



School of Computer Science and Engineering

Faculty of Engineering

The University of New South Wales

Deep learning approaches for managing falls in aged care facilities

by

Abanob Tawfik

Thesis submitted as a requirement for the degree of
Bachelor of Engineering in Software Engineering

Submitted: April 2021

Supervisor: Dr. Reza Argha

Student ID: z5075490

Abstract

Falls are one of the leading causes of death and unintentional injuries worldwide. Adults over the age of 65 and individuals in care facilities tend to suffer more falls. Activity recognition using camera data has been significantly researched. However, they raise privacy concerns for subjects. Wearable sensors are another alternative, including pendants and smartwatches. However, this approach will require the individual's responsibility.

In this thesis, I will set out to develop a deep learning anomaly-based fall detection approach that uses mmWave radar sensors to monitor individuals and create a back-end system to collect data from the sensors and detect falls in real-time.

Acknowledgements

The work from this topic has been heavily inspired by my wonderful supervisor Dr. Reza Argha. I would like to personally thank Reza for providing me with this topic, support and feedback along the way, and someone who would guide me down the right path, especially for the deep-learning section. The mentorship and support received was invaluable.

I would also like to thank my assessor, Dr. Nigel Lovell, for providing critical feedback and guidance along the way. Furthermore, I would also like to thank Dr. Michael Stevens for his constant support; he has provided many valuable insights into the topic and created a fantastic atmosphere for weekly meetings. Lastly, I would like to thank Ariyamehr Rezaei; he had helped me set up the code base and showed me how to access and use the data required for the model.

Abbreviations

ADL Activities of Daily Living

AE Autoencoder

VAE Variational Autoencoder

SAE Sparse Autoencoder

GAN Generative Adversarial Network

CNN Convolutional Neural Network

WHO World Health Organization

OSVM One-Class Support Vector Machine

SVM Support Vector Machine

URFD UR Fall Dataset

FDD Fall Detection Dataset

Multicam Multiple Cameras Fall Dataset

mmWave Millimeter Wave

OCC One Class Classifier

OCNN One Class Nearest Neighbour

DSCAE Deep Spatio-Convolutional Autoencoder

TFD Thermal Fall Dataset

PCA Principal Component Analysis

Contents

1	Introduction	1
2	Background	3
2.1	Non-Automated Fall Detection	5
2.1.1	Push-Button Alarm Systems	5
2.2	Automated Fall Detection	6
2.2.1	Dataless Methods	6
2.2.2	Data-driven Methods	7
2.2.2.1	Supervised Learning	8
2.2.2.1.1	Machine Learning Approaches	9
2.2.2.1.2	Convolutional Neural Networks	10
2.2.2.2	Unsupervised Learning	15
2.2.2.3	Semi-Supervised Learning	16
2.2.2.3.1	One Class Classifiers	17
2.2.2.3.2	Generative Models and Autoencoders	18
2.2.3	Data Availability Issues for Fall Detection	22
2.3	mmWave Data	23
2.4	Problem Statement	24

3	Methodology	26
3.1	Data Cleaning and Structuring	27
3.2	Data Pre-Processing	30
3.2.1	Loading Data	31
3.2.2	Generating Frames and Frame Sequencing	31
3.2.3	Oversampling	31
3.2.4	Saving and Loading	32
3.3	Data Splitting	32
3.4	Model Development	34
3.5	Thresholding	36
3.6	Technologies	37
4	Results and Discussion	38
4.1	Results	39
4.1.1	Model Output	39
4.1.2	Metrics	40
4.1.2.1	Accuracy	41
4.1.2.2	F1 Score	41
4.1.2.3	ROC Curve	42
4.1.2.4	Log Loss	43
4.2	Evaluation	44
4.2.1	Outcomes and Analysis	44
4.2.2	Data Issues	45
4.2.2.1	Target Tracking	45
4.2.2.2	Noise Removal	47
4.2.3	What Could of Been Different	47

5 Conclusion	48
5.1 Future Work	48
Bibliography	50
Appendix 1	53

List of Figures

2.1	A typical wearable push-button alarm worn around the neck. (SureSafe, 2021)	5
2.2	Fall detection flow for wearable technology. (Shahzad, 2018)	7
2.3	Labelled data for a simple supervised learning classifier. (GeeksForGeeks, 2020)	8
2.4	CNN structure for the study on human activity recognition. (Ji, 2013) .	11
2.5	CNN structure for the study on fall detection using optical flow. Keep note the input is the stacked horizontal and vertical optical flow images. (Núñez-Marcos, 2017)	12
2.6	Sliding window approach used to maintain motion history. (Núñez-Marcos, 2017)	12
2.7	Camera feed footage for the proposed CNN model. (Núñez-Marcos, 2017)	13
2.8	CNN structure and data extraction used for proposed model. (Jin, 2019)	14
2.9	The output of a variational autoencoder. (Goodfellow, 2015)	18
2.10	The basic structure of a variational autoencoder. (A. Al, 2020)	19
2.11	Visual representation of a sparse autoencoder, light blue nodes are inactive. (Z. Syoya, 2018)	20
2.12	Variational autoencoder with the use of LSTM. (Jin, 2020)	21
2.13	Deep spatio-convolutional autoencoder structure. (Nogas, 2020)	22
2.14	Data captured from mmWave sensor visualised. (Jin, 2020)	24
3.1	Generalised pipeline of thesis project.	27

3.2	Participant data output file structure.	28
3.3	Contents of sample points_cloud.csv file.	29
3.4	Output sample of data cleaning script.	30
3.5	Different possible data splitting techniques, right technique implemented.	33
3.6	Different possible data splitting techniques, right technique implemented	33
3.7	Variational Autoencoder model with all layers involved	35
3.8	Architecture used for this project	37
4.1	Losses from model side by side	39
(a)	ADL losses	39
(b)	Fall losses	39
4.2	Fall losses put ontop of ADL losses	40
4.3	ROC Curve for model	43
4.4	Multiple target id's when only one target in scene.	45
4.5	50,000 points missing from track id file.	45
4.6	Random restart in track id file 71,529 rows in.	46
4.7	Frame 243 missing from the sample.	46
B.1	Correct noise removal behaviour.	53
B.2	Incorrect noise removal behaviour.	54
B.3	Fall frame 1 normal.	54
B.4	Fall frame 2 wrong.	55
B.5	Fall frame 3 normal.	55
B.6	Fall frame 4 wrong.	56
B.7	Fall frame 5 normal.	56
B.8	Fall frame 6 wrong.	57

List of Tables

2.1	Fall detection techniques established in current date.	4
2.2	Result of classification from traditional machine learning approach. (Palmerini, 2020)	10
4.1	Model accuracy accross all activities.	41
4.2	Model F1 Score	42

Chapter 1

Introduction

Falls are the second leading cause of accidental injury deaths worldwide, where each year, an estimated 646,000 individuals die from falls globally [1]. This has led to the need for fall detection-based systems to provide an immediate response; however, current methods in place all come with drawbacks.

Phone-based and smartwatch-based fall detection have risen in popularity. However, in research conducted by *Luque et al.* [2], it was shown that the rate of battery consumption while running complex algorithms was a major drawback. These methods also place responsibility on the individual by making sure they always have their devices on them at all times, and that these devices are charged.

Camera-based fall detection has been another popular choice for fall detection, especially with the prominence of convolutional neural networks (CNN). Unfortunately these methods are supervised learning techniques that require a large amount of data. Real fall data is often unacquariable, as shown in research by *Khan et al.* [3]; falls are rare events, and simulated falls don't encapsulate all aspects of a natural fall.

Privacy concerns also arise when it comes to camera-based fall detection. Data required for camera-based fall detection tend to be a sequence of frames that are stored in video form. This effectively puts individuals under constant surveillance where some third

party is storing their activity data. [5] It was shown that one of the most significant drawbacks of camera-based recognition is the central public opinion that this technology causes a threat to individual privacy.

Currently, the biggest issue in fall detection technology using Machine Learning has been the collection of usable data. Research performed by *Schwickert et al.* [4] has shown that 94% of data gathered for supervised learning techniques and validating models for semi-supervised learning techniques has been done through the use of simulated falls. Amongst this, according to research by *Khan et al.* [3], there is still no standard for fall detection that has been shown to perform consistently well.

Due to these issues, this thesis aims to develop a deep learning-based model to detect human activities and identify falls as anomalies using mmWave sensors while maintaining personal privacy. The model developed during this thesis will also run in real-time to detect falls and alert an individual upon detection. If this model performs at a high specificity in real-time with a low count of false alarms, then this will be taken to facilities in VitalCare for use.

Chapter 2 sets out to explain in further detail; the motivation behind the topic choice, previous work conducted relating to fall detection, the reasoning behind mmWave sensors, and describe the problem statement. Chapter 3 sets out to describe functional requirements, implementation of systems, and technology choice. Chapter 4 sets out to present the results of this research, and evaluate the results in great detail. Finally, Chapter 5 sets out to conclude this report as well as offer suggestions for future work continuing from this project.

Chapter 2

Background

Every year 37.3 million falls occur that are severe enough to require medical attention [1]. Elderly people, in particular, tend to suffer the largest number of fatal falls. [7] Each year, one in four people over the age of 60 experience a fall, and one in three people over the age of 65 experience a fall. Falls remain one of the leading reasons for elderly admissions to the hospital at a rate of 38%, and the second-largest cause being vehicle injuries at a rate of 13%.

The health direct [6] states that a fall usually occurs due to gradual changes in our bodies that make walking difficult or hazards inside the current environments. The changes that occur due to aging tend to be balance issues, weakening muscles that make it harder to walk, degradation in eyesight, and slower reaction times.

Falls have a severe risk involved, the most common injuries being fractures to the thigh and hip at 38%, followed by injuries to the head at 20%. Falls cause more injury deaths in Australia than car collisions. To help visualise the impact of falls, A Western Australian dies every 26 hours, is admitted to a hospital every 19 minutes, and presents to an emergency room every 12 minutes due to falls. According to the CDC [9], amongst people over the age of 65, \$50 billion is spent on medical costs related to non-fatal fall injuries, and \$754 million is spent related to fatal falls. These costs cover hospital fees, doctor checkups, rehabilitation, medical equipment, prescribed drugs, and insurance.

These issues have led to the requirement for fall detection technology to acquire immediate response. If an individual has fallen over and is rendered unconscious, they are at significant risk and are unable in any way to seek help. In the modern-day and age, fall detection is performed through a variety of different techniques, illustrated in Table 2.1.

Fall Detection Method			
Technique	Type	Machine Learning Type	category
push-button alarm system	wearable sensor	N/A	non-automated
mobile phone/smartwatch	wearable sensor	N/A	automated
wearable pendants	wearable sensor	supervised learning	automated
camera based monitoring	ambient sensor	supervised learning	automated
IR monitoring	ambient sensor	semi-supervised learning	automated
mmWave monitoring	ambient sensor	semi-supervised learning	automated

Table 2.1: Fall detection techniques established in current date.

It is also worth noting that fusing information from different types of sensors for fall detection has been a new area of research. In a study performed by *G. Koshmak et al.* [27] multiple types of sensors were fused to determine the effect on accuracy. When combining wearable sensors, accuracy had increased; however, numerous digital devices had to be attached to an individual's body, requiring even more responsibility and inconveniencing the individual.

This study had also investigated fusing wearable sensors and ambient sensors. In most cases, the fusion had increased accuracy compared to each sensor component. However, this approach is still new and actively being researched. Each fusion requires a unique

algorithm and testing.

Research in the field of fall detection can be categorised into two major categories, automated and non-automated. The following section will outline the groundwork for both methods.

2.1 Non-Automated Fall Detection

2.1.1 Push-Button Alarm Systems

Push-button alarm systems work by sending an alarm when an individual presses the button on their supplied wearable technology. Typical push-button alarms are displayed below in Figure 2.1.



Figure 2.1: A typical wearable push-button alarm worn around the neck. (SureSafe, 2021)

Upon falling, the individual would press the button located on the device to signal for an alert. The individual is free to choose the location of the device, as they bear personal responsibility for the usage of the device.

However, these methods have been shown in studies from *Fleming et al.* [10] to be often ineffective at dealing with more dangerous falls. In the year of investigation on 265 fall reports, 82% of falls occurred when the individual was isolated. Of the 60% of people who had fallen, 80% of them were unable to get up afterward, and 30% had laid on the floor for over an hour. A large portion of the study population was using push-button alarms; however, barriers of use arose. If an individual is unconscious after falling, they are unable to activate their alarm. There was also the issue of many individuals who had a device available but were not wearing it during their falls. Often this was due to forgetfulness or the individuals decision to not wear their alarms.

2.2 Automated Fall Detection

Automated fall detection attempts to take the responsibility off the user by automating the process of sending the alarm. This approach can be broken down into two different categories, dataless methods and data-driven methods.

2.2.1 Dataless Methods

Dataless fall detection methods are a branch of fall detection that only requires current motion information to detect falls. Typically wearable sensors are used for dataless methods, and often this approach is seen as the evolution of push-button alarm systems. The sensors work in a similar regard to being worn by the individual; however, algorithms in place can automatically detect falls and alert an emergency contact or carer. Typically these systems are in place on smartphones, smartwatches and pendants. [11] These systems work through motion detection by using an accelerometer and a gyroscope located inside the devices. These sensors are designed to detect everyday activity and the motion of a fall. The accelerometer measures the change in velocity divided by time; this is the technology that changes screen view on a phone when rotated. The gyroscope is a device used to determine the orientation using the earth's gravity, allowing the tracking of rotational information. Custom algorithms are

in place to accurately sense when falls take place. Figure 2.2 shows the general flow of this fall detection methodology.

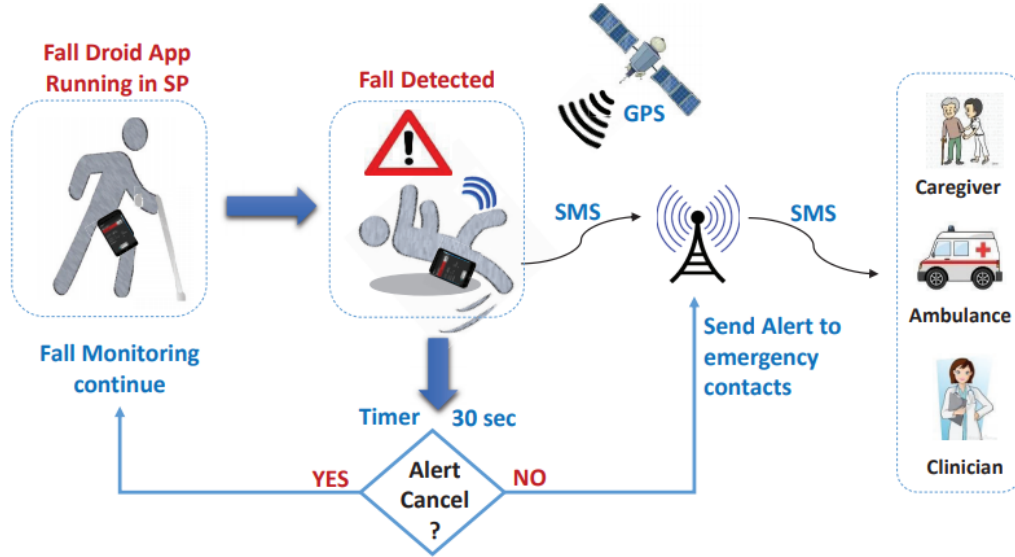


Figure 2.2: Fall detection flow for wearable technology. (Shahzad, 2018)

These fall detection methods are a significant improvement to a push-button alarm system as they take most of the responsibility away from the user. However, they still require the user to always have the devices on them and making sure their device is charged. In particular, the pendants need the user to be in proximity to the connector that sends the alert.

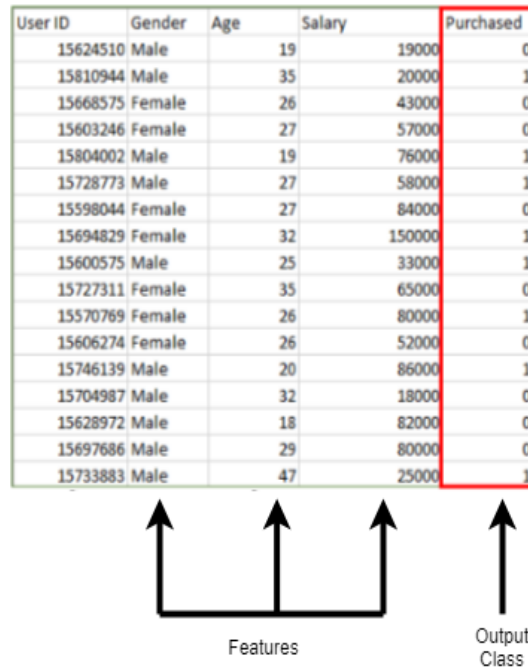
2.2.2 Data-driven Methods

Data-driven fall detection is a method of fall detection that uses previous fall data information to predict whether a fall has occurred or not. Typically machine learning or deep learning is used for these approaches. Machine learning is learning through examples and classifying new samples based on weights learned during training. Deep learning is an extension of machine learning that uses multi-layered neural networks to perform more powerful classification. We can often categorise machine learning and deep learning into three major sections, supervised learning, unsupervised learning, and semi-supervised learning.

The following section will outline the difference between the different categories of machine learning and deep learning. All sections discussed that are not directly classified as machine learning can be assumed to be deep learning.

2.2.2.1 Supervised Learning

Supervised learning is a branch of machine learning that learns how to fit incoming example data to a specific class output. This can also be seen as predicting an output class based on the incoming data. A straightforward example would be a model that indicates whether a person earns a salary over \$50,000 based on the person's occupation, age, and other characteristics. However, this process requires labelled data to train up a model. Labelled data is where each feature is defined for the example data input. Figure 2.3 displays an outline of data used in supervised learning. As shown, a combination of gender, age, and salary is used to predict the output class purchased (whether or not the person has purchased the item).



User ID	Gender	Age	Salary	Purchased
15624510	Male	19	19000	0
15810944	Male	35	20000	1
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	1
15728773	Male	27	58000	1
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	1
15727311	Female	35	65000	0
15570769	Female	26	80000	1
15606274	Female	26	52000	0
15746139	Male	20	86000	1
15704987	Male	32	18000	0
15628972	Male	18	82000	0
15697686	Male	29	80000	0
15733883	Male	47	25000	1

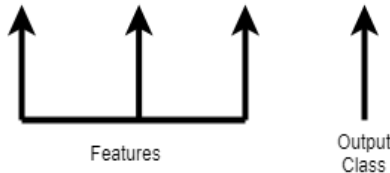


Figure 2.3: Labelled data for a simple supervised learning classifier. (GeeksForGeeks, 2020)

For fall detection, typically, the input would be a video stream passed into a model. Each frame of the video's pixel data is split into three colour channels for RGB, and other times the data would be motion information. The following section will explain the types of models used for supervised learning for fall detection.

2.2.2.1.1 Machine Learning Approaches

Fall detection using traditional machine learning is typically used with wearable sensors. The model on the wearable device attempts to learn the motion of a fall using accelerometers and gyroscopes (similar to section 2.2.1); however, rather than using inbuilt algorithms to do a simple fall check, motion data is used as input for a trained model to detect falls.

In an article written by *Palmerini et al.* [18], sensors worn on the lower back would track motion data with an accelerometer. One hundred and forty-three simulated falls were gathered from 40 subjects within the group's age range, 69.2 ± 12.7 . In addition to fall data, regular activities of daily living (ADL) data was also recorded from 15 subjects to offer a comparison, none of which had performed a fall.

Many models were used to observe the performance and provide comparisons; this included Naïve Bayes, logistic regression, K-Nearest Neighbour (KNN), random forests, and Support Vector Machines (SVM). Motion data was passed into each model, and one-class classification is performed to output whether a fall occurred or not.

The results of this study showed extremely high specificity with these techniques. Table 2.2 outlines the stand-out traditional machine learning techniques and their specificity for this study.

Results of classification		
Model	Specificity	False Alarm Rate hourly
Naïve Bayes	99.1%	1.09
Logistic Regression	99.3%	0.76
KNN	99.2%	0.92
SVM	99.5%	0.56
Random Forests	98.9%	1.32

Table 2.2: Result of classification from traditional machine learning approach. (Palmerini, 2020)

While these models have very high specificity, this was likely due to the models being heavily biased towards detecting falls. This can be seen from the false alarm rates, which are incredibly high. Due to the high rate of false alarms, this system would be unviable as a means of fall detection.

2.2.2.1.2 Convolutional Neural Networks

In the case of fall detection using convolutional neural networks (CNN), many approaches can be taken. The first approach is to take the optical flow of an image to capture motion history and using a 3D CNN combined with transfer learning. Another method would be to use a 3D CNN and stacking multiple frames together as input. Finally, another approach is to use mmWave data in a deep 2D CNN.

In a study performed by *Ji et al.* [14], 3D CNN's were used to detect human activities on TRECVID data which consists of 49 hours of video footage from London airport. The study had focused on recognising three different human actions, putting a cellphone to the ear, pointing, and placing an object somewhere. Alongside having data on these cases, many negative samples were inserted where none of these actions were present in the samples. The model used for this study is shown below in Figure 2.4.

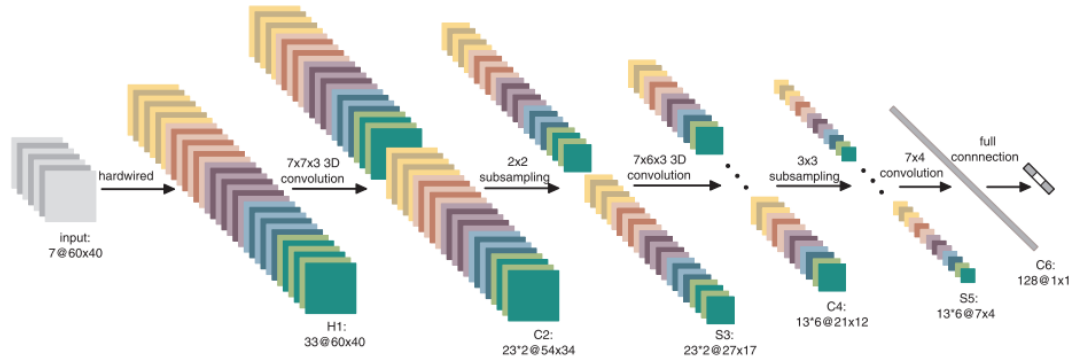


Figure 2.4: CNN structure for the study on human activity recognition. (Ji, 2013)

While this study did not directly detect falls, the extension is that human activity recognition can be extended to also include falls amongst the activities listed and potentially even more. The results from this study had shown that under a false positive rate of 0.1%, the model had on average 71.48% precision, 2.43% recall, and an AUC value of 0.0139. Using a false positive rate of 1%, the model had an average 55.84% precision, 11.47% recall, and an AUC value of 0.6993. This approach had outperformed all other state-of-the-art models such as 2D CNNs and spatial pyramid matching SVMs. However, it is worth mentioning that the CellToEar precision and recall were typically much lower than ObjectPut and Pointing for the results. This could be due to the action of bringing a cellphone to the ear being much more complex than simply putting down an object or pointing at something. It is also worth mentioning that having as high specificity as possible for fall detection is the main priority. Misclassifying a fall as a false negative could be the difference between life and death. This model's low accuracy would be unsuitable for fall detection, especially when other models have been shown to perform with much higher accuracy.

In another study performed by *Núñez-Marcos et al.* [15], 3D CNN's were used to detect falls on simulated fall data through the use of transfer learning. This study had first converted each video frame into an optical flow image. The horizontal and vertical vector fields were separated and passed into the CNN as a stack similar to the previous study. The model used for this study is displayed below in Figure 2.5 and is very deep with many convolving layers. Finally, three fully connected layers are then softmaxed

to predict the binary classification if the example was a fall or not.



Figure 2.5: CNN structure for the study on fall detection using optical flow. Keep note the input is the stacked horizontal and vertical optical flow images. (Núñez-Marcos, 2017)

Transfer learning was used to account for the lack of public fall data. First, the CNN was trained on the Imagenet images dataset. Next, the model was trained on the UCF101 action recognition dataset, using optical flow images for input. Finally, the CNN was frozen during training (no weight updates) to fine-tune the final fully connected layers. A sequence of frames was extracted using a sliding window approach to extract the frames where fall or no fall sequences occur.

A sliding window approach was used to bunch incoming sequences of frames to preserve the motion history within the series of frames. The specific approach taken was to use a step of one which makes sure that no motion history is lost. However, this causes training and classification to be slower. Figure 2.6 displays a visual representation of the sliding window approach. It would be interesting to research further the performance vs. efficiency trade-off between different step sizes for a sliding window approach.

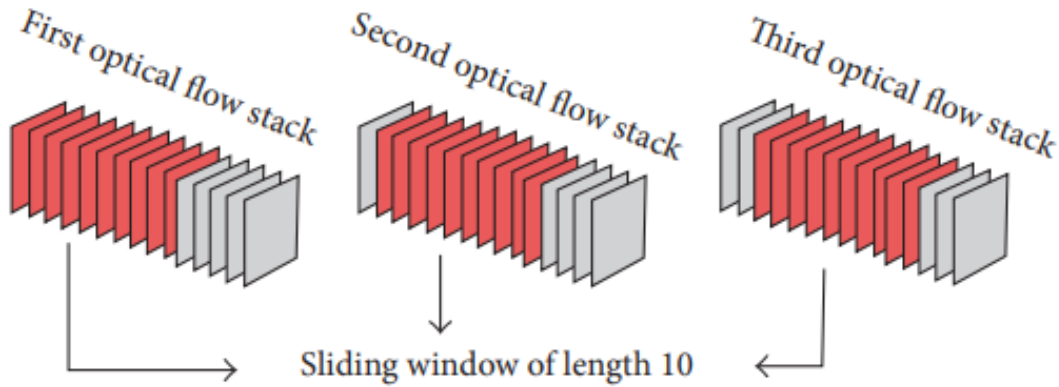


Figure 2.6: Sliding window approach used to maintain motion history. (Núñez-Marcos, 2017)

This study showed that it was challenging to learn how to classify the "fall" class. In particular, attempts were made to improve the importance of the fall class, such as adding bias to the fall class in the learning. While this proved to increase the classification of falls significantly, it did come at the cost of more false positives. Three different fall datasets were used for testing, UR fall dataset (URFD), fall detection dataset (FDD), and multiple cameras fall dataset (Multicam). The model had shown extremely high sensitivity, which is how good the system predicts falls, and the model has shown high specificity. On the URFD dataset, the model had a 100% sensitivity rate and 94.86% specificity rate. On the FDD dataset, the model had a 93.47% sensitivity rate and 97.23% specificity rate. On the Multicam dataset, the model had a 98.07% sensitivity rate and 96.20%. This method produces exceptionally high accuracy and would be a suitable fall detection model. However, it does come with the issue of privacy concerns. Figure 2.7 shows the data involved with this classification type, which flags privacy concerns as this data is stored by a third party, constantly monitoring individuals.

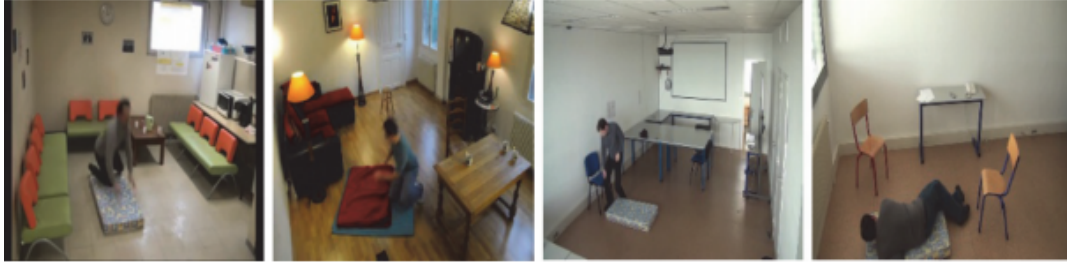


Figure 2.7: Camera feed footage for the proposed CNN model. (Núñez-Marcos, 2017)

Jin et al. [15] performed a study to detect patient activity, primarily fall detection using millimeter wave (mmWave) Radar and Deep CNNs. This developed model was able to distinguish patients and perform activity recognition on all the patients. The model would first detect the multiple patients from the radar using custom-built algorithms. Doppler features for each patient were then passed into a deep CNN. This model is shown below in Figure 2.8

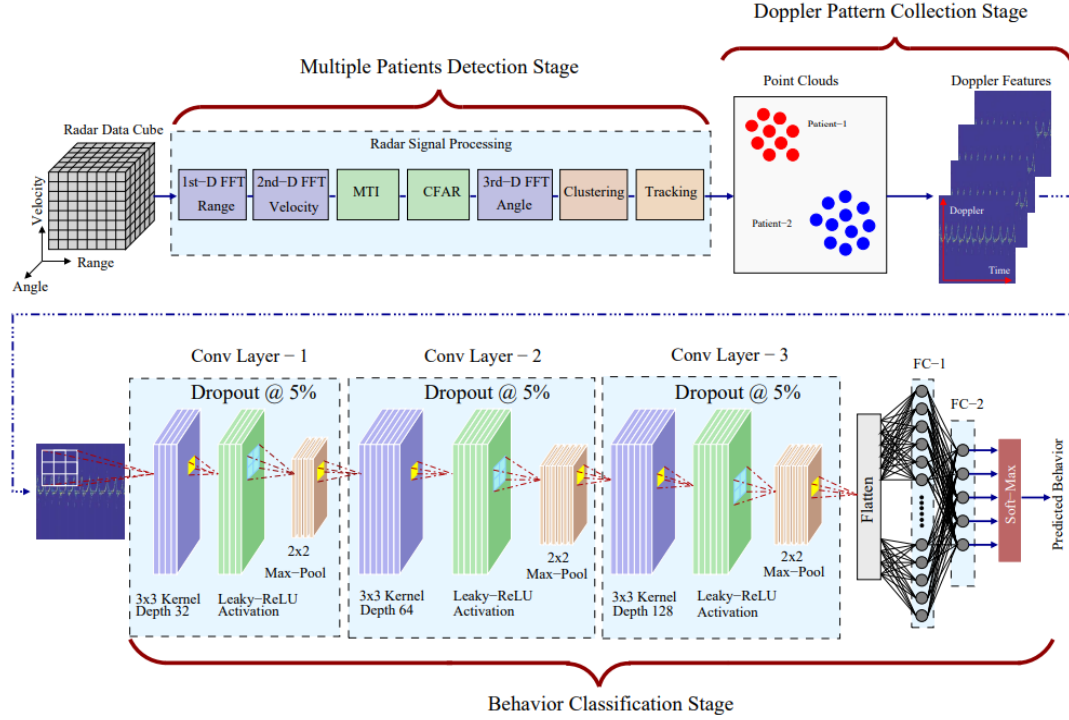


Figure 2.8: CNN structure and data extraction used for proposed model. (Jin, 2019)

The model in this study was able to classify behaviour into the following categories: walking, falling, swinging hand for help, seizure, and restless movement. The data used for this study was simulated falls with only one patient.

One issue that arose with this fall detection method was defining a robust dataset that covers all kinds of situations, as when a fall was in another direction, the fall was misclassified.

The result of this study was based on the number of patients in the frame. When performing behaviour prediction on one patient, specifically for falling, the inference accuracy was 84.49%, adding another user to the frame had significantly decreased the inference accuracy to 66.02%. This is likely due to the increased complexity of the task to classify multiple people separately, and possibly deepening the network or training for more epochs may be needed to improve this accuracy.

A major advantage of this approach was the ability to maintain the privacy of individ-

uals due to the data used for classification. Video footage is not taken from subjects, and subjects are displayed as many points, illustrated in the data collection portion of Figure 2.8.

2.2.2.2 Unsupervised Learning

Unsupervised learning is a branch of machine learning that attempts to discover patterns and information in data to create clusters and grouping. This approach has major benefits for anomaly detection. Typically autoencoders make excellent models for unsupervised anomaly detection.

In a study conducted by *Kiran et al.* [29], many deep networks were used to determine anomalies in video data. UCSD and CUHK-Avenue datasets were used to evaluate the performance of all models and allow comparison. Both the optical flow and raw image sequences were used as input.

This study showed that deeper autoencoder models struggle to detect anomalies through the use of poor reconstruction error in practice. Variational autoencoder (VAE) models performed consistently well and outperformed principal component analysis (PCA) models. Further study is required to understand why stochastic autoencoders perform better. Convolutional long short-term memory (LSTM) models had performed close to the same level as PCA models; however, they were slightly worse than VAE models. 3D convolutional autoencoders had performed slightly better than PCA models when used on optical flow data. These models also allow for the modeling of local motion patterns.

In another study performed by *Karadayi et al.* [30], a hybrid deep learning framework was used to detect anomalies in multivariate spatio-temporal data. This study uses the autoencoder structure, where the encoder is a CNN-based encoder that extracts spatial features of the multivariate data. An LSTM is used as the decoder as LSTM models excel at capturing historical information. The proposed model was compared to state-of-the-art anomaly detection models in terms of performance.

The models were all tested on the buoy dataset, which has many meteorological measurements. Buoy measurements from 2005 have many missing values and or missing features, making this a suitable dataset to test unsupervised learning approaches.

The result of this study had shown that in terms of consistency, the proposed model had performed better than the state-of-the-art models. The proposed model had consistent increases/decreases in anomaly scores from sequence to sequence, whereas other state-of-the-art models had large jumps and drops.

When models were compared in terms of their outliers association with hurricane intensity index, which is a measure of how much correlation exists between outlier scores generated by an algorithm and the ground truth, the hybrid deep learning framework had outperformed all models. The hybrid model had an average score of 0.855, whereas an isolation forest model average score of 0.708 and was the only model that had even come close.

One significant benefit of unsupervised learning approaches is that these models do not require fall data to train and use. While these studies have only shown anomaly detection in a general sense, this can be extended into fall detection, where falls are identified as anomalies. [28] However, unsupervised methods rely on assumptions for the distribution of anomalies and have no prior understanding of anomalies. In most cases, acquiring labelled ADL data and a few samples of labelled anomaly data can lead to improved accuracy.

2.2.2.3 Semi-Supervised Learning

Semi-supervised learning is a branch of machine learning that combines labelled data examples with unlabelled data to group similar data based on available information from the labelled data. This approach is powerful as it combines benefits from both unsupervised learning and supervised learning.

The key benefit to semi-supervised learning is that it removes the need to perform data labelling on most of the data collected, which can be cumbersome, exhausting, and

time-consuming [3].

A typical approach to semi-supervised learning for fall detection is through the use of anomaly detection models. These models are trained on ADL data rather than fall data. When the model encounters unfamiliar data such as falls or anomalies, then the loss function spikes indicating an anomaly. The following section will go into more detail on anomaly detection models.

2.2.2.3.1 One Class Classifiers

One class classifiers (OCC) are a machine learning method that focuses primarily on classifying a single class and treating any other output as the same. Typically these methods employ traditional machine learning techniques such as KNN or SVM.

In a study performed by *Medrano et al.* [22], falls were detected as anomalies through the use of a smartphone. ADL data on ten volunteers were recorded, including eight types of falls. Two types of one-class classifiers were compared to determine which outlier model performed better. The better model was then compared to state-of-the-art supervised learning models.

This study showed that one class nearest neighbour (OCNN) had performed best with an AUC of 0.9554, whereas one-class SVM (OSVM) had an AUC of 0.9439. However, compared to state-of-the-art SVM models, it had performed worse, where SVMs had an AUC of 0.968. This is likely due to the rate of false alarms that occur with an outlier detection model.

In another study performed by *Yu et al.* [23], falls were detected using an online OSVM. Video data frames were used for this study. First, the frame's background would be subtracted, then feature extraction would be performed, and the processed input would be sent into the OSVM for classification. Other algorithms were in place to reduce the rate of false positives, such as checking the duration spent on the floor and the amplitude of the movement.

The result of this study had shown extremely high accuracy with a 100% true positive

rate and a 3% false positive rate. However, the data used for this method puts user privacy at risk since camera feed is used.

2.2.2.3.2 Generative Models and Autoencoders

Generative models is a branch of machine learning that attempts to create or recreate data from given inputs. Two categories for generative models are variational autoencoders (VAE) and generative adversarial networks (GAN).

GAN networks typically would not be used for classification, as the aim of this network is to fool the critic in the actor-critic scenario and convince the critic the output generated is real data.

For the task of fall detection, VAE networks can perform exceptionally well. VAE models provide the ability to reconstruct data from input data and compress data. VAE models aim to reproduce the input as close as possible based on the knowledge learned during training. The output of VAE models is a classification with a probability likelihood, as shown in Figure 2.9. This makes VAE networks exceptional at anomaly detection by using a threshold confidence value to determine if an input is an anomaly.

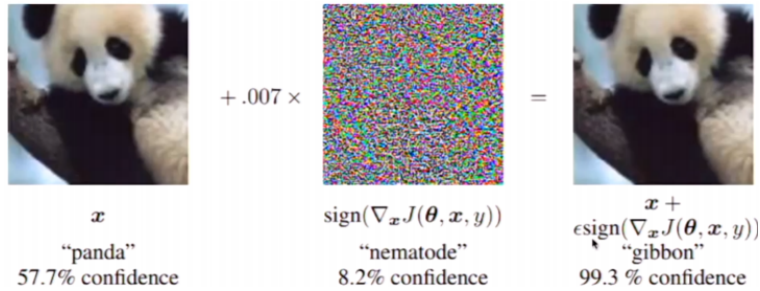


Figure 2.9: The output of a variational autoencoder. (Goodfellow, 2015)

The structure of a typical VAE model is shown below in Figure 2.10. Image data input is fed into the model. The model begins to encode the image by compressing the data to represent the essential features learned during training. At the bottleneck, which is the red box on the figure, the final encoded data representation of the input is located. This specific image went from four features down to two features. The generative section

of the model comes after the bottleneck, where the model attempts to reconstruct the original input image from the encoded input.

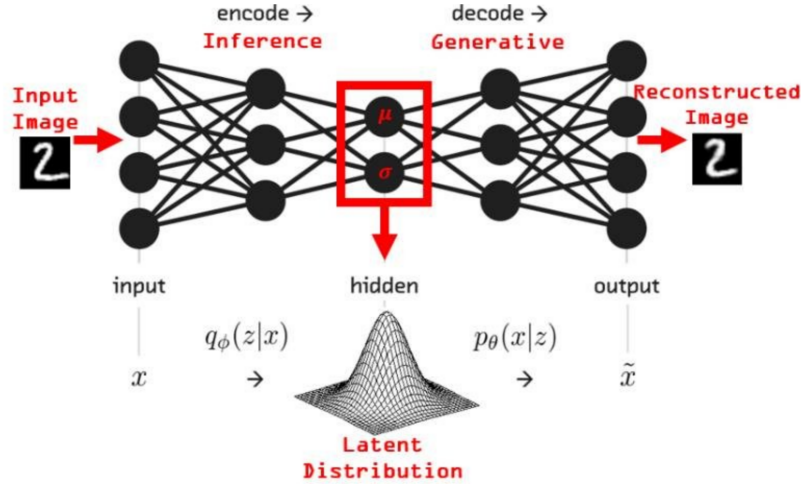


Figure 2.10: The basic structure of a variational autoencoder. (A. Al, 2020)

Sparse autoencoders (SAE) are a type of autoencoder that follows the general autoencoder structure. However, the critical difference is in how the hidden activations work. SAEs can include equal or more nodes at the bottleneck; however, only a small number of hidden units are allowed to be active simultaneously, creating artificial dynamic bottlenecks. This sparsity constraint forces the model to respond to unique statistical features of the training data, where nodes specialise in categorising particular features. One key advantage of this type of autoencoder is that it has improved performance on classification tasks. Figure 2.11 displays a visual representation of an SAE.

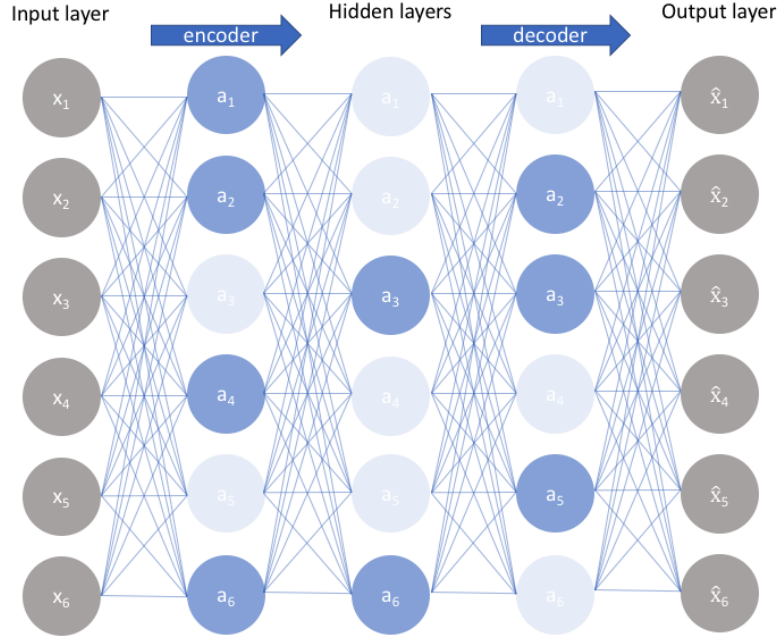


Figure 2.11: Visual representation of a sparse autoencoder, light blue nodes are inactive. (Z. Syoya, 2018)

In a study performed by *Jin et al.* [21], VAE networks alongside LSTM networks were used to detect falls using mmWave radar data. The study had first processed mmWave data on ADL to capture doppler features and centroid data for the input frame. This data would then be input fed into a hybrid model that modifies the typical bottleneck of a VAE model.

Rather than employing a typical bottleneck with hidden layers, a sequence to sequence model is used instead to capture history motion data. The model used is shown in Figure 2.12.

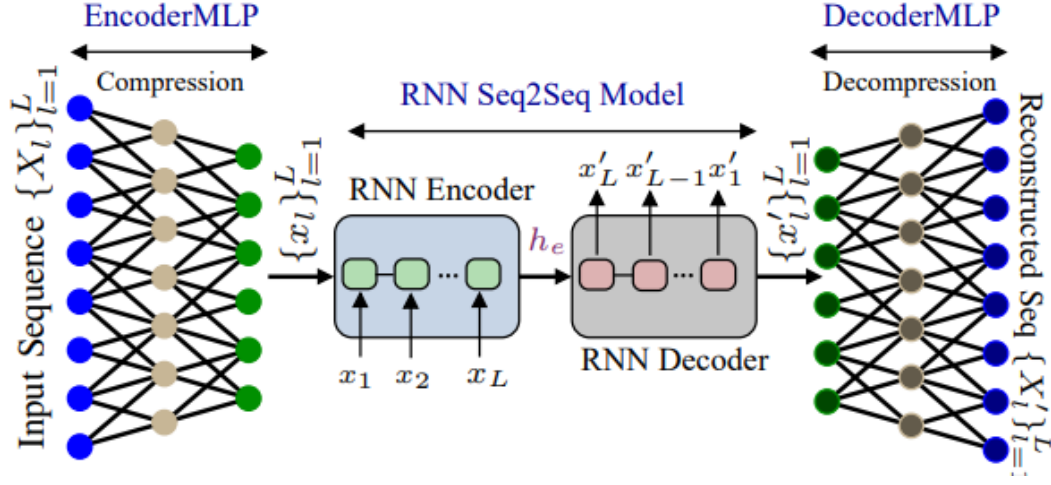


Figure 2.12: Variational autoencoder with the use of LSTM. (Jin, 2020)

The model would output the likelihood of the image reconstruction, and when the model output was below a threshold value, this would indicate an anomaly had occurred. Centroid information is used to investigate anomalies further and reduce false-positive rates. When the centroid height rapidly decreases, alongside detecting an anomaly, this would indicate a fall has occurred.

To test the model, 50 simulated falls alongside negative samples were created and passed into the model for classification. The result of this study had shown that it had achieved a specificity of 98% (49/50), and only two false alarms were triggered amongst the negative samples. This method proves extremely promising as it maintains the subjects' privacy while also providing high specificity. However, this method has only been tested on simulated falls, which often cannot represent an actual fall situation. The model has not been tested or designed to work in a real-time environment.

In another study performed by *Nogas et al.* [24], a deep spatio-convolutional autoencoder (DSCAE) was used to detect falls as anomalies. Video frames stacked together were used as input to the network, where 3D convolutions take place for encoding inputs. Three convolutional layers are used to encode the $8 \times 64 \times 64 \times 1$ data into $2 \times 8 \times 8 \times 8$ data. Encoded inputs are then up-sampled and sent through further convolutional layers to reconstruct the encoded inputs. Figure 2.13 illustrates a simplified representation

of the network structure.

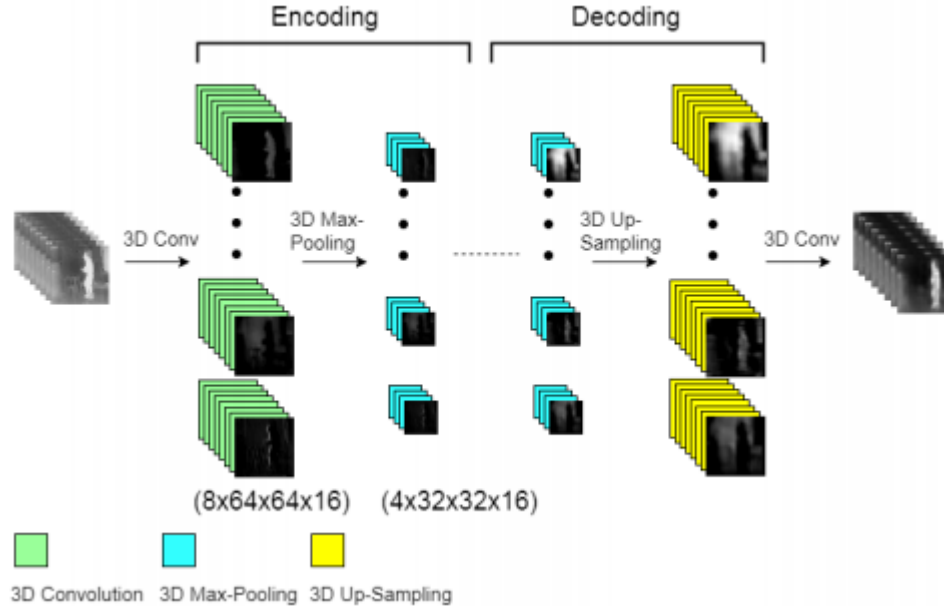


Figure 2.13: Deep spatio-convolutional autoencoder structure. (Nogas, 2020)

The model uses a novel threshold system to determine anomalies. Rather than computing an anomaly score on a frame by frame basis, the model calculates an anomaly score for each frame alongside reconstruction error information in adjacent video frame stacks. This was shown to perform better in real-world scenarios.

Three different public datasets were used to test the model, thermal fall dataset (TFD), URFD, and the SDU dataset. The result of this study shows that this model outperforms other autoencoder methods and other machine learning approaches. The AUC of the model on the three datasets averaged 0.91, whereas a traditional deep autoencoder approach had an average AUC of 0.78 and a K-nearest neighbour approach had an AUC of 0.58.

2.2.3 Data Availability Issues for Fall Detection

In the field of fall detection using machine learning, acquiring fall data has been a significant challenge. In a study conducted by *Schwickert et al.* [4], it was shown that

94% of studies use simulated falls for training models, validating models, or both. Fall data is rare, as falls are rare events, and simulating a fall cannot necessarily encapsulate all aspects of a natural fall.

For supervised learning techniques, this can be problematic for a few reasons. If a model has only been trained to work on simulated falls, it may overfit during training and perform poorly in a real-world event. Falls from healthy young subjects are not the same as a fall from an elderly person [3]. Another issue is that supervised learning methods require a large amount of data for training to ensure the model does not overfit the training dataset; this is especially true in more complex models that must learn more weights. The data provided must also be evenly distributed in output classes to ensure the model is not biased towards a specific output class. Due to the rarity of falls, it would be unrealistic to train a supervised learning model with real fall data due to the lack of real falls. This issue is less problematic for anomaly detection methods as they primarily use ADL data. However, when it comes to testing the models, often simulated falls are used.

2.3 mmWave Data

Millimeter-wave radar is a new sensing technology that allows for the detection of objects. These radars provide information regarding the range, velocity, and angle of these objects. mmWave is a contactless technology that operates between 30GHz and 300GHZ. The use of small wavelengths allows sub-mm range accuracy and can penetrate through materials such as drywall, plastic, or clothing [25, 26]. Another critical feature of mmWave sensing technology is the ability to preserve user privacy while maintaining motion and positional information. Figure 2.14 shows the processed data from a mmWave radar alongside a camera view. On the left of the image, you can see the person has fallen; however, on the right, we see a point cloud representation of that motion history, abstracting the individual from the motion entirely.

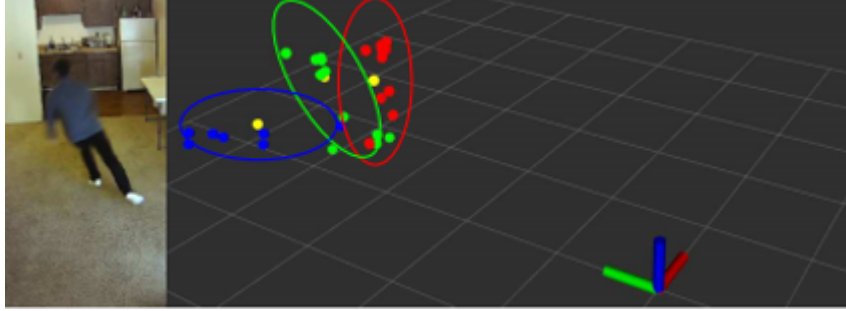


Figure 2.14: Data captured from mmWave sensor visualised. (Jin, 2020)

2.4 Problem Statement

Previous studies have shown the difficulty in detecting falls in real-time. Often, a study would excel in one aspect of fall detection but present drawbacks that cannot be ignored. Supervised learning models would present a high specificity; however, this is done through simulated falls and often overfit when it comes to real-world scenarios. Supervised learning models also present the privacy concern as often camera feed data is used, placing subjects under constant surveillance. Due to the lack of data available, these models cannot be trained with real fall data.

Semi-supervised learning approaches seem to do an excellent job in producing models that obtain high specificity and do not require fall data for training. While these models are almost ideal, they continue to use simulated falls to validate the dataset and are not designed to work in real-time. Another issue discovered in the study was that determining a threshold value that minimises false alarms while maximising the correctly classified true positive case was challenging.

In this thesis, the goal is to create a generative fall detection model that can detect falls as anomalies in real-time while maintaining the subject's privacy and achieving a high specificity on natural falls, with a low rate of false positives. Privacy will be maintained as mmWave radar data will be used (see Figure 2.14). This model will also work 24/7 without placing any responsibility on the subject.

Tensorflow and Keras API will be used to develop the planned model. Edge computers will run locally to collect and process data and pass it onto a pre-trained model to detect falls in real-time.

Chapter 3

Methodology

This chapter aims to outline the work undertaken throughout thesis A, B, and C, and the working code in thesis C.

This thesis aimed to develop a pipeline that is able to:

1. Clean and label raw mmWave data for training.
2. Process raw mmWave data into frame sequences.
3. Data Splitting.
4. Develop a generative model to reconstruct frame sequences.
5. Threshold code to determine if a fall occurred.

Figure 3.1 gives a high-level overview of the project, generalised flow, and pipeline.

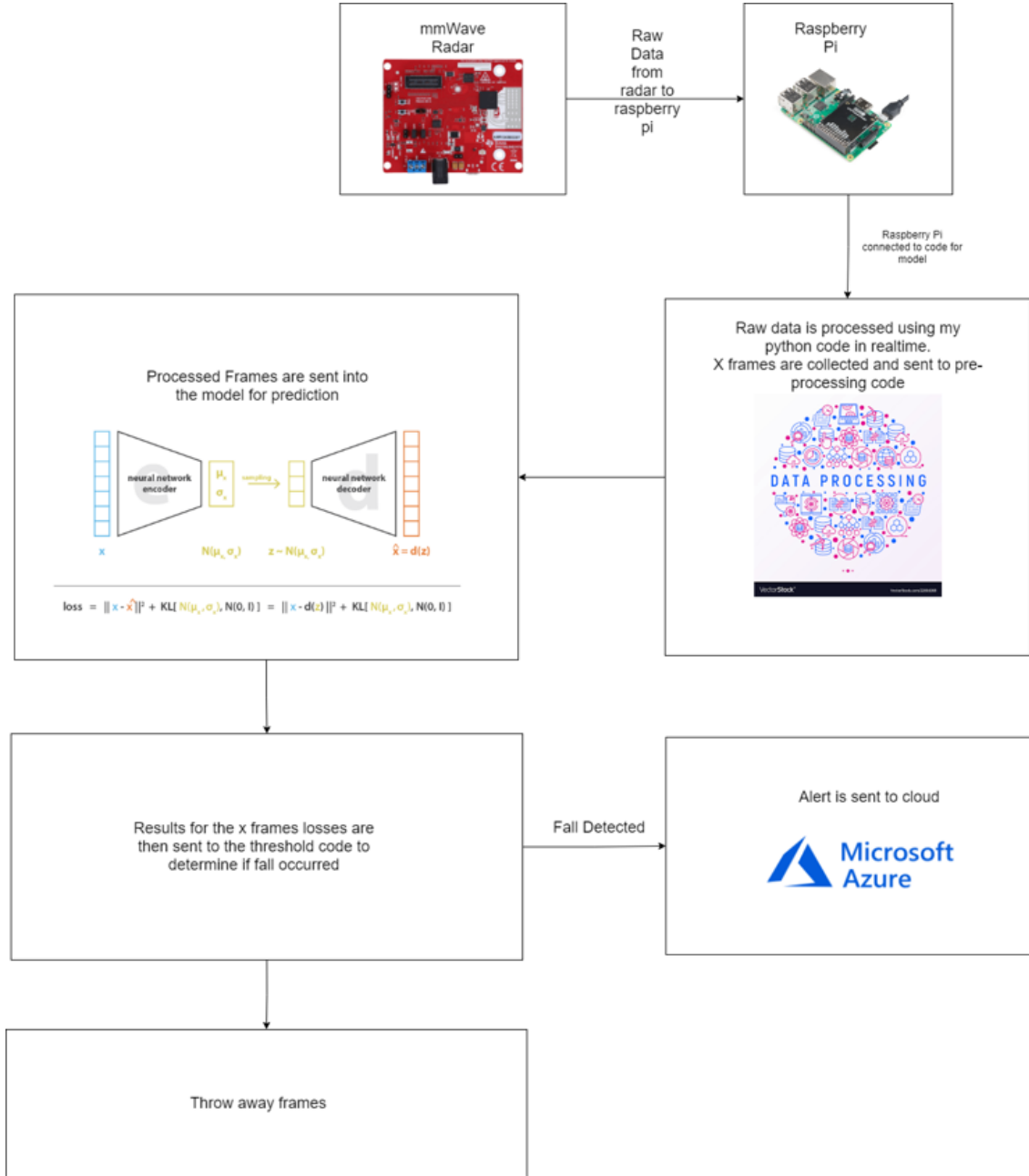


Figure 3.1: Generalised pipeline of thesis project.

3.1 Data Cleaning and Structuring

The aim of data cleaning was to create structured and labelled data out of the raw mmWave sensor output. This process also involved the removal of outliers and invalid

data points obtained from the sensor. Participant data from the sensor arrives as one big point cloud dump containing information on the following:

- Timestamp
- Frame Number
- xyz Coordinates
- Doppler
- SNR
- Range
- Azimuth
- Elevation

Figure 3.2 illustrates a sample subject folder containing all the point cloud information for all activities performed by the subject during that day. Inside of the `points_cloud.csv` file we have the raw sensor data used for constructing frame sequences. Inside the `track_ids.csv` file we have target tracking information associated with each point in the `points_cloud.csv` file. Finally, the `target_list.csv` file contains information about all targets that are found in the `track_ids.csv` file.

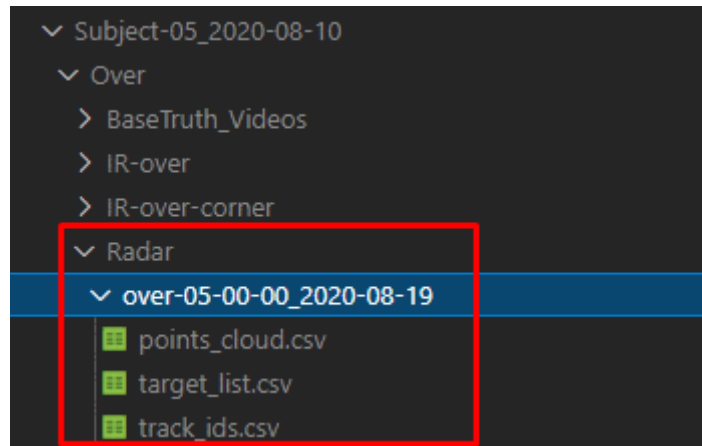


Figure 3.2: Participant data output file structure.

Figure 3.3 shows the content of the points_cloud.csv file in Figure 3.1. As shown below, there is no structure to this data, and there is no way of determining which activity is which based on the data we have based on the data's current form.

The image shows a screenshot of a text editor displaying the contents of a CSV file named 'points_cloud.csv'. The file contains a large number of rows of data, each representing a point in a 3D cloud. The columns are labeled: timestamp, frame_number, x, y, z, doppler, snr, range, azimuth, elevation. The data is unstructured and lacks any clear grouping or headers.

Figure 3.3: Contents of sample points_cloud.csv file.

I resolved the issue of unstructured data by developing a flexible python script that is able to link the raw points_cloud.csv file to the corresponding ground truth file that contains the duration of each activity. Using this information I can calculate how many frames are in each activity and the order in which they occur in. After synchronizing the clocks between the mmWave sensor and the ground truth video camera, I was able to go through the massive points_cloud.csv file and determine which activity had occurred and the number of frames for said activity. I had created sub-samples that only contain the data for the activity. These sub-samples look identical to the original points_cloud.csv in terms of structure. Figures 3.4 demonstrate the result of data cleaning, where the files are labelled based on which directory they are saved in. This powerful script allows for labelling the unstructured data based on activity.

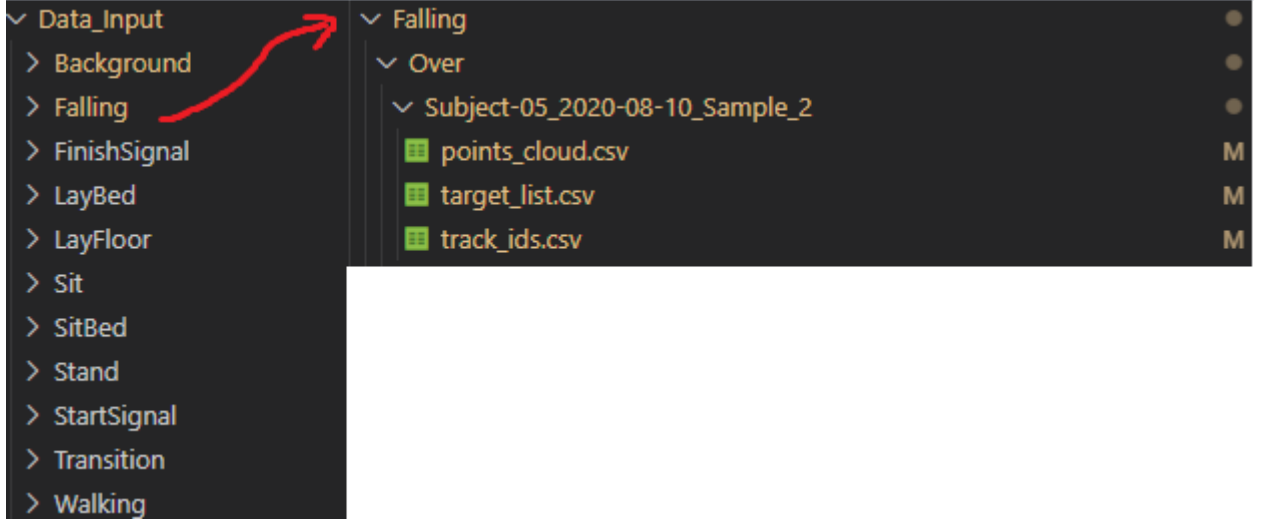


Figure 3.4: Output sample of data cleaning script.

3.2 Data Pre-Processing

For any machine learning project, data pre-processing is the most important step, and correct execution is crucial to acquire meaningful results. The data pre-processing pipeline had the following steps in order to turn the raw mmWave data into frame sequences.

1. Loading all raw data and categorising the data by activity.
2. Generating frames from this data by grouping data points.
3. Generating frame sequences by using a sliding window to move across the frame arrays.
4. Oversampling each frame in the sequence to ensure all frames are of equal length.
5. Saving and loading these frame sequences.

3.2.1 Loading Data

Raw mmWave data is loaded into the program through the use of a map. The map enables the categorization of raw data loaded from points cloud.csv files. Each activity consists of a number of samples and is represented in the program as an array of sample data for map value. The map is of the following structure $\text{Map}<\text{Activity}, [\text{Sample}]>$. The script parses all activities, and for each sample in said activity, the points_cloud.csv file is loaded and stored in the map as a numpy array under the corresponding activity.

3.2.2 Generating Frames and Frame Sequencing

To generate frames out of raw mmWave point data, we go through each sample one at a time and group together points that have the same timestamp using a map. The map is of the following structure $\text{Map}<\text{Frame}, [\text{Point}]>$. Due to overlapping timestamps across samples, the sample name was also combined with the Frame key to generate a unique timestamp for each sample.

Once frames were generated, the next step was to perform frame sequencing. First, the map is flattened into an array. Next, each frame is scanned for invalid points and invalid data. With this array of frames, a sliding window is used with a customisable step size to move across the array and generate frame sequences for each sample. The size of each frame sequence is a customisable parameter set by the user.

3.2.3 Oversampling

Ensuring all frames are of the same size is crucial for a machine learning model, as models only work with static input size. Due to the variable nature of point cloud data, all frames have a different number of points. In order to standardise input into the model, all frames are oversampled to the largest possible frame size. The largest frame size is computed during the frame generation as an optimisation step, constantly updating the maximum, rather than recomputing through a linear search. The oversampling

technique used is based on a paper written by *Jin et al.* [15] where the number of points in a frame are extended, whilst keeping the same distribution.

3.2.4 Saving and Loading

Oversampling the frame sequence concludes the final step of the data pre-processing pipeline. Due to the tedious and complex nature of data pre-processing, checkpointing data that has already been pre-processed saves a tremendous amount of time when running the program. Every sample that had all its frame sequences oversampled is saved into a numpy file. When performing data pre-processing we ignore all samples that have been already saved.

To work with the saved data, a function is created to load all saved samples and store them in a map based on their activities.

3.3 Data Splitting

To ensure that the model can be trained and tested correctly, appropriate data splitting must be performed. The most important aspect when it comes to splitting the dataset is to capture the split percentage across all activities. Initially, I was taking the percentage across the whole dataset from start to finish, resulting in certain splits missing activities. In order to counteract this issue, I had taken the percentage of samples across all activities by traversing the activity map. Figure 3.5 displays a visual representation of taking an 80% split of the dataset. It can be noted that the correct splitting technique that was implemented captures all activities across any split.



Figure 3.5: Different possible data splitting techniques, right technique implemented.

Through the use of this data splitting technique, I was able to construct subsets of the original data in order to perform training, threshold calculation, and threshold evaluation. The following splits were performed:

- Training the model: 80% of non-fall samples
- Computing the threshold: 50% of fall samples, 10% of non fall-samples
- validating threshold and testing: 50% of fall samples, 10% of non fall-samples

Figure 3.6 gives a visual view of the data splitting that was performed.

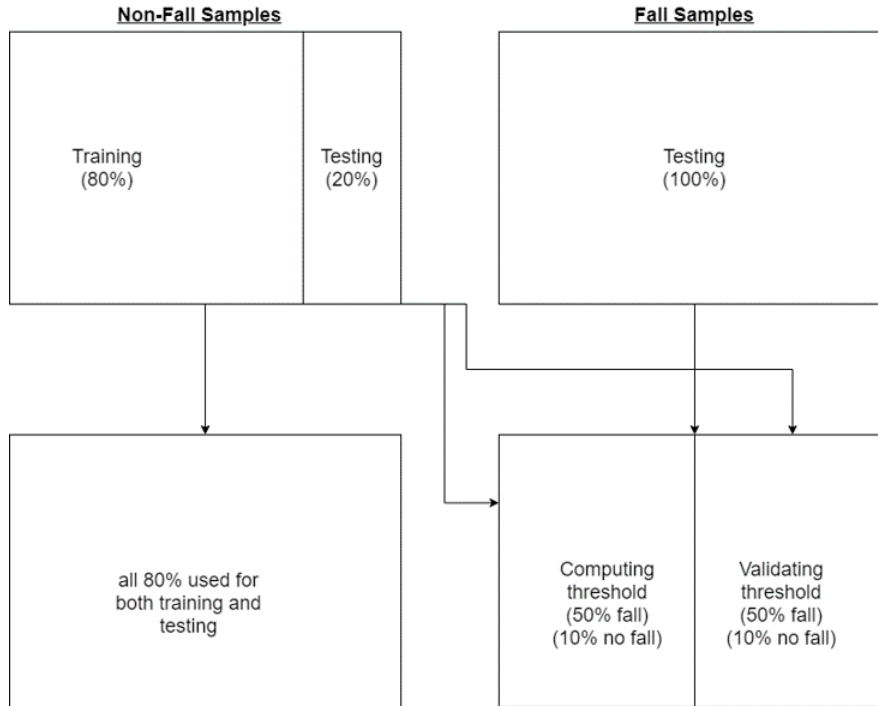


Figure 3.6: Different possible data splitting techniques, right technique implemented

3.4 Model Development

The model developed for this paper was created using Tensorflow and Keras. A Variational Autoencoder (VAE) was constructed using a Recurrent Neural Network (RNN) as the bottleneck. This model was influenced from the paper written by *Jin et al.* [15]. RNN models allow for effective information capture from data that relies on previous information. For this reason, an RNN was chosen over the traditional linear layer at the bottleneck. The input to the model is a series of frame sequences in the following format $[x, \text{window_size}, \text{max_frame_size}, 4]$. x represents how many frame sequences are being passed into the model ($x = 1$ in real-time). Window size is a customisable hyper-parameter that determines how many frames are being reconstructed at once. Finally, the max frame size is the largest number of points in a single frame, all frames are oversampled to match this size. The output of the model is a combination of the input mean and input log variance which is used to compute the reconstruction loss of the network. The layers to the network developed are the following:

1. Input: Input layer $[?, 20, 288, 4]$
2. Encoding: Linear layer flatten inputs $[?, 20, 288, 4] \rightarrow [?, 20, 1152]$
3. Encoding: Dense encoding layer (shrink input) $[?, 20, 1152] \rightarrow [?, 20, 288]$
4. Encoding: Input mean dense layer $[?, 20, 288] \rightarrow [?, 20, 72]$
 Encoding: Input log variance layer $[?, 20, 288] \rightarrow [?, 20, 72]$
5. Encoding: Sampling layer $[?, 20, 72]$ and $[?, 20, 72] \rightarrow [?, 20, 72]$
6. Bottleneck: Simple RNN layer $[?, 20, 72] \rightarrow [?, 72]$
7. Bottleneck: RepeatVector with 20 repeats $[?, 72] \rightarrow [?, 20, 72]$
8. Bottleneck: Simple RNN layer $[?, 20, 72] \rightarrow [?, 20, 72]$
9. Decoding: RNN output layer $[?, 20, 72] \rightarrow [?, 20, 72]$
10. Decoding: Dense decoding layer (expand input) $[?, 20, 72] \rightarrow [?, 20, 288]$
11. Decoding: reconstructed mean layer $[?, 20, 288] \rightarrow [?, 20, 4]$
 Decoding: reconstructed log variance $[?, 20, 288] \rightarrow [?, 20, 4]$
12. Decoding: combination layer (Concatenate) $[?, 20, 4]$ and $[?, 20, 4] \rightarrow [?, 20, 8]$
13. Decoding: RepeatVector with 288 repeats $[?, 20, 8] \rightarrow [?, 20, 288, 8]$

14. Output: Linear Layer $[?, 20, 288, 8] \rightarrow [?, 20, 288, 8]$

Figure 3.7 gives a visual representation of the VAE that was developed.

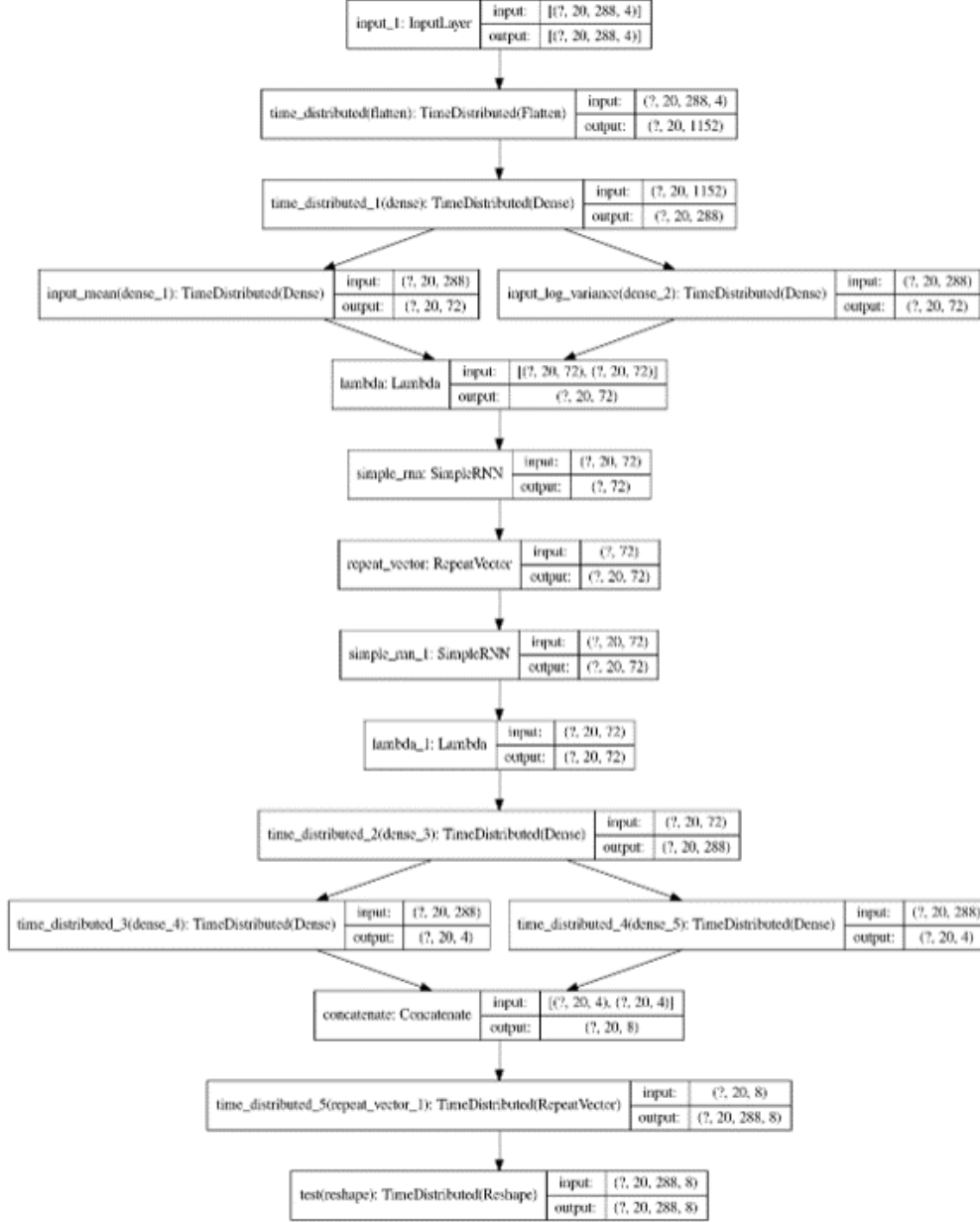


Figure 3.7: Variational Autoencoder model with all layers involved

Using the output of this network, we are able to compute the reconstruction loss using the traditional VAE loss function. *Rocca J.* [31] explains that VAEs are optimised to maximise the probability that " $x^* = x$ " (probability our reconstructed output is

the input) when we sample z from the distribution $q_x^*(z)$ and then sampling x^* from the distribution $p(x|z)$. In other words, we want to maximise the likelihood of the reconstructed input whilst staying close to the distribution of our input. To achieve this behaviour we can maximise the expected log-likelihood whilst minimising the KL divergence between our distributions. The following loss function was implemented in python.

$$\mathbb{E}_Q[\log P(x|z)] - KL(Q(Z)||P(Z))$$

3.5 Thresholding

Given the output of the model is a single loss value representing reconstruction loss, and the binary nature of our classification problem, thresholding will allow us to distinguish between two output classes. Our model is trained to reconstruct ADL activities giving us overall a lower reconstruction loss on ADL data. When other activities are reconstructed through the model, there is a higher reconstruction loss as the model has not been trained on these types of activities. The key benefit to semi-supervised learning is that we don't need to train using fall data to predict falls, however, this requires a fine-tuned threshold to distinguish between abnormalities and regular activity data. This project aims to predict falls in a manner that maximises the true positive rate whilst minimising the false alarm rates. Rather than performing multi-variable optimisation on two separate values, we had used Youden's Index to create one value that is the true positive rate minus the false positive rate. The aim is then to maximise the Youdin's Index to determine an optimal threshold for distinguishing falls and ADL.

It is worth noting that thresholding creates a trade-off for overall model accuracy, a higher threshold will result in less accuracy on fall samples but a higher accuracy on ADL samples, and vice-versa. This makes the aim of our threshold to find an equilibrium between ADL and fall accuracy, whilst also allowing for the ability to adjust the threshold to suit specific performance requirements.

3.6 Technologies

The language of choice for the entire project was python. All data pre-processing and algorithms are carried out in python due to the simplistic nature of python code. Data pre-processing is a complex task that required many algorithms involving linking multiple files together. If performance was a requirement, the code would have been performed in C++. To develop the model TensorFlow and Keras were the chosen machine learning framework. The team I had worked with had collaborated using bitbucket as well as a shared scratch on katana for a centralised dataset. Figure 3.8 gives a visual overview of the architecture and technology used throughout the project.

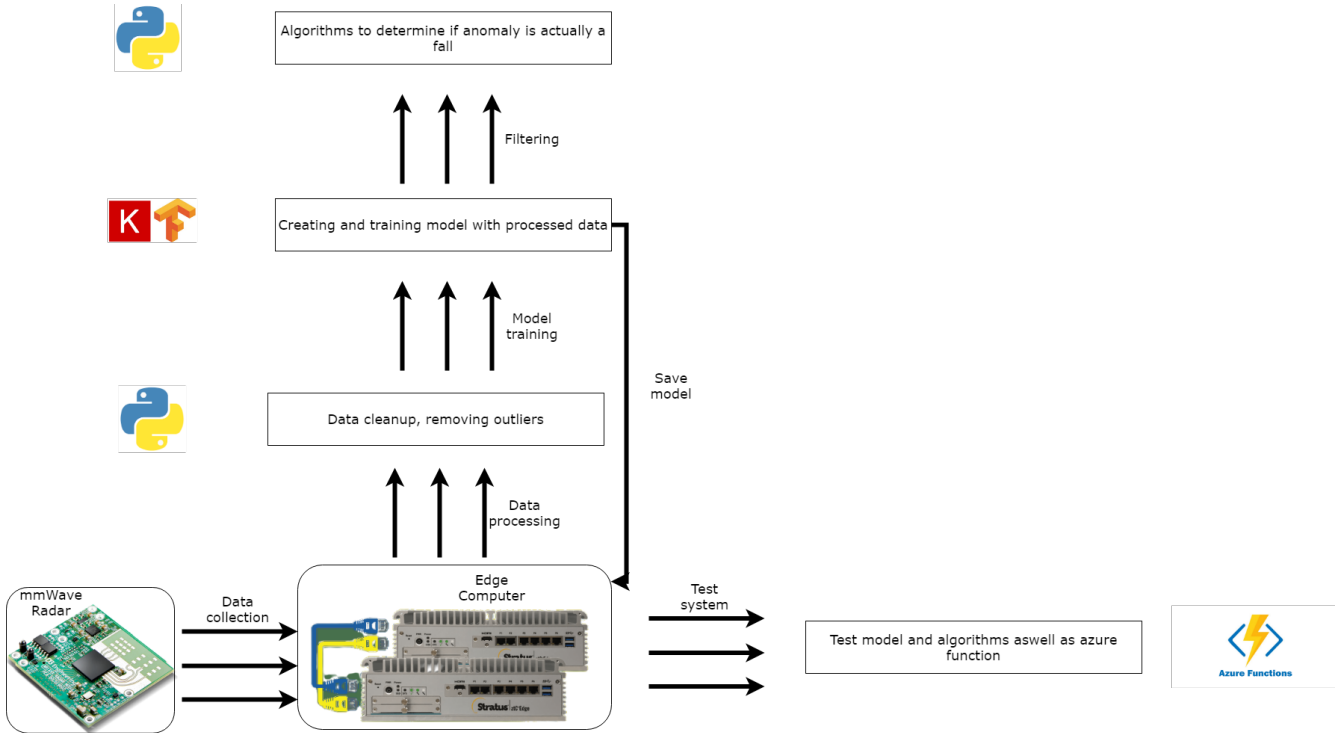


Figure 3.8: Architecture used for this project

Chapter 4

Results and Discussion

Results for this project were generated using a model that was trained on 1554 regular activity samples. The activities that constitute abnormal data are just falls, and the activities that constituted ADL data are the following:

- LayBed
- LayFloor
- Sit
- Background
- SitBed
- Stand
- Transition
- Walking

The model was trained for 100 epochs, with a window size of 20, and a window step size of 1. A window size of 20 was chosen due to the average length of fall sequences ranging from 20-30 frames. Using a window step size of 1 in conjunction with this

window size allows us to generate more contexts and train on more data. With this layout, an activity sample has the following size.

$$sample_sequences = \frac{total_frames - window_size}{window_step_size}$$

To test the model, we must first compute a threshold. This threshold is computed using 194 ADL samples and 158 fall samples. To validate the threshold and determine the overall performance of the model, as well as threshold performance, 194 ADL samples, and 158 fall samples, are used.

4.1 Results

4.1.1 Model Output

To display the results of the models' output after the loss function is applied, a histogram was made using the loss values for ADL data and fall data. Figure 4.1a shows the losses for ADL data, Figure 4.1b shows the losses for fall data and Figure 4.2 shows the losses for the fall data put on top of the ADL data.

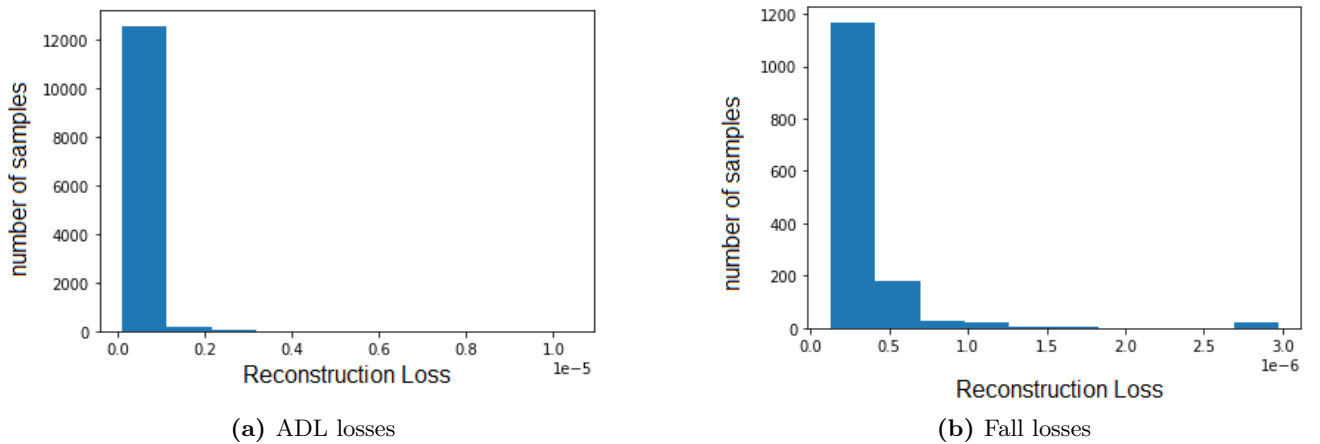


Figure 4.1: Losses from model side by side

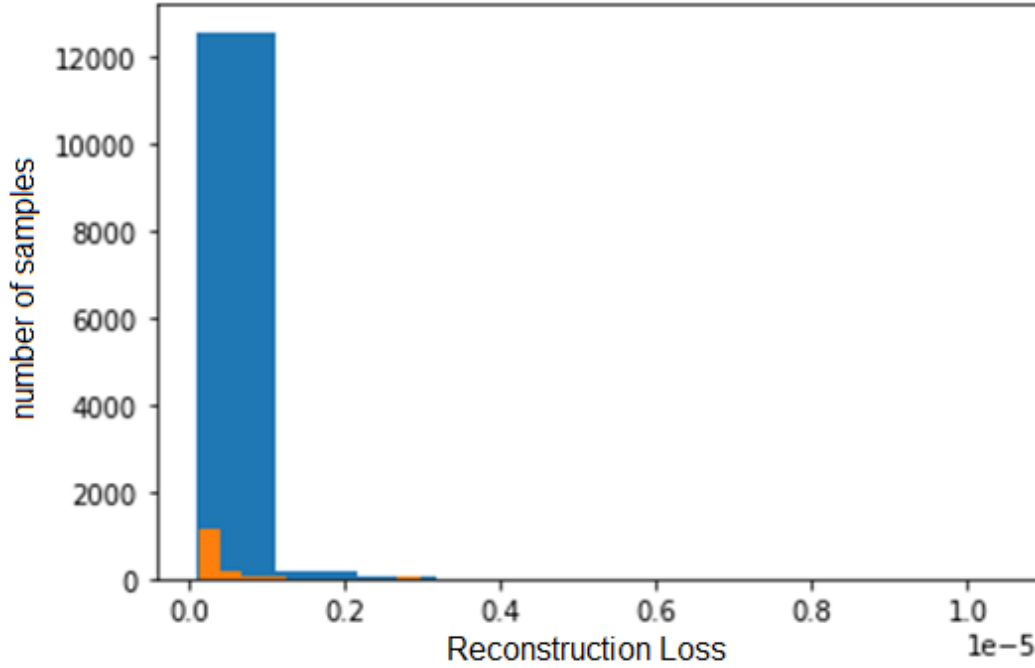


Figure 4.2: Fall losses put ontop of ADL losses

These results show that there is a much larger (factor of 10) number of frame samples in ADL data in contrast to fall samples. The imbalance of fall samples to ADL samples can be contributed to the rarity of falls in contrast to regular activities.

Another key result to be noted is that there is a significant overlap between fall losses and ADL losses. This overlap makes it impossible to find an optimal threshold to distinguish between falls and regular activities. Ideally, there would be a clear divide between the fall losses and ADL losses.

4.1.2 Metrics

To generate the following metrics, 194 ADL samples and 158 fall samples are used. After the threshold is computed using Youden's Index, each sample is checked one at a time. For each sample, we check every single set of frames in the frame sequence, and if any of them are above the threshold, we detect a fall. If the entire sample has losses below the threshold then it is detected as a normal activity. When running this

model in real-time it is worth noting that we only handle 1 set of frames at a time for classification.

4.1.2.1 Accuracy

Table 4.1 gives the model’s classification accuracy for falls, ADL data, and combined accuracy. The model accuracy gives an overview of the number of correctly identified samples.

Model Accuracy	
Activity	Accuracy
Falls	40.00%
ADL	41.12%
Total	40.78%

Table 4.1: Model accuracy accross all activities.

4.1.2.2 F1 Score

When it comes to binary classification, the F1 score gives a deeper insight into model performance. To compute the F1 score, we require the model’s precision and recall. Precision as a metric gives the percentage of positive classifications that were actually correct [32]. Model precision is given by the following formula.

$$Precision = \frac{True_Positives}{True_Positives + False_Positives}$$

Recall as a metric gives us the percentage of actual positive classifications that were identified correctly [32]. The model recall is computed from the following formula.

$$Recall = \frac{True_Positives}{True_Positives + False_Negatives}$$

The F1 score is a combination of precision and recall that aims to give a better idea of the number of incorrectly classified instances in contrast to accuracy. Model F1 score is computed from the following formula.

$$F1Score = \frac{2 * (Precision * Recall)}{(Precision + Recall)}$$

Table 4.2 displays the model's precision, recall, and F1 Score.

F1 Score	
Metric	Result
Precision	26%
Recall	40%
F1 Score	31%

Table 4.2: Model F1 Score

4.1.2.3 ROC Curve

ROC curves are an excellent metric for determining the performance of a model where a threshold is required. ROC curves allow us to determine how well a model distinguishes between output classes. The curve is a map of the True Positive Rate against the False Positive Rate. The area under the ROC gives a metric to how well the model performed, where 1 is perfect performance. Ideally, the ROC curve follows a logarithmic shape, maximising the area under the curve. Figure 4.3 displays the model's ROC curve.

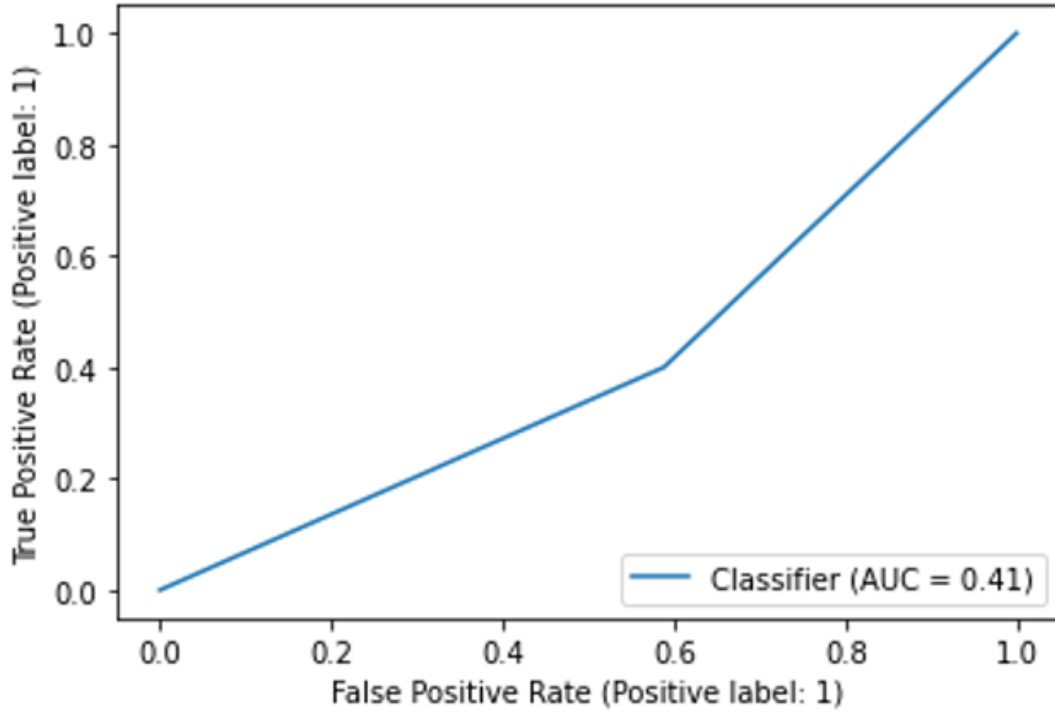


Figure 4.3: ROC Curve for model

The model developed had an ROC curve that follows the shape of a step function. The step function shifts between two linear functions. The model AUC is 0.41, indicating poor performance.

4.1.2.4 Log Loss

Log-loss is a metric that determines the probability that the prediction is close to the true value. The more the probability diverges from the actual value, the higher the log-loss value [33]. A perfect model has a log-loss score of 0. The higher the log-loss score the worse the performance of the model. The formula to compute log-loss is the following.

$$Logloss = -\frac{1}{N} \sum_{i=1}^N y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$

The current model had a log loss value of 20.45 indicating poor performance of the model.

4.2 Evaluation

4.2.1 Outcomes and Analysis

Through this project, I have developed many scripts and programs that are flexible and simple to work with. I have developed a powerful and specialised data cleaning script that is able to remove noise points, outliers, invalid data points as well as label the data. The data cleaning script is able to create sub-samples out of raw data containing activity data. Alongside this script is a data pre-processing script that groups together points into frame sequences. Connected to these scripts is a detachable variational autoencoder that has capabilities to be run in real-time. With special thanks to Dr. Reza, I was able to develop code for determining an optimal threshold using Youden's Index. The final major outcome of this thesis was determining the underlying issues that severely affected the model's result.

The major takeaway from this project is the versatile and flexible pipeline that was developed. Despite the issues in the data used in this project, the methodology carried out was tested and verified for correctness. The entire process from processing raw data into determining model performance and optimal threshold on a different dataset is as simple as changing the dataset.

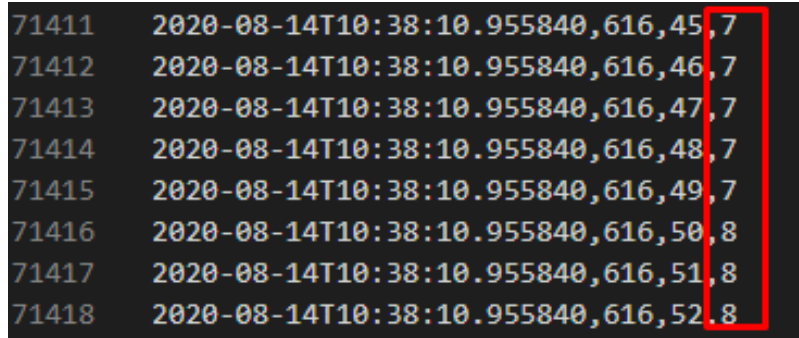
The model had performed poorly in distinguishing falls from ADL activities. However, this issue can be attributed to a poor dataset, and in any machine learning project, your model is as good as the data it has to work with. When compared with other models developed by colleague Nathan Ng, the accuracy had only differed by 2%. The same models developed had performed exceptionally well when used on different datasets. Whilst the issue could be that this approach is unrealistic for the detection of falls and the model performs poorly, it is more likely that the issue lays inside of the dataset.

4.2.2 Data Issues

When performing a in-depth look into the dataset major issues had emerged that were not taken into consideration during data collection. This project had been reliant on this dataset and worked under the assumption that the data was not problematic.

4.2.2.1 Target Tracking

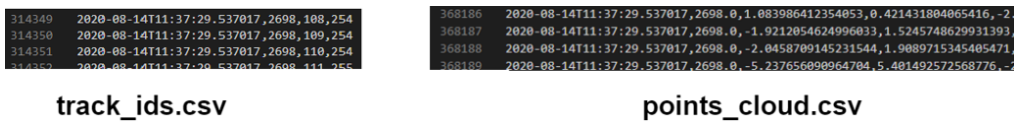
Target tracking is a crucial step in determining which point belongs to which entity. In the samples since there is only one person in the scene, we expect that there is only one target that is being tracked. However when observing the dataset it can be noted that multiple targets are being detected. Figure 4.4 displays the issue of multiple targets being tracked.



71411	2020-08-14T10:38:10.955840,616,45,7
71412	2020-08-14T10:38:10.955840,616,46,7
71413	2020-08-14T10:38:10.955840,616,47,7
71414	2020-08-14T10:38:10.955840,616,48,7
71415	2020-08-14T10:38:10.955840,616,49,7
71416	2020-08-14T10:38:10.955840,616,50,8
71417	2020-08-14T10:38:10.955840,616,51,8
71418	2020-08-14T10:38:10.955840,616,52,8

Figure 4.4: Multiple target id's when only one target in scene.

Alongside having multiple targets detected, the synchronisation between the target id file and raw point data file are inconsistently out of sync. In some cases, the target id file would be missing up to 50,000 points, meaning those points could not be linked to any target or determined to be noise. Figure 4.5 shows the large disparity between the track id file and point cloud file.



track_ids.csv	points_cloud.csv
314349 2020-08-14T11:37:29.537017,2698,109,254	368186 2020-08-14T11:37:29.537017,2698,0,1.083986412354053,0.421431884065416,-2.
314350 2020-08-14T11:37:29.537017,2698,109,254	368187 2020-08-14T11:37:29.537017,2698,0,-1.9212054624996033,1.5245748629931393,
314351 2020-08-14T11:37:29.537017,2698,110,254	368188 2020-08-14T11:37:29.537017,2698,0,-2.0458709145231544,1.9089715345405471,
314352 2020-08-14T11:37:29.537017,2698,111,255	368189 2020-08-14T11:37:29.537017,2698,0,-5.237656090964784,5.401492572568776,-2

Figure 4.5: 50,000 points missing from track id file.

4.2.2.2 Noise Removal

An enormous chunk of the data was attributed to noise. Around 80% of all points were classified as noise when determining the track ids. The removal of this noise created a largely sparse dataset and displayed quite some strange results. In appendix 1, a sequence of images displays the severity of the noise removal when it comes to this dataset. The first two images in appendix 1 show the contrast between noise removal correctly working and noise removal not working entirely. Following is a sequence of images showing frames that are classified as falls after having noise removal performed on them. These frames are in sequential order from one fall sample. As shown in appendix 1, it would randomly switch between correctly removing noise and being unable to distinguish noise.

The inability to consistently determine noise from regular data, alongside 80% of the data being noise itself, makes it impossible to achieve meaningful results.

4.2.3 What Could of Been Different

When looking back to thesis A, it was clear I was not checking back on my work as often as I should have. When developing complex algorithms and specific scripts, it is important to take a step back to ask yourself what are you doing, and make sure you're on the right track. Had I understood the data earlier on and done more work with the data, the entire scope of my project may have changed altogether. The data issues came to life in thesis C when trying to analyse my results for the first time. The final change I would have made going back would have been to manage my time better to get more out of this project. Often I would find myself stuck on specific issues that did not have an obvious solution. In these situations, it would have been better to ask my supervisor for help or consider alternative approaches instead of forcing certain solutions to work.

Chapter 5

Conclusion

This report has outlined the motivation and implications of a fall detection system using semi-supervised learning. A plan was made to develop this system in a flexible and maintainable manner. The pipeline created lays the groundwork for any future work, the only change needed is in the dataset itself. Despite poor results when determining model accuracy, this is likely attributed to a poor dataset. The model developed has a major impact in the field of fall detection, in that it does not require fall data to get better. Due to the semi-supervised nature of this learning, we can classify falls without having ever trained on one.

5.1 Future Work

Moving forward, there are many directions in which this project can be taken. However, before any further work is done, there are immediate changes that must happen moving forward.

- **Target Tracking Algorithms:** The current data issues can be attributed to the target tracking performed by the mmWave sensor. The sensor itself has built-in algorithms and processes to track targets and classify noise. It is quite evident

that these algorithms aren't perfect, and further data pre-processing algorithms should be performed to accurately track the target and classify noise.

- **Convolutional layers in autoencoder:** In contrast to using raw mmWave data, occupancy grids can be created using frame data and passed into the variational autoencoder through convolutional layers. This model can be directly compared to the current model.
- **Feature extraction:** Using a 3D CNN and occupancy grids to perform automatic feature extraction. These extracted features can be connected to the variational autoencoder. This model can also be directly compared to the current model.
- **Realtime:** Model reads in a set of frames in real-time and performs prediction. Currently, all the groundwork is there, all that needs to be done is live pre-processing on incoming frames and passing them through the saved model. This however should only be done when the model has been shown to work well.
- **Azure Function alert:** System to send an alert to a chosen emergency contact upon detecting a fall. This would be done through an Azure function. Similar to real-time implementation, this should only be done when the model has been shown to work well.
- **Fall Prevention:** During data cleaning, transition data is captured from activity A to activity B. This is done through a customisable python script, where the last x frames of activity A are combined with the first x frames of activity B (x is a customisable hyperparameter). Using this transition data for training fall prevention can be looked into.

Bibliography

- [1] WHO, Falls, World Health Organization, 2018, [Online]. Available: <https://www.who.int/news-room/fact-sheets/detail/falls>
- [2] R. Luque, E. Casilari, M.-J. Morón, G. Redondo, Comparison and Characterisation of Android-Based Fall Systems, *Sensors* 14, IEEE, 2014, 18543-18574
- [3] S. S. Khan, J. Hoey, Review of Fall Detection Techniques: a Data Availability Perspective, University of Waterloo, 2016, pp.6, 8-9, 12, 19
- [4] L. Schwickert, C. Becker, U. Lindemann, C. Maréchal, A. Bourke, L. Chiari, J. Helbostad, W. Zijlstra, K. Aminian, C. Todd, S. Bandinelli, J. Klenk, Fall detection with body-worn sensors: a systematic review, *Zeitschrift für Gerontologie und Geriatrie*, IEEE, 2013, 706–719
- [5] B. Marr, Facial Recognition Technology: Here Are The Important Pros and Cons, *Forbes*, 2019, [Online]. Available: <https://www.forbes.com/sites/bernardmarr/2019/08/19/facial-recognition-technology-here-are-the-important-pros-and-cons>
- [6] healthdirect, Falls and The Elderly, healthdirect, 2020, [Online]. Available: <https://www.healthdirect.gov.au/falls>
- [7] Stay On Your Feet, Falls facts , Stay On Your Feet, 2021, [Online]. Available: <https://www.stayonyourfeet.com.au/over60/what-can-cause-a-fall/falls-facts/>
- [8] R. W. Broadley, J. Klenk, S. B. Thies, L. P.J. Kenney, M. H. Granat, Methods For The Real-World Evaluation of Fall Detection Technology: a Scoping Review, *MDPI*, 2018, pp. 1-3
- [9] CDC, Cost of Older Adult Falls, Centers for Disease Control, 2020, [Online]. Available: <https://www.cdc.gov/homeandrecreationalsafety/falls/data/fallcost.html>
- [10] J. Fleming, C. Brayne, Inability To Get Up After Falling, Subsequent Time On Floor, and Summoning Help: Prospective Cohort Study in People Over 90, *BMJ*, 2008, [Online]. Available: pp. 1, 3-4

- [11] Medical Alert Advice, Is The Apple Watch Series 4, 5 or 6 With Fall Detection Right For You?, Medical Alert Advice, 2021, [Online]. Available: <https://www.medicalalertadvice.com/articles/apple-watch-fall-detection/>
- [12] A. Shahzad, K. Kim, FallDroid: an Automated Smart Phone Based Fall Detection System Using Multiple Kernel Learning, IEEE, 2018, pp. 1-2, 9
- [13] G. W. Lindsay, Convolutional Neural Networks as a Model of The Visual System: Past, Present, and Future, J Cogn Neurosci, 2020, pp. 3-5
- [14] S. Ji, M. Yang, K. Yu, 3D Convolutional Neural Networks For Human Action Recognition, IEEE, 2013, pp. 221-230
- [15] A. Núñez-Marcos, G. Azkune, I. Ignacio Arganda-Carreras, Vision-Based Fall Detection With Convolutional Neural Networks, Hindawi, 2017, pp. 221-230
- [16] J. Arunnehru, G. Chamundeeswari, S. P. Bharathi, Human Action Recognition Using 3D Convolutional Neural Networks With 3D Motion Cuboids in Surveillance Videos, ScienceDirect, 2018, pp. 472-477
- [17] F. Jin, R. Zhang, A. Sengupta, S. Cao, S. Hariri, N. K. Agarwal, S. K. Agarwal, Multiple Patients Behavior Detection in Real-time Using mmWave Radar and Deep CNNs, IEEE, 2019, pp. 1-5
- [18] L. Palmerini, J. Klenk, C. Becker, L. Chiari, Accelerometer-Based Fall Detection Using Machine Learning: Training and Testing on Real-World Falls, MDPI, 2020, pp. 2-15
- [19] I. J. Goodfellow, J. Shlens C. Szegedy, Explaining and Harnessing Adversarial Examples, ICLR, 2015, pp. 1-6
- [20] Abacus.AI, The Intuition Behind Variational Autoencoders, Abacus.AI, 2020, [Online]. Available: <https://medium.com/@realityenginesai/understanding-variational-autoencoders-and-their-applications-81a4f99efc0d>
- [21] F. Jin, A. Sengupta, S. Cao, mmFall: Fall Detection Using 4D MmWave Radar and a Hybrid Variational RNN AutoEncoder, IEEE, 2020, pp. 1-10
- [22] C. Medrano, R. Igual, I. Plaza, M. Castro, Detecting Falls as Novelties in Acceleration Patterns Acquired With Smartphones, PloS one, 2012, pp. 1-9
- [23] M. Yu, Y. Yu, A. Rhuma, S. M. R. Naqvi, L. Wang, J. A. Chambers, An Online One Class Support Vector Machine-Based Person-Specific Fall Detection System For Monitoring an Elderly Individual in a Room Environment, IEEE, 2013, pp. 1002–1014
- [24] J. Nogas, S. S. Khan, A. Mihailidis, DeepFall – Non-invasive Fall Detection With Deep Spatio-Temporal Convolutional Autoencoders, arXiv, 2020, pp. 1–18

- [25] C. Lovescu, S. Rao, mmWave Radar Sensors - What Is mmWave, Texas Instruments, 2020, [Online]. Available: <https://www.ti.com/sensors/mmwave-radar/what-is-mmwave.html>
- [26] C. Lovescu, S. Rao, The Fundamentals of Millimeter Wave Sensors, Texas Instruments, 2020, pp. 1-9
- [27] G. Koshmak, M. Lindén, A. Loutfi, Challenges and Issues in Multisensor Fusion Approach For Fall Detection: Review Paper, Hindawi, 2015, pp. 1-12
- [28] G. Pang, C. Shen, L. Cao, A. V. D. Hengel, Deep Learning for Anomaly Detection: a Review, arXiv, 2020, pp. 1-36
- [29] B. R. Kiran, D. M. Thomas, R. Parakkal, An Overview of Deep Learning Based Methods For Unsupervised and Semi-Supervised Anomaly Detection in Videos, MDPI, 2018, pp. 1-25
- [30] Y. Karadayi, M. N. Aydin, A. S. Ögürcü, A Hybrid Deep Learning Framework for Unsupervised Anomaly Detection in Multivariate Spatio-Temporal Data, MDPI, 2020, pp. 1-25
- [31] Rocca J., Understanding Variational Autoencoders (VAEs)., towards data science, 2020, [Online]. Available: <https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73> <https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73>
- [32] Google, Classification: Precision and Recall, Google, 2020, [Online]. Available: <https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall>
- [33] Dembla G., Intuition behind Log-loss score, Towards Data Science, 2020, [Online]. Available: <https://towardsdatascience.com/intuition-behind-log-loss-score-4e0c9979680a>

Appendix 1

Figure B.1 and B.2 show noise removal working in two contrasting ways

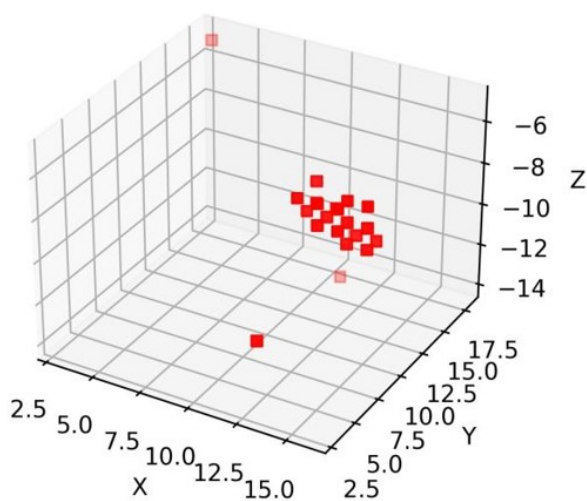


Figure B.1: Correct noise removal behaviour.

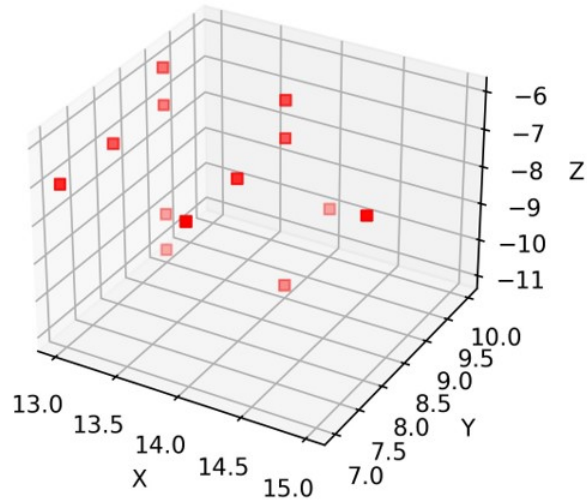


Figure B.2: Incorrect noise removal behaviour.

These next sequence of images show a fall sample frame by frame and display the point clouds corresponding. It can be shown that the target is being lost inconsistently, polluting the falling sample with noise data.

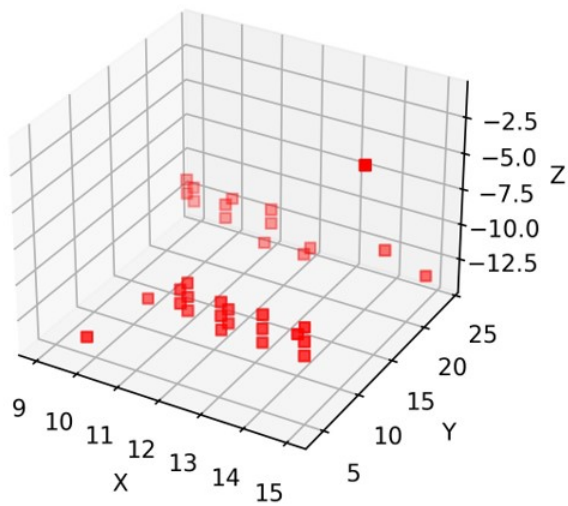


Figure B.3: Fall frame 1 normal.

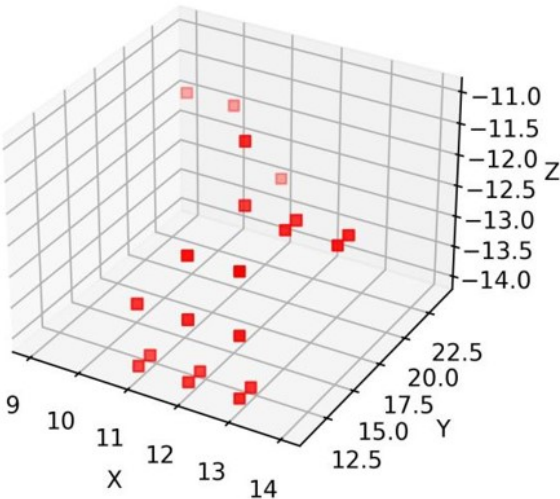


Figure B.4: Fall frame 2 wrong.

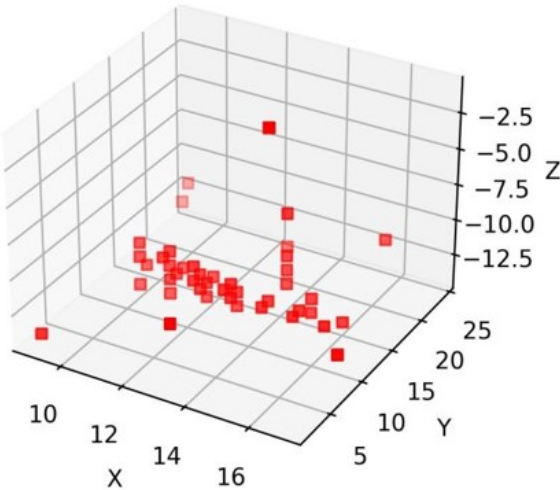


Figure B.5: Fall frame 3 normal.

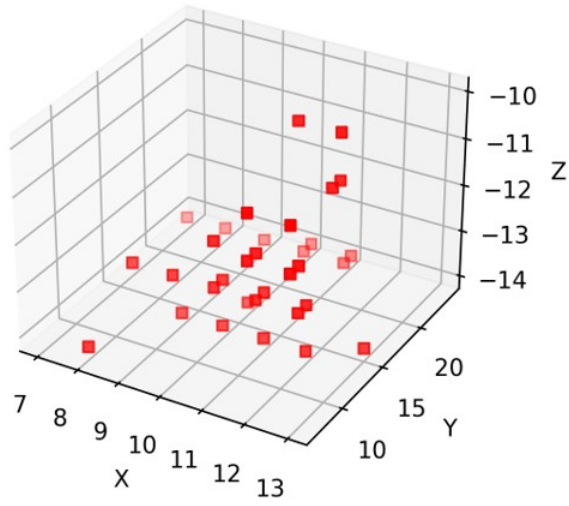


Figure B.6: Fall frame 4 wrong.

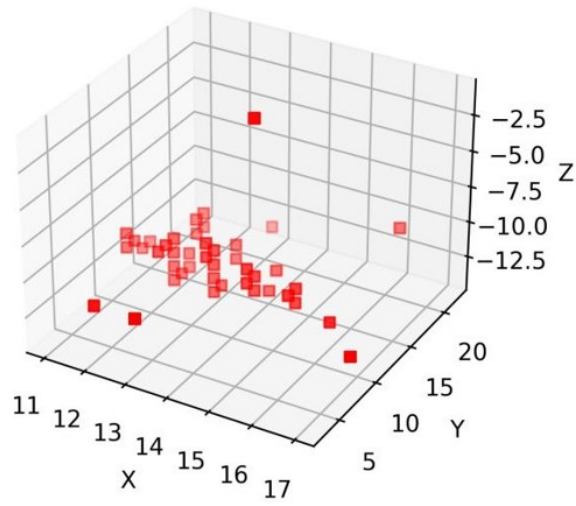


Figure B.7: Fall frame 5 normal.

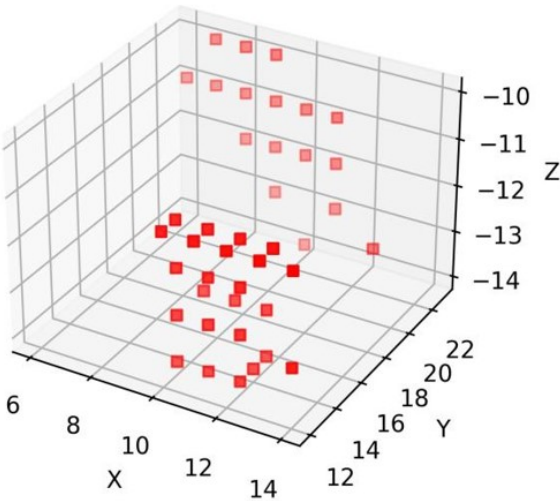


Figure B.8: Fall frame 6 wrong.