Gesture Recognition Using Efficient Empirical Feature Extraction With Millimeter-Wave Technology," in *IEEE Sensors Journal*, vol. 21, no. 13, pp. 15161-15170, 1 July1, 2021, doi: 10.1109/JSEN.2021.3072680.

[16] A. Ninos, J. Hasch, M. E. P. Alvarez and T. Zwick, "Synthetic Radar Dataset Generator for Macro-Gesture Recognition," in *IEEE Access*, vol. 9, pp. 76576-76584, 2021, doi: 10.1109/ACCESS.2021.3082843.

[17] J. W. Smith, O. Furxhi and M. Torlak, "An FCNN-Based Super-Resolution mmWave Radar Framework for Contactless Musical Instrument Interface," in *IEEE Transactions on Multimedia*, doi: 10.1109/TMM.2021.3079695.

[18] C. Iovescu, S. Rao, "The fundamentals of millimeter wave radar sensors," T, Texas Instruments, Dallas, TX, USA. 2020. Accessed on: Sept 1st, 2021. [Online]. [Available at]: https://www.ti.com/lit/wp/spyy005a/spyy005a.pdf?ts=1623352330728&ref_url=https%253A%252F%252F

[19] B. van Berlo, A. Elkelany, T. Ozcelebi and N. Meratnia, "Millimeter Wave Sensing: A Review of Application Pipelines and Building Blocks," in *IEEE Sensors Journal*, vol. 21, no. 9, pp. 10332-10368, 1 May1, 2021, doi: 10.1109/JSEN.2021.3057450.

[20] C. Katzlberger, "Object Detection with Automotvie Radar Sensors using CFAR-Algorithms", BSc, Johannes Kepler University Linz, 2018.

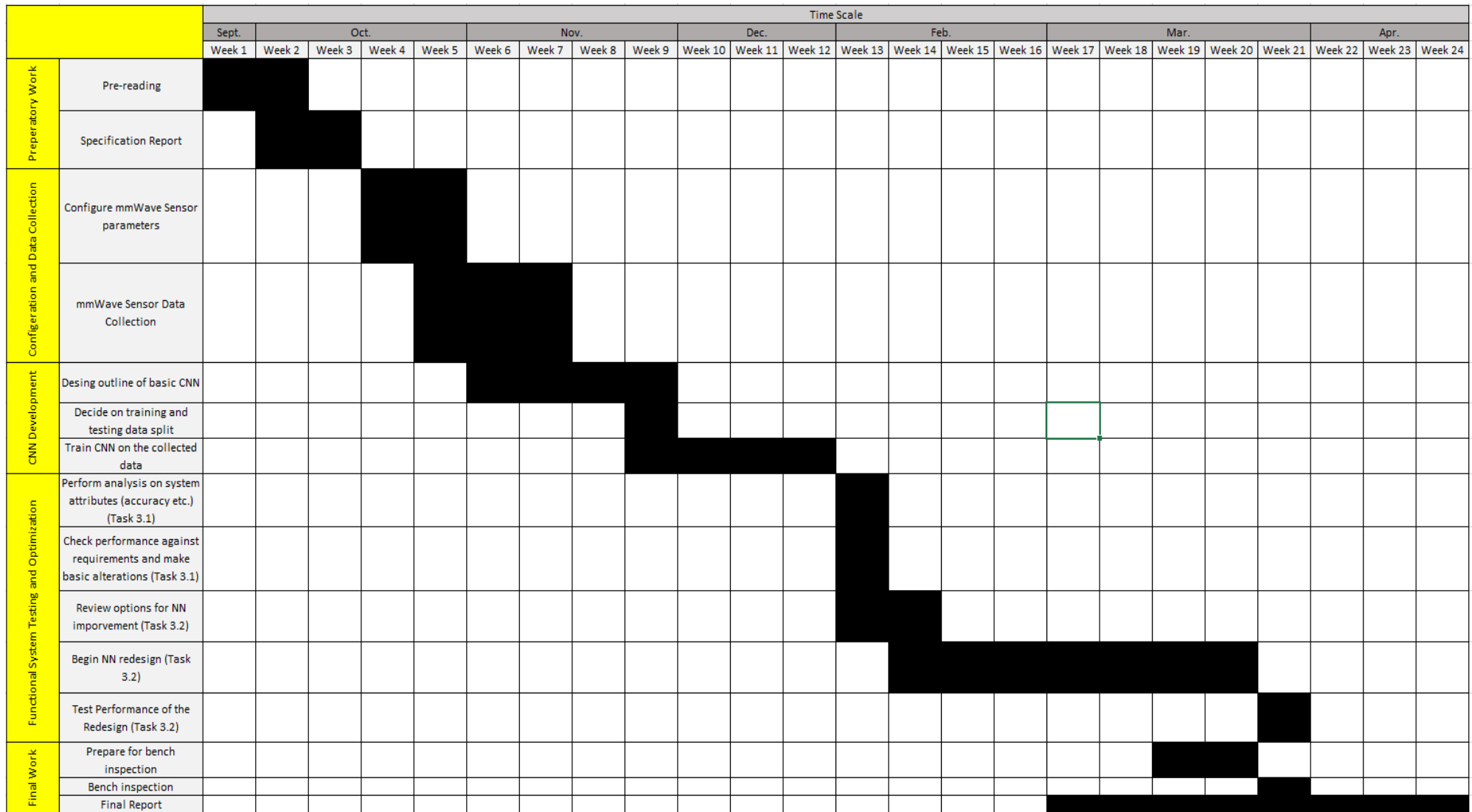
[21] S. Albawi, T. A. Mohammed and S. Al-Zawi, "Understanding of a convolutional neural network," 2017 International Conference on Engineering and Technology (ICET), 2017, pp. 1-6, doi: 10.1109/ICEngTechnol.2017.8308186.

[22] Z. Li, F. Liu, W. Yang, S. Peng and J. Zhou, "A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects," in IEEE Transactions on Neural Networks and Learning Systems, doi: 10.1109/TNNLS.2021.3084827.

[23] R. C. Gonzalez, "Deep Convolutional Neural Networks [Lecture Notes]," in IEEE Signal Processing Magazine, vol. 35, no. 6, pp. 79-87, Nov. 2018, doi: 10.1109/MSP.2018.2842646.

Appendices

a. Original Gantt Chart



b. Revised Gantt Chart

		Time Scale									
		Feb.		Mar.					Apr.		
		Week 15	Week 16	Week 17	Week 18	Week 19	Week 20	Week 21	Week 22	Week 23	Week 24
Research and Porting	Research about the new library										
	Port salvaged code										
Reperform Data Collection	Collect New Training Data For the 5 Gestures										
	Collect Extra Test Set										
CNN Training	Retrain the CNN on the new data										
Functional System Testing and Optimization	Perform analysis on system attributes (accuracy etc.) (Task 3.1)										
	Check performance against requirements and make basic alterations (Task 3.1)										
	Improve live recognition performance (Replaced Task 3.2)										
	Develop Demo To Show Performance (Replaced Task 3.2)										
Final Work	Prepare for bench inspection										
	Bench inspection										
	Final Report										

c. Data Streaming Code

```
import pandas
import os
import keyboard
import time
import numpy as np
import tensorflow as tf
import asyncio

from tensorflow import keras
from pyemmWave.utils import load_cfg_file
from pyemmWave.sensor import Sensor
from pyemmWave.IWR6843AOP import IWR6843AOP
from asyncio import get_event_loop, sleep

print(tf.version.VERSION)

model = keras.models.load_model("ConvNet.h5")

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

""" Selects the mode of operation, either live data classification or collection
    will create the needed file structure and generate any needed csv files """
def user_mode():
    mode = 0
    file_index = 1
    dir_path = ""
    parent_dir = "/data/"
    sensor_data = pandas.DataFrame([], columns=["x", "y", "z", "Doppler"])
    print("Select Mode: ")
    while not mode or mode > 2:
        mode = int(input("[1] - Gesture Classification | [2] - Data Capture \n"))
        print(mode)

    match mode:

        case (1):
            print("[1] Live Data Capture Selected")
            new_gesture_name = "place holder" #Stops errors
            dir_path = parent_dir

        case (2):
            print("[2] Data Capture Selected")
            new_gesture_name = str(input("Name of gesture: "))
            dir_path = os.path.join(parent_dir, new_gesture_name)
            #Check for existing gesture name folder
            if not os.path.exists(dir_path):
                os.mkdir(dir_path) #Create it if needed
                sensor_data.to_csv(f"data/{new_gesture_name}/{new_gesture_name}_{file_index}.csv", index=False)
            else: #Else create a new csv file in the folder to store the data in
                print("Gesture Already Exists, will create new dataset within gesture file")
```

```

        while (os.path.exists(f"data/{new_gesture_name}/{new_gesture_name}_{file_index}.csv")):
            file_index += 1

        sensor_data.to_csv(f"data/{new_gesture_name}/{new_gesture_name}_{file_index}.csv", index=False)

    return new_gesture_name, dir_path, mode, file_index

""" Sets up the connection ports for the mmwave sensor, initialises sensor,
sends config file, etc.

Function taken from the example given in PyMmWave tutorial """
def configure_sensor():

    sensor1 = IWR6843AOP("1", verbose=False)
    file = load_cfg_file("mmwave_config/xwr68xx_AOP_config_10FPS_maxRange_30cm.cfg") #Select the config file to upload

    # Windows port names
    config_connected = sensor1.connect_config('COM7', 115200)
    if not config_connected:
        print("Config connection failed.")
        exit()

    # Your DATA serial port name
    data_connected = sensor1.connect_data('COM8', 921600)
    if not data_connected:
        print("Data connection failed.")
        exit()

    # Will exit after a fixed number of failed attempts
    if not sensor1.send_config(file, max_retries=5):
        print("Sending configuration failed")
        exit()

    # Sets up doppler filtering to remove static noise
    sensor1.configure_filtering(0.01)

    return sensor1

""" Small self explanatory function to control the keyboard upon the detection of certain gestures
Takes input from the print_data function in the form of a string """
def VLC_Control(predicted_gesture):

    match predicted_gesture:

        case ("Forward Palm"):
            keyboard.send("space")

        case ("Left Swipe"):
            keyboard.send("left")

```



```

case("Right Swipe"):
    keyboard.send("right")

case("Up Swipe"):
    for _ in range(10):

        keyboard.send("up")

case("Down Swipe"):
    for _ in range(10):

        keyboard.send("down")

""" Simple class to print data from the sensor, inherits from the default Sensor class
in pywave """
async def print_data(sens: Sensor, gesture_name, directory, mode, index):

    cache_flush = 0
    train_data = []
    live_test_data = []
    data_to_store = []
    encoded_labels = ["Left Swipe", "Right Swipe", "Up Swipe", "Down Swipe", "Forward Palm"]
    encoded_labels = np.array(encoded_labels)

    await asyncio.sleep(1)
    while sens.is_alive():

        sensor_object_data = await sens.get_data() # get_data() -> DopplerPointCloud.
        incoming_data_array = sensor_object_data.get()

        print("\n New scan!")

        # Small check to remove the bug that causes extra data to fill the array.
        if cache_flush: # CAN CAUSE PERFORMANCE LOSS ON SLOW MACHINES
            incoming_data_array = []
            cache_flush = 0
            print("\n Cache flushed!")

        if np.all(np.sum(incoming_data_array)): #If the sensor is returning data check

            if np.size(incoming_data_array) > 4: # Average the data if multiple objects are found

                average_of_data = np.sum(incoming_data_array, axis=0) / incoming_data_array.shape[0]
                average_of_data = average_of_data.reshape(1, 4).astype("float32") # Correct the shape
                train_data.append(average_of_data)

            else: train_data.append(incoming_data_array)

        print(np.size(train_data))

        # If live classification mode was selected

```

```

if mode == 1 and len(train_data) >= 3: #When the amount of data is equal to the input size of the CNN
    live_test_data = np.stack(train_data)
    live_test_data = live_test_data.reshape(-1, 3, 4).astype("float32") #Stack a reshape to input
    prediction = encoded_labels[tf.argmax(model.predict(live_test_data), axis=1)] # Predict a gesture
    confidence = np.max((100 * model.predict(live_test_data))) # Get the max confidence of the gesture
    print(f"\nThe predicted gesture is: {prediction}, and the confidence is: {confidence}")
    VLC_Control(prediction) # Control the keyboard depending on the gesture
    train_data = []
    live_test_data = []
    cache_flush = 1
    time.sleep(2) # Allow the user time to move hand away

# If data collection mode is selected
# When array is the amount of data needed
elif mode == 2 and np.size(train_data) > 2000: # 4000 for 1002 data points, 2000 for 502
    data_to_store = np.vstack(train_data)
    gesture_data = pandas.DataFrame(data_to_store) # Store the array data in csv file
    gesture_data.to_csv(f"{directory}/{gesture_name}_{index}.csv", mode='a', index=False, header=False)
    exit() # close program and shut sensor off.

def main():

    gesture_name, directory, mode, index = user_mode()
    sensor = configure_sensor()

    event_loop = get_event_loop()
    event_loop.create_task(sensor.start_sensor())
    event_loop.create_task(print_data(sensor, gesture_name, directory, mode, index))
    event_loop.run_forever()

if __name__ == "__main__":
    main()

```

d. CNN Code

```
import pandas as pd
import numpy as np
import os

from tensorflow import keras

from keras.layers import Dense, BatchNormalization, Dropout, Conv1D, Flatten, MaxPooling1D

from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3' # Removes GPU tensorflow errors


# Extract the gesture data from the CSV files

left_swipe_train = pd.concat(map(pd.read_csv, ["data/left_swipe/left_swipe_1.csv", "data/left_swipe/left_swipe_2.csv"]), ignore_index=True)
right_swipe_train = pd.concat(map(pd.read_csv, ["data/right_swipe/right_swipe_2.csv", "data/right_swipe/right_swipe_3.csv"]), ignore_index=True)
up_swipe_train = pd.concat(map(pd.read_csv, ["data/up_swipe/up_swipe_1.csv", "data/up_swipe/up_swipe_2.csv"]), ignore_index=True)
down_swipe_train = pd.concat(map(pd.read_csv, ["data/down_swipe/down_swipe_1.csv", "data/down_swipe/down_swipe_2.csv"]), ignore_index=True)
forward_palm_train = pd.concat(map(pd.read_csv, ["data/forward_palm/forward_palm_1.csv", "data/forward_palm/forward_palm_2.csv"]),
ignore_index=True)


x_train = [] # Empty array for the train data to be stored in
my_idx = ["x", "y", "z", "Doppler"] # List of the columns of data to be taken from the CSV files
idx_size = len(my_idx)

sample_size = 3 # Amount of samples taken, also sets the time sample windows of the 1D ConvNet


# left_swipe_train
for i in range(int(len(left_swipe_train)/sample_size)):
    x_train.append(left_swipe_train[sample_size*i:sample_size*i+sample_size][my_idx].values.tolist())


# right_swipe_train
for i in range(int(len(right_swipe_train)/sample_size)):
    x_train.append(right_swipe_train[sample_size*i:sample_size*i+sample_size][my_idx].values.tolist())


# up_swipe_train
for i in range(int(len(up_swipe_train)/sample_size)):
    x_train.append(up_swipe_train[sample_size*i:sample_size*i+sample_size][my_idx].values.tolist())


# down_swipe_train
for i in range(int(len(down_swipe_train)/sample_size)):
    x_train.append(down_swipe_train[sample_size*i:sample_size*i+sample_size][my_idx].values.tolist())


# forward_palm_train
for i in range(int(len(forward_palm_train)/sample_size)):
    x_train.append(forward_palm_train[sample_size*i:sample_size*i+sample_size][my_idx].values.tolist())


# Reshape the array to match the needed input to the 1D convNet
x_train = np.asarray(x_train).reshape(-1, sample_size, idx_size).astype("float32")


# Create the labels for the various gestures
left_swipe_y = np.full(int(len(left_swipe_train)/sample_size), 0)
```

```

right_swipe_y = np.full(int(len(right_swipe_train)/sample_size), 1)
up_swipe_y = np.full(int(len(up_swipe_train)/sample_size), 2)
down_swipe_y = np.full(int(len(down_swipe_train)/sample_size), 3)
forward_palm_y = np.full(int(len(forward_palm_train)/sample_size), 4)

# Combine the labels into a single y_train array
y_train = np.concatenate((left_swipe_y, right_swipe_y, up_swipe_y, down_swipe_y, forward_palm_y))

#Split the data into train and validation split.
x_train, x_val, y_train, y_val = train_test_split(x_train,
                                                y_train,
                                                test_size=0.2,
                                                random_state=42)

# The CNN architecture
model = keras.Sequential()
model.add(Conv1D(12, 3, padding="same", activation="relu", input_shape=(sample_size, idx_size)))
model.add(Dense(16, activation="relu"))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(MaxPooling1D())
model.add(Flatten())
model.add(Dense(5, activation="softmax")) #Final output layer, number of neurons relates to the amount of classes

#Set the models learning rate and metrics
model.compile(
    loss = keras.losses.SparseCategoricalCrossentropy(),
    optimizer = keras.optimizers.Adam(learning_rate=0.0001), # 0.0001 found to work the best
    metrics = ["accuracy"],
)

# Set up tensorboard in order to obtain graphs of the training results
callbacks = [keras.callbacks.TensorBoard(log_dir='logs',
                                         histogram_freq=1,
                                         write_graph=True,
                                         write_images=True,
                                         update_freq='epoch',
                                         profile_batch=2,
                                         embeddings_freq=1)]

# Train the model on the train data over 100 epochs,
model.fit(x_train, y_train, batch_size=8, epochs=100, verbose=2, shuffle=True, validation_data=(x_val, y_val), callbacks = callbacks)

model.save('ConvNet.h5') #Save model
model.summary() #Display the models layer information

```