Detecting human abnormal behaviour through a video generated model

THOMAS GATT

Supervised by Mr Dylan Seychell

Co-supervised by Prof Alexiei Dingli

Department of Artificial Intelligence

Faculty of Information and Communication Technology

University of Malta

March, 2019

A dissertation submitted in partial fulfilment of the requirements for the degree of M.Sc. Artificial Intelligence.

Statement of Originality

I, the undersigned, declare that this is my own work unless where otherwise acknowledged and referenced.

Candidate	Thomas Gatt
Signed	______________________
Date	August 30, 2019

To my wife Christina

For all her support throughout this period.

EU funds for Malta | 2014 2020

The research work disclosed in this publication is partially funded by the Endeavour Scholarship Scheme (Malta). Scholarships are part-financed by the European Union - European Social Fund (ESF) - Operational Programme II – Cohesion Policy 2014-2020 "Investing in human capital to create more opportunities and promote the well-being of society".

European Union – European Structural and Investment Funds
Operational Programme II – Cohesion Policy 2014-2020
"Investing in human capital to create more opportunities and promote the well-being of society"
Scholarships are part-financed by the European Union - European Social Funds (ESF)
Co-financing rate: 80% EU Funds;20% National Funds

Acknowledgements

First and foremost I would like to express my gratitude to my supervisor, Mr Dylan Seychell and my co-supervisor Prof Alexiei Dingli, whose expertise, understanding and patience helped me to complete my dissertation.

I would also like to thank my wife Christina and my family, especially my brother William, for the support they provided me with throughout this journey. I must also acknowledge my colleagues without whose encouragement and support, I would not have finished this dissertation.

Last but not least I would like to thank all the participants who took the time from their busy schedules to participate in this study. Without their participation and feedback, this study would not have been possible.

Abstract

Detecting human abnormal activities is the process of observing rare events that deviate from normality. In this study, an automated camera-based system that is able to monitor and detect irregular human behaviour is proposed. Pre-trained pose estimation models are used to detect the person in the frame and extract the body keypoints. Such data is used to train two types of AutoEncoders in a semi-supervised approach where the goal is to learn a general representation of the normal behaviour. Specifically, the AutoEncoders are based on Long short term memory (LSTM) and convolutional layers respectively, for their ability to learn local temporal features. To classify the data sequences, the reconstruction error of the model is used. Evaluated on two types of datasets, the results show that both types of models were able to correctly distinguish between normal and abnormal data sequences, with an average F-score of 0.93. The results also show that the proposed method outperformed similar work done on the same dataset. Furthermore, it was also determined that pose estimated data compares very well with sensor data. This shows that pose estimated data can be informative enough to understand and classify human actions.

Contents

List of Figures

List of Tables

1 Introduction

Our brains are capable of capturing data around us using senses, such as vision, and creating a hierarchy of concepts. Throughout our lives we train our brain in various ways. Human beings are able to make sense out of things and understand patterns. We also learn how to identify and recognise objects as well as detect abnormal events. A computer program such as a calculator can be programmed to perform arithmetic operations by accepting numeric inputs and an operation. However, for a computer to be able to recognise and detect events, for instance, it first needs to learn how real-world things look (in terms of pixels or other representations) and then be able to classify them.

1.1 Motivation

Approximately 28 to 35% of people aged 65 and over fall twice or more throughout a year [59]. According to the same report, individuals sustaining falls often require medical attention, with some falls possibly also leading to injury deaths. It is also reported that older people living in nursing homes fall more frequently than those residing in their residences.

A person who is home alone doing some mundane task might suddenly feel sick and could, in a matter of seconds, fall to the floor or wind up helpless on a chair. If no one is around to help, the individual's health might deteriorate, possibly to the point of death. A system which detects such abnormal events can provide timely aid by triggering an alarm when an anomalous event (such as a fall) is detected.

In anomaly detection, the main objective is to detect any events that are abnormal, suspicious or rare [10]. With specific regard to human behaviour, abnormality refers

to behaviour that is unusual and deviates from normality. The day-to-day activities is what defines 'normal' activities. On the other hand, activities (or behaviour) that do not follow such normal day-to-day activities are defined as 'abnormal'. For example, an individual walking from point A to point B is considered normal; however if the individual is walking and suddenly falls to the floor then that is considered abnormal. In general, any instance not seen in the training set is considered as abnormal.

1.2 Problem Definition

Current abnormality detection systems focus on detecting irregular objects or understanding the scene in a holistic way [100], rather than on human behaviour. Other detection systems rely on wearable sensors [98] that must be worn at all times. However such method might be intrusive for some or some may simply forget to put the sensors on. Others use special equipment such as depth cameras [23]. Such equipment, which is usually not present in nursing homes, requires additional installation and could be expensive to acquire when compared to traditional cameras. Other studies aim to detect falls by training models on specific features such as velocity characteristics and tracking of the head [68] and joint positions [97]. However, such approaches only allow detection of specific changes in the body and might not capture other abnormal behaviour performed by the individual. For example, a person might be showing signs of distress by waving his/her hands.

1.3 Aims and Objectives

The majority of the work in anomaly detection focuses on general scene understanding such as pedestrian tracking, crowd control, unrestricted areas and irregular object detection [86; 100; 101]. This study focuses on individual-human based activities and the detection of irregular human behaviour based on human posture from a single camera. In this study, irregular or abnormal human behaviour refers to abnormality in the body posture. This means that, from a sequence, consideration is given only to human representation, specifically the body keypoints. In this case, any human sequence pose that is not seen in the training set is considered as abnormal.

The main research questions that this study aims to answer include:

1. How would human action classification models perform if they were to be trained on pose data estimated from a 2D image instead of sensor data?
2. Is it possible to train a semi-supervised model using solely body estimated keypoints to detect anomalous sequences?

Based on these research questions, the main aim is to determine whether a set of human body keypoints extracted from RGB images can be used to detect abnormal human behaviour such as falling. The study also seeks to determine whether human body keypoints extracted from RGB images are comparable to sensor data collected from humans performing various actions.

In order to achieve these aims, the following objectives were set up:

- To process and collect body keypoints from video footage containing normal and abnormal behaviour. This is presented in Chapter 4, Section 4.4.
- To train a number of supervised models in order to compare sensor data with estimated body keypoints from RGB images. This is presented in Chapter 4, Section 4.6.
- To train a number of semi-supervised models that should be able to detect abnormal human behaviour. The methodology as well as the model architectures are presented in Chapter 4, Section 4.5.
- To evaluate the effectiveness of all the models. This is presented in Chapter 5.

1.4 Proposed Solution

The proposed idea for this dissertation is a system that monitors the body posture of an individual through a traditional camera, and learns the way the user behaves. The system would learn patterns over time and would therefore be capable of building a personalised user model. The set-up in itself is non-invasive and respects the individual's privacy as only the skeletal data is used for processing. The individual's body posture would be extracted from live video footage. This is done by collecting body key points such as left and right shoulders, elbows, hips and knees.

A model would be trained to learn the typical behaviour of the individual in terms of body posture. Afterwards, the model would be capable of detecting body posture that deviates from the behaviour it was trained on. In a real world scenario, a voice user interface (such as Google Assistant) could be used to communicate with the individual if an abnormal event is detected. Such system would be able to act accordingly such as calling the person's relatives or the ambulance.

It is therefore desirable to have a system that can actually perform intelligent detection of irregular behaviour as a timely decision might possibly save a life. The main contribution of this dissertation is the ability to detect abnormal behaviour using solely data from a traditional camera. Having such intelligent detection in place, future development such as systems that communicate directly with the individual could be implemented.

In contrast to the current research, this study aims to detect abnormal or irregular behaviour by learning changes in the human skeleton (and thus focus on the action being undertaken). Training the model with just a representation of the human being from a whole frame allows the model to be trained solely on the human action being performed. This ensures that the background itself or objects visible in the scene do not contribute to detecting abnormalities. The type of architecture selected also allows the model to learn in a semi-supervised approach. This means that the model only requires data which contains the day-to-day behaviour. Unlike data that contains abnormal behaviour, such data is not scarce and usually is the only available data. Furthermore, unlike some current camera-based solutions, this study allows detection of abnormal behaviour using a traditional camera.

Evaluated on a challenging Activities of Daily Living (ADL) video dataset [5], the proposed method achieved a promising average F-score of 0.93, an improvement of $\approx$ 0.30 F-score on similar work by Debard et al. [19]. Furthermore, it is shown that pose estimated data is comparable to sensor data and is informative enough to describe a human action.

1.5 Document Structure

The document is structured into 4 main parts. The first part presents an overview of the relevant literature and machine learning techniques employed in the area. The second part outlines how the literature explored in the first part is applied to reach the aims of this research. The third part presents the evaluation techniques employed to evaluate the proposed solution. And finally, in the fourth part, a list of possible future work together with a recap of the study is given.

Part 1 - Literature Review. This part is organised into two chapters as follows:

Chapter 2: Background and Machine Learning Techniques: This chapter gives an overall context to the next chapter as it covers the background and the machine learning techniques used in anomaly detection.

Chapter 3: Overview and Related Work: This chapter provides a review of the work related to techniques employed in sequence-based data (such as videos and time-series data) where the temporal aspect is important. It also presents an overview of the related work in the field.

Part 2 - Methodology

Chapter 4: This chapter provides a detailed description of the methodology and implementation adopted to develop a technique that aims to detect abnormal behaviour, such as falls, from videos. An overview of the different modules of the system and how these will be combined together are also outlined.

Part 3 - Results and Evaluation

Chapter 5: This chapter includes the results and interpretation of the experiments outlined in Chapter 4. An overview on the datasets utilised is also presented. This chapter includes two types of evaluations - the first one is against the ground truth and the other one is against similar work in the field.

Part 4 - Conclusion and Future Work

Chapter 6: In this chapter, a summary of the whole study is given. This is followed by a detailed description of how each objective was achieved. To conclude, a list of possible future work is given.

2

Background and Machine Learning Techniques

2.1 Introduction

Human action understanding and recognition have received much attention during the last few years [4; 53; 63; 78]. Various machine learning techniques have been employed to understand behaviour and activity patterns, scene understanding as well as intelligent surveillance systems. In anomaly detection, the main objective is to detect any events that are abnormal, suspicious or rare. Video abnormality detection is interesting because it can be applied to various applications namely in the healthcare sector, such as identifying an early onset of an epileptic fit and for security surveillance purposes such as identifying terrorist activities. For this study, anomalies are defined as rare or unexpected events that were not observed in the dataset. Furthermore, human behaviour refers to the body posture of the individual.

Video anomaly detection is a popular topic for many researchers. Various techniques were employed ranging from the use of hand-crafted features and the use of traditional machine learning algorithms such as Hidden Markov Models (HMM) [101], Support Vector Machines (SVM) [98], to the use of neural networks (both shallow and deep) such as Convolutional Neural Networks (CNN)[33; 76; 86], AutoEncoders (AE) [43; 55], Recurrent Neural Networks (RNN) [53; 81] as well as a combination of networks (for example: Convolutional AutoEncoders presented in [32]).

This chapter aims at providing an overview of the machine learning techniques mentioned above whilst the next chapter highlights key techniques and approaches in anomaly detection related to this study.

2.2 Anomaly Detection

Anomaly Detection (AD) deals with the process of detecting rare events or observations that deviate from normality. The goal is to identify abnormal patterns from the data which in nature are rare. Such abnormal patterns are also usually known as outliers, anomalies or discordant observations. Some applications for AD include fault detection in systems, video surveillance, health care and credit card fraud. The study for detecting outliers in data has been on ongoing research topic.

Chandola et al. [10] argue that due to a number of challenges, the anomaly detection problem is not always an easy task. The nature, availability and other factors of the data make the problem more challenging to solve. Some factors that were mentioned in the paper include:

- A data point which is very close to the normal cluster but not part of it can actually be normal or abnormal.
- Different domains employ different types of anomalous data and therefore implementing a technique in a specific domain does not imply that the same technique will produce the same results for another domain.
- Since abnormal events are in nature rare, availability of annotated data is very limited
- Noise in the data can be classified as abnormal when in actual fact it could be otherwise

In their survey paper, Chandola et al. [10] highlight the three types of anomalies: (1) Point Anomalies, (2) Contextual Anomalies and (3) Collective Anomalies. A point anomaly is the simplest form as it deals with a single data point in the dataset and is not part of a context. On the other hand, a contextual anomaly takes into account the context of the data point. As a real world example, let us assume a temperature time series data depicted in Figure 2.1 where both t_1 and t_2 values are equal however they occur in a different context. t_1 in this case can be considered as normal, however t_2 is an abnormal data point. In this case, the context is time. The third type, collective anomaly, deals with a collection of related data points. As an example let us consider a bearing time series dataset whereby a group of data instances at times t_1, t_2, ..., t_n is a complete data sequence. In this case, a single data point in the sequence might not be anomalous but the presence of it in a given sequence is abnormal. In relation to this study, the focus will be on the last two types of anomalies, contextual and collective - where both the context (time) and the sequence (different poses at different timestamps) of events are important.

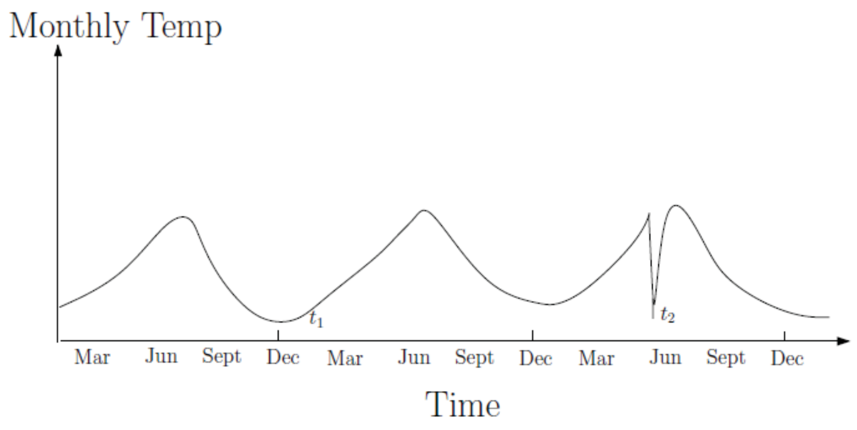


Figure 2.1: Temperature Time Series
Source: Chandola et al. [10]

Kiran et al. describe anomaly detection as an "*unsupervised pattern recognition task that can be defined under different statistical models*". A model is trained to observe and learn the general representation of the data. The normal class distribution $\mathcal{D}$ is calculated using the training dataset. A loss function is then used to minimize the model's error. After training the model, unseen data points can then be checked for abnormality. An abnormal instance is poorly reconstructed by the model where an anomaly threshold score is used to detect where the point should be classified as normal or abnormal [43].

2.3 Machine Learning

Human beings are able to make sense out of things and understand patterns. We also learn how to identify and recognise objects. A computer program such as a calculator can be programmed to do additions and other arithmetic operations by accepting numeric input and an operation. However, for a computer to be able to recognise cats and dogs in an image, for instance, it first needs to learn what cats and dogs look like (in terms of pixels) in order to be able to classify them. This process is referred to as 'machine learning' whereby a model is able to learn data patterns automatically without explicit coding rules [1].

> *"Machine learning is programming computers to optimize a performance criterion using example data or past experience."* Alpaydin [1]

Machine learning can be sub-divided into 4 types: (1) Supervised learning, (2) Semi-supervised learning, (3) Unsupervised learning and (4) Reinforcement learning.

Supervised learning refers to learning by example where a set of x values contain a y value and the model learns to model x to y. This is further explained in Section 2.4. On the other hand, in unsupervised learning there are the x values but no ys. In this case, the aim of the model is to be able to find patterns or clusters within the data. Semi-supervised is very similar to supervised learning, however with a very important difference - the model is trained with just one class of the data. This technique is very common in anomaly detection and is explained further in Section 2.2. Last but not least, reinforcement learning deals with learning by doing. In this case, the model learns from previous action sequences that maximises a reward. A typical example is in a game where a high reward is only given if the sequence of moves are correct [1].

2.3.1 Support Vector Machines

Traditional machine learning techniques such as Support Vector Machines (SVM) are typically used in supervised learning for classification tasks. In its simplest form, an SVM aims at learning the parameters of a hyperplane in N-dimensional space (where N denotes the number of features) that best classifies the data. An example is illustrated in Figure 2.2 where points are classified by two classes represented by blue and orange points. The model, in this case, learns the parameters of the hyperplane that best segregate the data. Data points closer to the hyperplanes are called support vectors. These points are the most important for the algorithm as during parameter estimation, such support vectors contribute to the position and orientation of the hyperplane. The aim of an SVM is to find the hyperplane with the largest margin possible (denoted as $2/|w|$ in the figure) [21].

In the example illustrated, the data can be linearly separable. However, data is not always separated in with a linear model. Different kernel functions (also known as kernel tricks) can be used to map the data into a higher dimensional space in order for the

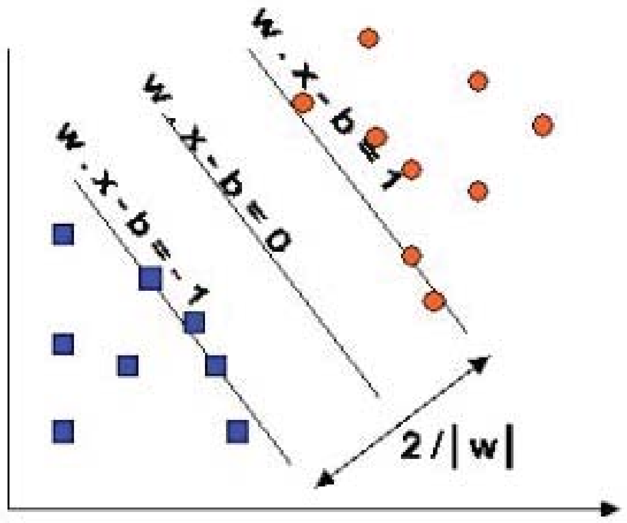


Figure 2.2: SVM: Learned hyperplane that best segregates the data
(Source: Durgesh and Lekha [21])

algorithm to find a hyperplane that can linearly separate the data. Depending on the amount of samples, this process can be computationally expensive. Some examples of kernel functions include *linear*, *polynomial*, *radial basis function (RBF)* and *sigmoid*. According to [21], the RBF kernel is the most used as it has fewer hyperparameters and less "numerical difficulties".

2.3.1.1 One-Class SVM

Originally proposed by Schölkopf et al. [73], one-class SVMs (OC-SVM) are models trained to detect outliers or anomalies. As outlined in the beginning of this chapter, outliers are typically defined as rare events that deviate from the other observations. The aim of a OC-SVM is to find a function that best describes the regions with high density of data points (similar to SVM but to one-class only). As opposed to other models, this is done without estimating the probability density of the data [73]. In anomaly detection, various studies use this unsupervised technique to classify normal and abnormal events [30; 93; 98]. Such use cases are explored in the Chapter 3.

2.3.2 Hidden Markov Models

Hidden Markov Models (HMM) and other variants have been extensively used in the field of detecting anomalous events [38; 95; 101]. Modelling time is a crucial step as most often, data in forms of speech or video contains the time dimension that makes up the whole sequence. For instance, the natural form of the data in a speech recogniser is in a sequence, so modelling all of it is important.

A HMM is a model that belongs to the Markov family. In probability theory, a Markov model is a model that describes a sequence of possible events in terms of probability. The Markov property states that in a Markov model the next state always depends on the last state. However, in a Hidden Markov Model, the state itself is hidden (not observable) but the data leading to the state is observable. Thus, in a HMM, the computed state (prediction) depends on the previous time steps states and the current observation. Such conditional independence is depicted as a graphical model in Figure 2.3. The top nodes in the graph S_t refer to the hidden state at time t whereas the nodes at the bottom Y_t refer to the observations at time t [25].

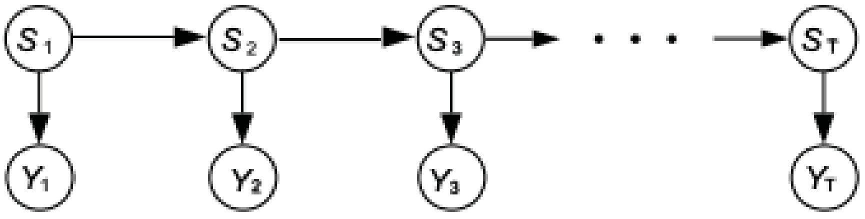


Figure 2.3: Graphical model for a Hidden Markov Model. (Source: Ghahramani [25])

2.3.2.1 HMM: Speech Recognition

One of the most popular application of HMM is Speech Recognition. At phoneme level, words are modelled statistically to represent the various sounds of the language. Since words are constructed in a temporal structure, a HMM is trained to learn the probability transition model of various words based on the observations provided - audio in this case. So in this case, the word is the hidden state and the audio features represent the observations. After the model is trained, a word is recognised by calculating the maximum likelihood of $P(word|audiofeatures)$. Here, Bayes rule[1] is used to transform the equation to: $P(audiofeatures|word)P(word)$ [22].

[1] $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$

2.3.2.2 HMM: Anomaly Detection

The goal of a HMM is usually to compute the state given a sequence. But HMMs were also proven to detect anomalous sequences. A sequence is considered anomalous when an observation is highly unlikely to happen. In such instances, a threshold is typically used for classification. For example; in a similar fashion to one-class classifiers, Görnitz et al. [27] propose an approach of detecting abnormal sequences from data by learning the probability distribution of the normal data. Use cases on this technique are explored further in Chapter 3.

2.4 Artificial Neural Networks

Artificial neural networks, which are also known as multilayer perceptrons (MLPs) or feedforward neural networks, consist of mathematical functions with the main goal being the learning of an approximate function f^* that maps an input x to output y. The term 'feedforward' refers to the fact that arrows from neurons are directed forward to the output. On the other hand, the term 'multilayer' means that the network consists of more than just the input and the output layer. Such intermediate layers are called hidden layers and denoted as h [26].

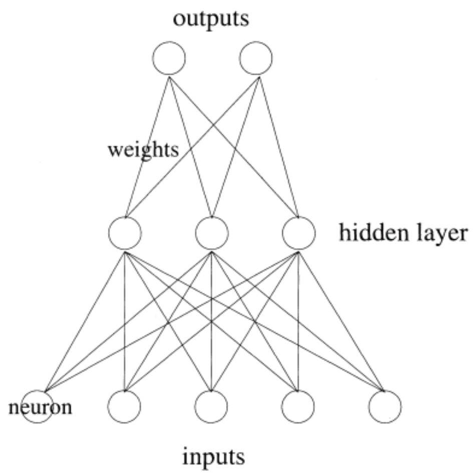


Figure 2.4: Typical Neural Network architecture
Source: Wang [92]

A typical architecture of a neural network is shown in Figure 2.4. The network consists of three layers: input, hidden and output. The nodes in the graph are usually called neurons or units and each edge connecting them contains a number called weight. During training, the network learns the values of such parameters (weights and biases) such that the best function is approximated. A loss (or cost) function is used to calculate the error between the predicted output and the actual output. The error is then minimised using Gradient Descent - which is a gradient-based method that deals with updating the weights of the network based on the error [26; 92]. A more detailed explanation is provided in this section.

2.4.1 Activation Functions

An artificial neuron within a neural network computes a weighted sum of its input and adds a bias value. The value of such neuron could range between $-\infty$ to $+\infty$. Such values could stop the network from proper training. To overcome such problem, activation functions are used. Such functions are applied to the weighted sum of each neuron and does not allow each neuron to increase indefinitely as more terms are added. Popular activation functions include *TanH*, *Sigmoid*, and *Rectified Linear Unit (ReLU)*. The output of a Sigmoid (or logistic) function is always a number between 0 and 1 whereas that of a TanH (also known as hyperbolic tangent) is between -1 and 1. ReLU's output is quite straight forward, 0 if the number is negative and the same number when the input is equal or greater than 0. One of the drawbacks of ReLU is called the 'dying ReLU' problem. This happens with neurons firing negative values and in turn having an activation value of 0. Since the gradient is 0, such neurons will never recover. To mitigate this issue, one of the variants of ReLU called Leaky ReLU allows for negative numbers to be multiplied by a fixed parameter α_i (small value such as 0.01) instead of 0 and thus allowing such neurons to recover. Since these functions are non-linear, the network is able to learn more complex non-linear functions, making it more robust in understanding different relationships within the data. These functions are also differentiable which means that an optimisation technique such as gradient descent (outlined in this section) can be used to optimise the network [26; 92].

In classification tasks, the *softmax* activation function is typically used for the last layer. Similar to the sigmoid function, the *softmax* function transforms the outputs of each unit between 0 and 1. However, each output is divided such that the total sum is equal to 1. This allows the network to output a categorical probability distribution for all n different classes. The output from such function is then used to output a prediction by finding the class with the highest probability.

2.4.2 Training a Neural Network

The algorithm used to compute the gradients of the loss (or cost) function across the network is called backpropogation. This algorithm was introduced by Rumelhart et al. [70] whereby its goal is to update the weights and biases of every layer of the network in order to minimise the loss function. Gradient descent (GD) and its variants is one of the most popular optimisers for neural networks. It deals with minimising the cost function $J(\theta)$ by computing the gradient of the cost function itself with respect to the parameters (θ). The gradient value is then used to update the parameters as defined in equation 2.1 [69].

$$\theta = \theta - \alpha \frac{\partial J(\theta)}{\partial \theta} \tag{2.1}$$

2.4.2.1 Optimising a Neural Network

A learning rate α is usually set to control the step size downward the slope (of the gradient). A large learning rate might cause the network to never converge as it may overshoot the minimum value. On the other hand, if α is too small, convergence will be much slower [69]. It is therefore desirable to tune such hyper-parameter in a way that the network converges in the shortest time possible. Smith [80] argues that training neural network models with cyclical learning rates as an alternate to fixed rates improves overall accuracy. The author reports that training the model with a very small learning rate (for e.g. 0.001) for the first few iterations and then increasing it linearly or exponentially yields good results.

One of the loss functions which is usually used for regression tasks is the mean-squared error (MSE). The error between the predicted output and the actual output is computed to calculate how 'off' the predicted value is from the ground truth. MSE is defined by equation 2.2 where N refers to the number of samples, y^i denotes the true output value and $\hat{y^i}$ denotes the predicted value. Since MSE is a quadratic function and therefore is in a form of a U-shape curve, there will be only one (global) minimum. This allows the network to optimise faster [26]. For classification tasks, a popular cost function used by researchers [53; 71; 81; 93] is called the cross-entropy or log-loss (equation 2.3). Such function measures the performance between a predicted probability value and the ground truth probability. The cross-entropy loss is high as the predicted probability diverges from the actual label. Other variants of cross-entropy are also available for multi-class classification and categorical multi-class classification [26].

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2.2}$$

$$\text{CrossEntropy} = -\frac{1}{N}\sum_{i=1}^{N}\left[y^{(i)}\log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)})\right] \tag{2.3}$$

One of the variants of the vanilla gradient descent (also known as batch gradient descent) is Stochastic Gradient Descent (SGD) [69]. Unlike batch gradient descent, SGD updates the weights for every training sample in an epoch and thus learning is performed faster. Such technique can be used for online learning too. In his paper, Ruder [69] also outlines another variant of gradient descent - mini-batch gradient descent. This method is "the best of both worlds" as it performs weight updates to mini-batches within an epoch of n training samples. Such technique allows reduction of weight updates variance. The author also lists algorithms used to improve SGD in terms of the learning-rate being adaptive. Some include: Momentum, Adagrad, Adadelta, RMSprop and Adam. Ruder also concludes that *Adam* is the preferred method overall mainly because according to [42], Adam's bias-correction and Momemtum outperforms all the techniques.

2.4.3 Underfitting and Overfitting

As outlined by Goodfellow et al. [26] in their book (Section 5.2 and 5.3), one of the challenges of a machine learning algorithm is to be able to perform well on new and unseen data. In a supervised classification task, for instance, the model should learn how to distinguish between the classes without being too closely fit to the data. On the other hand it should also capture the underlying data patterns well enough to be able to provide correct output. *Overfitting* occurs when the model fits exactly the training data, thus yielding a low error rate on the training set but a higher one on the test set. On the contrary, *underfitting* occurs when the model is not able to learn the proper function which results in high gap between the training and the test sets error values. Figure 2.5 illustrates the training and testing sets error when capacity is increased. The left end of the graph shows a high error for both sets which shows underfitting, whereas in the right end of the graph the generalisation gap is increasing (the training error is decreasing whilst the testing error is increasing). All this goes to show that the model is overfitting.

One of the techniques applied to allow the model to generalise and therefore reduce the generalisation error is called regularisation. Regularisation aims at controlling the

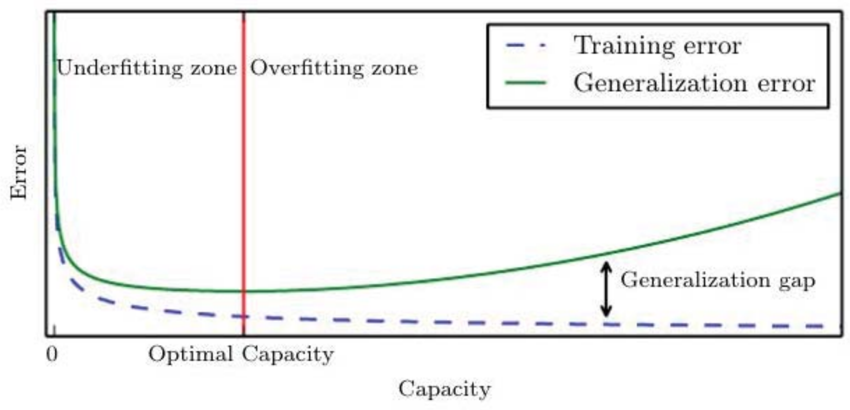


Figure 2.5: Under and Overfitting
Source: Goodfellow et al. [26]

model in a way not to be too complex when selecting weight values. Examples of such methods include the L^1 (Least Absolute Deviations) and L^2 (Least Square Errors) regularisers. A regulariser is added to the cost function and acts as weight decay. Most of the times, the L^2 regulariser is preferred as it decreases the weights of insignificant features to 0 [26].

Another technique outlined by Goodfellow et al. in their book is to have a separate validation set (apart from the training and testing sets) that will be used to tune the hyperparameters of the model. The hyperparameters of the model are important to tune because they control the 'capacity' of the network. The usual split is 80-20 where 80% would be for training the model, and (20%) for evaluating the performance of the model. 20% of the 80% is typically used for validating the model and tuning the hyperparameters. In small datasets, cross-validation (for e.g. *k*-fold cross validation) is typically used in order to prevent statistical uncertainty.

Early Stopping is another popular technique outlined in [64] to reduce overfitting. This deals with stopping the network from learning with the purpose of stopping it from overfitting. Prechelt also argues that validation error curves usually contain more than one minimum as it is not always a smooth curve. In view of this, the author outlines that a balance between training time and validation error needs to be achieved.

Dropout regularisation is another technique proposed by Srivastava et al. to prevent overfitting. Its main idea is to randomly 'switch off' a number of neurons (typically 50%

in a given layer) in order to force other neurons to learn, thus preventing the main neurons from taking over. In their paper, the authors report that this method improves the performance of supervised learning tasks in speech recognition and computer vision, amongst others [82].

2.5 Deep Neural Networks

Results reported by researchers in the field show that the use of deep neural networks and its variants outperform the traditional techniques (such as SVM and HMM) [43]. Recent studies also show that convolution neural networks and similar architectures have established themselves as very powerful techniques in learning useful representations from raw data without the need for hand-crafted features [33; 44; 76; 86]. In this section, an overview of deep neural networks and how they differ from vanilla artificial neural networks is given.

Due to advancements in computer power (thanks to graphical processing units and their ability to handle matrix calculations in parallel), deep models can be created and trained. The term ‘deep’ refers to the increase in the number of hidden layers within a neural network [26]. Deep architectures allow the model to learn both higher and lower feature hierarchies. A typical ‘non-deep’ neural network might have 1 to 3 layers of neurons however a deep neural network might comprise of 11 or more layers as reported in the VGG paper [79]. In 2014, a deep network consisting of 16 or 19 layers was considered very deep, however recent architectures such as a Residual Network (ResNet) [34] can contain hundreds (or thousands) of residual layers and still perform very well whilst outperforming other architectures.

2.6 Convolutional Neural Networks

The era of deep learning evolved in 2012, when Krizhevsky et al. trained a deep convolutional neural network (CNN) to classify a huge amount of image data into 1000 classes in the ImageNet competition[2]. At that time the authors claimed that their error rates were much lower than the previous state-of-the-art. The network that they used to train the model included 650K neurons with 5 convolutional layers [45].

[2]`http://www.image-net.org/challenges/LSVRC/`

In anomaly detection, various researchers make use of CNNs in conjuction with other networks (outlined in Chapter 3) to extract meaningful features for video frames and be able to classify a sequence accordingly [4; 30; 40; 100].

The increase in computing power offered by GPUs and the huge amount of labelled image data has contributed to the increase in popularity of deep CNNs. As outlined in [50], a CNN is made up of a number of layers. A layer is a series of operations being applied to an input. In a typical neural network, every neuron is usually connected to every other neuron, forming a fully connected layer. However, in the case of CNNs this is not always so. An image of 128x128 pixels would result in having a neuron with 16,384 parameters in order for it to learn which is not efficient. In order to overcome this problem, each neuron can be connected to a few input neurons. For instance, in an image classification task, each neuron in the network can be configured to look at neighbouring pixels only rather than the whole input. Such network is inspired by the fact that neurons in a human brain are also locally connected [48].

CNNs have been around for quite some time. In [49], LeCun et al. introduced an end-to-end system capable of recognising images. In recent computer vision work, researchers are getting more promising results with CNNs due to possibility of training deeper networks [34; 79]. Furthermore, more efficient CNNs are now being trained to detect and localise objects in a given frame in real-time [37].

CNNs are also capable of recognizing objects when they presented in a different way (for e.g. different positions or smaller/larger appearance). This refers to the term *translation invariance* and is possible due to the fact that neuron connections (weights) are shared with other neurons, making the network even faster to train. Figure 2.6 illustrates weight sharing in a basic CNN architecture, where the first set of input neurons (3) is connected to the first neuron, the second set connected to the second neuron and the third set connected to the third neuron. The figure also shows the neuron connections w_1 to w_9 where the colours indicate same values (i.e. $w_1 = w_4 = w_7$ ...). Such set of weights is usually called a 'filter'. Therefore in a CNN, each region of the input is applied the same filter which in this case allows the network to only learn and keep w_1, w_2 w_3 [48].

A typical input to a CNN is an image in its raw data which consists of 3 channels - width, height and depth (usually Red, Green and Blue). For example, a $64x64$ image would be an input of $64x64x3$ matrix. A typical architecture of a CNN looks like the

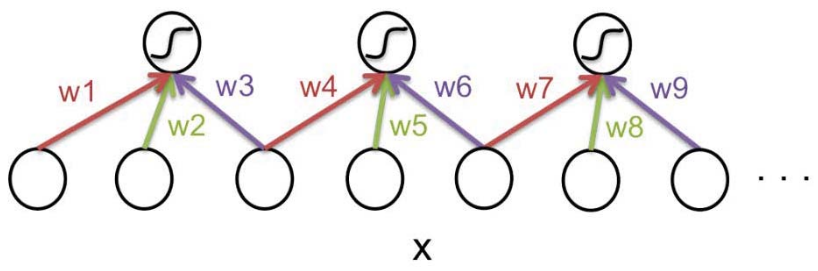


Figure 2.6: CNN Weight Sharing
Source: Le et al. [48]

following:

$$INPUT - CONV - RELU - POOL - FC$$

The CONV layer will perform convolution in two-dimension to regions of the input whereby each region of the image will be multiplied (dot product) by a filter containing a set of weights (also known as feature map) determined during training. If 12 filters are used, the result would be $64x64x12$. The next layer is ReLU – this is an activation function (outlined in Section 2.4) that simply transforms negative values to 0 and keeps positive values. The POOL layer will then reduce the dimensions of the input to $32x32x12$. Max pooling is usually used whereby the maximum number in each 'depth slice' will be picked. In order to create a deep convolutional neural network, these layers (CONV, ReLU and POOL), which are called convolutional blocks, are added to the network as another block layer after the POOL layer. The final layer in a typical architecture is the fully connected layer, which is responsible for computing the class probabilities. The *softmax* activation function is typically used at this level to output probabilities in a classification problem [50].

Backpropagation, which was outlined in Section 2.4, is also used to train a CNN. Stochastic gradient descent together with an adaptive-learning method such as Adam [42] is commonly used to update the weights at each layer. This process is referred to as 'learning' [50].

2.6.1 1D Convolutions

Since images in nature are 2-dimensional, many papers in computer vision focus on 2D CNNs. Just like 2D CNNs, 1D CNNs are useful when the data is in one dimensional. The use of 1D CNNs was recently explored by Kiranyaz et al. [44] and proven to work

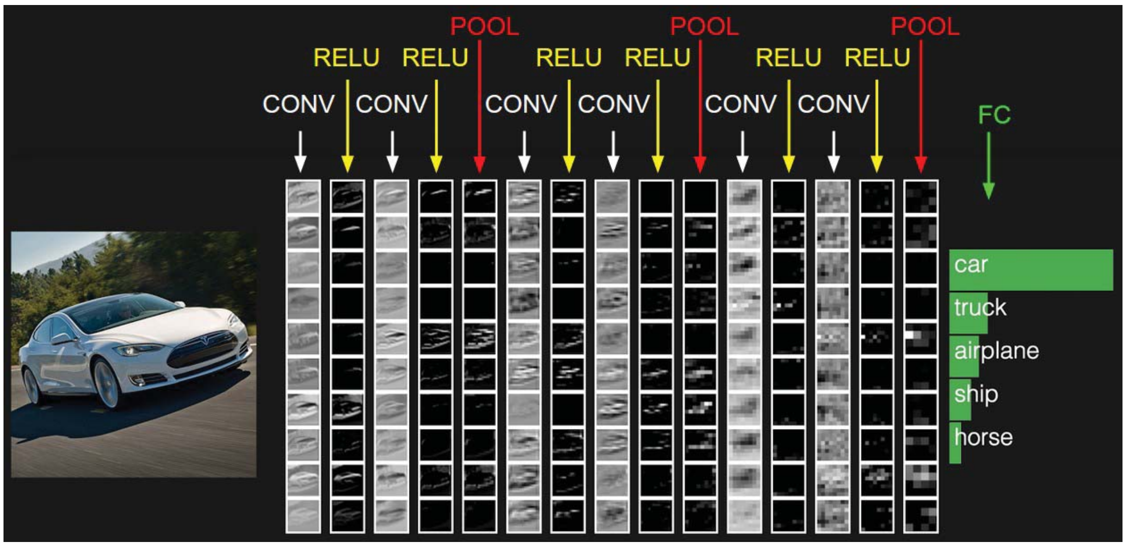


Figure 2.7: Deep Convolutional Neural Network Architecture showing activations at each layer together with the final output
Source: Li and Karpathy [50]

well with extracting discriminative features from sequences such as electrocardiogram signals. The researchers trained a 1D CNN with large 1D filter kernels for every patient and used the model for classification and detection of anomalous signals. They also point out that the speed and accuracy of such architecture make it a natural choice for detecting anomalous signals within the data because it can be powered by mobile devices [44]. Lim et al. [51] also explored the use of 1D CNNs to detect rare sounds. They used a 1D CNN for extracting frame-level features from a spectrogram followed by two LSTM layers to capture the temporal dimension of the features [51].

Another study which uses 1D ConvNets is presented by Zeng et al. [99]. They explore the use of 1D ConvNets on Human Action Recognition (HAR), specifically on sensor data captured from smartphones to automatically acquire key features that contribute to recognising activities. They show that their model is able to capture both local dependency and scale invariance of a signal as shown in other domains such as image and speech recognition. In a more recent study Cho and Yoon [13] present a 'Divide and Conquer' approach to classifying HAR. The authors specifically make use of 1D CNNs to create a two-stage modelling approach. The first stage deals with recognising whether the action is *dynamic* or *static*. Afterwards, individual activities are classified using 2 3-class classifiers.

2.7 Recurrent Neural Networks

A Recurrent Neural Network (RNN), unlike a vanilla NN, is able to understand context because it 'remembers what happened before'. In view of this, many researchers chose to work with RNNs for modelling their anomaly detection methods. RNNs were also proven to work effectively for sequence-based anomaly detection [4; 54; 55; 81].

A vanilla neural network accepts input as a fixed vector where it is processed by a set of hidden layers and finally an output is computed. In this case, the model ignores the previous input and therefore it is not able to model sequential data. In certain applications such as speech, language and video analysis, data is in a form of a sequence and as such, it needs to be fed to the model in its original form. A recurrent neural network is able to model such data sequence that is often in time steps. A recurrent neural network is able to do so as the hidden layer h at time t is determined by the input x_t and the hidden output of the last time step h_{t-1}. An RNN is also able to model variable-length sequences due to the fact that the parameters for the hidden state are shared [52]. The following equation specifies the calculations of a recurrent neural network for computation at every time step t.

$$h^{(t)} = \sigma(W^{hx}x^{(t)} + W^{hh}h^{(t-1)} + b_h) \tag{2.4}$$

Where W^{hx} represents the weight matrix between the input and the hidden layer, W^{hh} represents the weight matrix between the hidden layers, b holds a vector containing the bias parameters and σ refers to the activation function, typically the logistic sigmoid function or $tanh$.

Figure 2.9 shows an unfolded recurrent neural network across two timesteps (t_1, t_2) where the neurons at the hidden layer at time t_2 are determined by those of t_1.

Similar to RNNs, HMMs, are able to model an observed sequence of data. However, when compared with each other, RNNs are able to use the hidden states in a more efficient way. Furthermore, unlike HMMs, RNNs are able to handle long-term dependencies and thus overcoming the limitation of Markov models where a hidden state only depends on a previous state [52].

2.7.1 Long-Short Term Memory

Long-Short Term Memory (LSTM) as defined by Lipton [52] is a special type of RNN with its main advantage being the ability to handle long-term dependencies better as

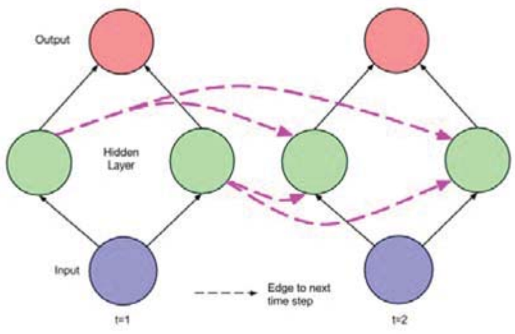


Figure 2.8: Unfolded Recurrent Neural Network
Source: Lipton [52]

well as to overcome the vanishing or exploding gradients problem.

As outlined by Pascanu et al. [62], the *vanishing* gradient problem occurs during the training of the model. When the gradients get very small there will be little to no improvements at all to the parameters of the model and this causes the network to stop training. This happens during backpropogation, when the gradients are back-propogated through the all hidden states. The gradient will be multiplied to the weight matrix over and over and eventually this will lead to a very small gradient signal until it is 0. On the contrary, if there is a large increase of the gradient, the network could suffer from the *exploding* gradient problem. The authors of the paper propose two methods of how these problems could be dealt with - to deal with the *exploding* gradient problem they propose a technique called gradient clipping whose its job is to clip the gradients between two numbers in order to prevent them from getting too big. On the other hand, to overcome the *vanishing* gradient problem they propose a regularisation term that forces the gradients not to vanish during the training of the network (back-propogation) [62]. In LSTMs, the vanishing gradient problem was overcome with the introduction of a gated architecture [26; 52].

The recurrent module in an LSTM is similar to that of a RNN but with a different structure. Instead of having a single neural network layer, it is made of four - where each interact together in order to preserve only the useful information. Introduced by Hochreiter and Schmidhuber [36] and further enhanced by Gers et al. [24], an LSTM contains a memory cell, usually denoted as c which contains the cell state. Data in the LSTM network passes through the forget f, input i and the output o gates. Such gates act like filters - allowing or limiting data to pass to or from other gates. Output from such gates will then update the memory cell c. The input i, forget f and output o gates make use of the logistic sigmoid function to output values between 0 and 1 whilst the vector that holds the internal state for c uses the *tanh* function [52].

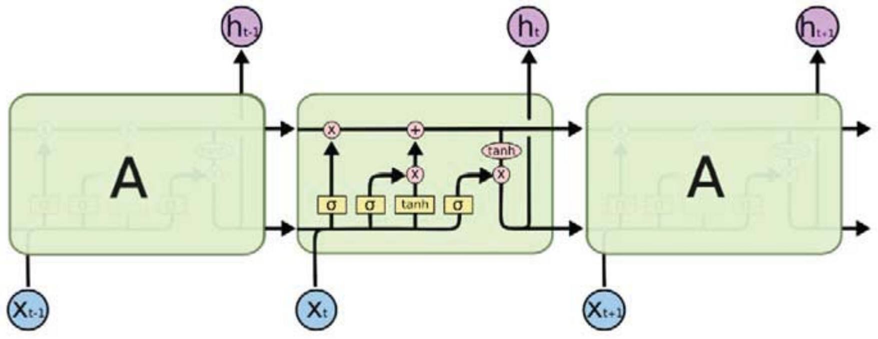


Figure 2.9: Unfolded LSTM Network
Source: Olah [58]

2.7.1.1 Gated Recurrent Units

A simpler but similar LSTM-variant called Gated Recurrent Unit (GRU) was proposed by Cho et al. [14]. The authors combined the input and forget gates into one, calling it the 'update' gate. Such gate controls how much information from the previous hidden state should be kept. The other gate is called the 'reset' gate - this controls how much information from the previous time-steps should be removed from the hidden state. They also got rid of the cell state making it simpler and more efficient computation wise [14].

Apart from GRU there are other LSTM variants. However a study conducted by Greff et al. [28] shows that the vanilla LSTM architecture performs very well on various tasks like handwriting and speech recognition. None of the variants improves performance significantly.

2.8 AutoEncoders

Autoencoders were first introduced by [70], aiming to reconstruct the input data at the output layer. An AutoEncoder is a fully connected (FC) neural network which contains one or more hidden layers. The main job of an AutoEncoder is to learn an approximation of the identify function by compressing the input layer x into a latent-space representation z (encoding) and then reconstruct the output from such representation (decoding). The encoder forces the hidden layer to learn good representations of the inputs. Figure 2.10 illustrates a four input and output AutoEncoder with one hidden layer consisting of two neurons. In this case, the network is forced to compress the input to two neurons and then to reconstruct the original output. Since the model is trained to reconstruct the input data (hence the input and output data are the same), AutoEncoders are classified as unsupervised learning methods and thus no labelling is required.

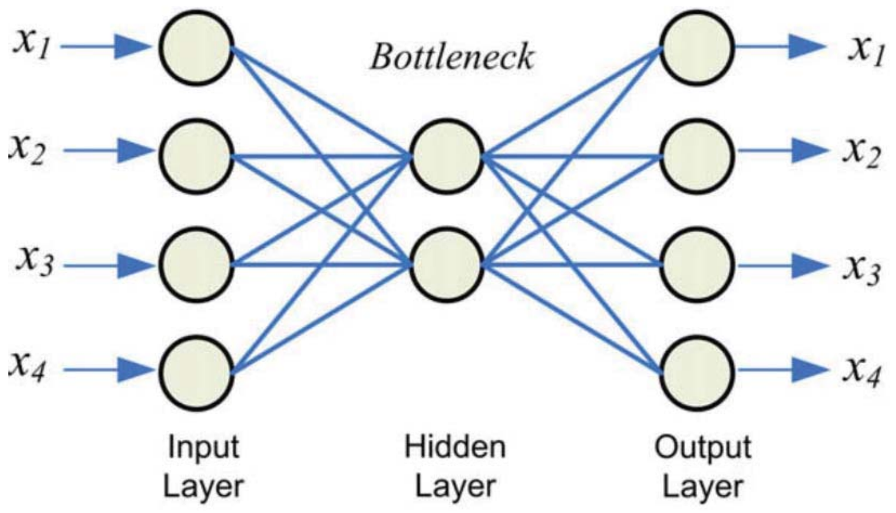


Figure 2.10: Structure of an AutoEncoder
Source: Nelwamondo et al. [57]

The difference between the input layer x and output layer y (reconstructed input) is called the reconstruction error. The main goal of the AutoEncoder model is to minimise the reconstruction error and therefore be able to reconstruct the input as close as possible. Typically, the cost (or loss) function J used to calculate the reconstruction error is mean-squared error (equation 2.5) [43]. Furthermore, to minimise the cost function, the backpropogation algorithm is used where stochastic gradient descent or its variants are typically used [2].

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}(x_i - x_i')^2 \tag{2.5}$$

2.8.1 AutoEncoders: Dimensionality Reduction

AutoEncoders can be used to convert high-dimensional data to low-dimensional data, forcing the network to learn good representations of the data [35]. Such technique facilitates data communication, visualisation and storage. This is similar to Principal Component Analysis (PCA), which is a technique used in statistical analysis to identify and extract patterns from a dataset. However, unlike PCA, AutoEncoders are not restricted to learning linear relationships only. This is because non-linear activation functions such as Sigmoid or ReLU can be used to help the network generalise with the data and introduce non-linearity [43]. In their paper, Hinton and Salakhutdinov [35] show that deep AutoEncoders are very effective for non-linear dimensionality reduction and show that such technique can be applied to large datasets as well.

Researchers also discovered that AutoEncoders can be used as an initialising method to 'pretrain' deep neural networks [7]. One of the main goals of pretraining a neural network is to overcome the generalisation problem of unseen data by pretraining one layer at a time.

2.8.2 Denoising AutoEncoders

AutoEncoders can also be trained to remove noise from data (e.g. images), a technique which is also referred to as 'denoising'. With specific regard to images; as input, a denoising autoencoder is fed with images that were tampered with noise. The model is then trained to reconstruct the original (*denoised*) image. An AutoEncoder will be able to deal with noise by learning meaningful specific characteristics [90]. Other use cases for denoising autoencoders include audio, signal or document denoising.

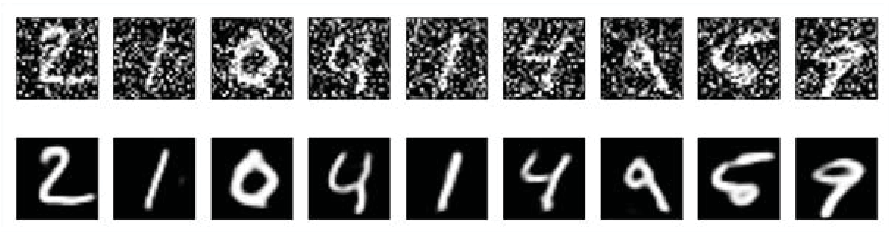


Figure 2.11: Denoising AutoEncoder. Top: Noisy images fed to the model, Bottom: Reconstructed by the model
Source: Chollet [15]

Figure 2.11 illustrates the noisy input fed to the model (top row) and the final reconstruction output by the same network after training. The network was trained with synthetic noisy MNIST digits that were generated by applying Gaussian noise. A stacked Convolutional AutoEncoder with 32 filters in each convolution block was then trained to reconstruct the noisy images to the original images [15].

2.8.3 Variational AutoEncoders

As outlined by An and Cho [2], a Variational AutoEncoder (VAE) is a probabilistic encoder-decoder that is able to generate new data samples of the same type the model was trained on. Unlike general AutoEncoders, where their main goal is to learn an arbitrary function, VAEs learn the parameters of a probability distribution that represent the data. Instead of forcing the network to learn a latent representation of the data, VAEs are trained to learn the latent distribution which consists of the mean and variance of the data. For calculating the reconstruction loss, the mean-squared error is typically used. This is combined with the Kullback-Liebler divergence to act as a regularisation term and therefore 'forces' the latent values to be sampled from a normal distrubtion [2].

2.8.4 AutoEncoders: Anomaly Detection

Apart from the various uses outlined in this section, AutoEncoders can also be used for detecting anomalous events or data. An AutoEncoder is expected to output a low reconstruction error if a normal event is fed to the network. On the other hand, it should output a high reconstruction error for an abnormal event which implies that the abnormal event is not 'similar' to the normal events trained by the model [2; 43; 55; 100].

A threshold is typically defined to detect abnormal events. After training the model, unseen data points can be checked for abnormality. An abnormal instance is naturally poorly reconstructed by the model where an anomaly threshold score is used to detect whereby the point should be classified as normal or abnormal. Furthermore, since the data is reconstructed with the low dimension representations, only the interesting features are retained and thus noise and features that do not contribute are automatically discarded. Use cases on this technique are explored further in Chapter 3.

2.9 Conclusion

This chapter aimed to provide the reader with an overview of machine learning techniques related to detecting anomalies in various domains. In particular; an overview of SVMs, HMMs, ANNs, CNNs, RNNs and AutoEncoders was presented.

In recent work, researchers discovered that deep neural networks such as deep convolutional networks, AutoEncoders and other variants achieved remarkable performance on learning difficult tasks and outperform the traditional methods such as SVM and HMM [13; 43]. In particular, LSTMs, when compared to HMMs or vanilla RNNs have shown excellent performance in learning sequence based data [55] as the architecture is capable of learning long-term dependencies without suffering from vanishing gradient problems [52]. AutoEncoders have also proven to be very effective in learning a compressed representation of the data in complex datasets as they are able to also learn non-linear relationships [35]. Furthermore, they can be used to train semi or unsupervised models which is useful in anomaly detection where the anomalous data is scarce or expensive to acquire. Recent studies also show that CNNs and similar architectures have established themselves as very powerful technique in learning useful representations from raw data without creating hand-crafted features [33; 44; 76; 86].

The next chapter aims at providing an overview of the literature related to anomaly detection techniques employed in sequence-based data (such as videos and time-series data) where the temporal aspect is important.

3

Overview and Related Work

3.1 Introduction

A video consists of a series of images each representing a time step of an event. In video understanding, time is an important aspect and therefore appropriate networks that can incorporate such dependency have to be used. In this chapter, an overview of literature relating to human activity detection as well as sequence-based techniques related to anomaly detection is provided. This chapter also provides an overview on a technique called Pose Estimation which allows key body joints such as the hands and hips to be estimated from an RGB image.

Several works in anomaly detection [3; 16; 30; 43; 84] argue that obtaining a labelled dataset is difficult and expensive mainly as a group of human experts is required to manually annotate the data. In view of this, they choose to go for semi-supervised or unsupervised techniques.

In their paper, Kiran et al. [43] present an overview of deep learning techniques for both unsupervised and semi-supervised anomaly detection in videos. They state that both unsupervised and semi-supervised have become well-established in anomaly detection. In this case, the main difference between unsupervised and semi-supervised is important. The semi-supervised learning problem deals with data from one class only - in this case the normal class (free from abnormalities). On the other hand, in unsupervised anomaly detection, the model is trained with both normal and abnormal data however with the assumption that normal instances are much higher than the abnormal ones.

In domains where labelled data is available, models can be trained in a supervised manner. In this case, the dataset in question should have instances for both the ’normal’ and ’abnormal’ classes. A model is trained on both classes and any unseen instance is then classified as ’normal’ or ’abnormal’ [10].

3.1.1 Model bias

An experiment undertaken by Ribeiro et al. shows that a classification model trained on images containing wolves and huskies predicts ‘Wolf’ when there is snow (or a light background) in the picture and ‘Husky’ otherwise. Figure 3.1 depicts the raw data fed to the model (a) and the explanation of the prediction from the classifier (b). After this experiment, the authors presented a balanced set of 10 predictions to a number of subjects without explanations of huskies and wolves images. One image of a wolf did not contain a snowy background and as such was predicted as a ‘husky’. Another image of a husky did contain a snowy background and so it was predicted as a ‘wolf‘. Showing just the model predictions 37% of the subjects trusted the model. Afterwards, the same subjects were shown the explanation of the model and the number of people who trusted the model dropped to 11%.

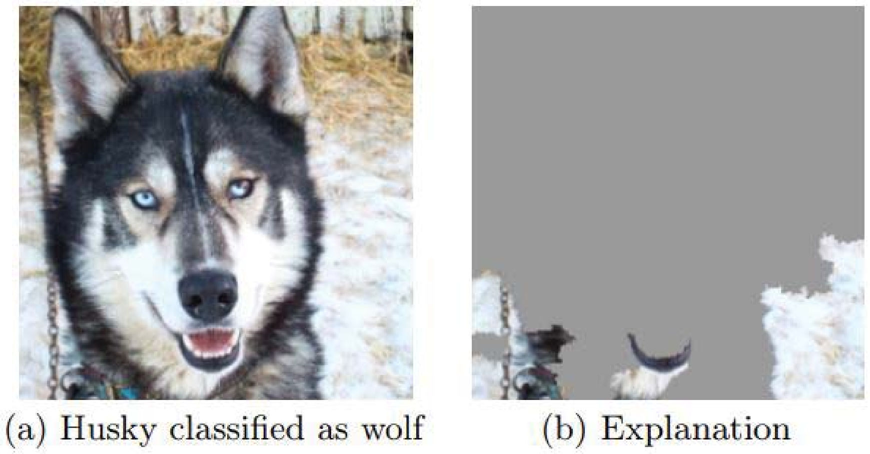


Figure 3.1: Raw data and Explanation of the ‘Wolf vs Husky’ experiment
Source: Ribeiro et al. [67]

This experiment shows that since the model was fed the whole raw picture (including the background), the model picked the snow (or the background) as the most determining feature to classify a wolf. The model would have probably behaved differently if only the raw data related to the dogs (i.e. without the background) was fed. With reference to this study, training the model with just a representation of a human being from a whole frame allows the model to be trained solely on the human action being performed. This ensures that the background itself or objects visible in the scene do not contribute to detecting abnormalities.

3.2 Abnormal Activity Detection

In more recent work in video anomaly detection, researchers have used deep learning methods such as CNNs in conjuction with other networks such as LSTMs and AutoEncoders. The area of anomaly detection used to be computed using hand crafted features. However, videos consist of non-linearity and more efficient methods are required for achieving better results. As a result, deep learning-based techniques have been employed with the main objective being improving the end results [86].

In human activity detection, for example, hand-crafted features requires domain expertise. Statistical features such as mean and variance used to be the features fed into a HMM [38; 95; 101] or an SVM [98]. However, recent studies show that CNNs have established themselves as very powerful techniques in learning useful representations from raw data [13; 33; 44; 51; 76; 86].

Zhao et al. [100] proposed a Spatio-Temporal AutoEncoder for detecting anomalies in videos that is able to model both spatio and temporal features of a video using a 3D convolution neural network also known as a 3D ConvNet. As input data, construct a temporal cuboid using a sliding window of 16 frames and resize each frame to $128x128$. In order to detect abnormal instances from a video sequence they train an AutoEncoder (after extracting spatio and temporal features from the 3D ConvNet) and use the reconstruction error as a measure to detect anomalous events. Specifically, they use the Euclidean loss defined as:

$$L_{rec} = \frac{1}{N} \sum_{i=1}^{N} \| X_i - f_{rec}(X_i) \|_2 \tag{3.1}$$

Source: Zhao et al. [100]

where X_i is the spatio-temporal instance and $f_{rec}(X_i)$ is the output of the AutoEncoder.

Furthermore, they design a prediction branch in the decoder in order to predict future frames. They use the same Euclidean loss for the prediction loss, however they propose a weight-decreasing method to the loss in order to cater for objects that gradually increases as time goes by. To classify events as normal or abnormal, the authors use the reconstruction errors of their training data to compute a 'regularity score'. Their method was evaluated on the UCSD Pedestrian dataset (both Ped1 and Ped2), CUHK Avenue and a Traffic dataset which the authors collected. To evaluate and compare their work they used the Area Under the Curve (AUC) and Equal Error Rate (EER) from the receiver operating characteristic (ROC) curve. Such graph is produced by varying the anomaly score threshold. Compared with other research, their method outperformed state-of-the-art approaches in all datasets presented.

Xu et al. [93] presented similar research to the above however they propose a different architecture. They use a stacked denoising auto-encoder deep network in order to extract feature representations for motion and appearance in videos. In order to predict the anomaly score, they trained multiple one-class SVM models. As opposed to Zhao et al. [100]'s work, the authors of this paper extracted image patches and optical flow patches and as a result discarded the spatial information of the video. Gutoski et al. [30] present a Convolutional AutoEncoder video anomaly detector. Using the reconstruction error as output from the Autoencoder and a one-class SVM for classification, the authors report that their work produces acceptable performance. Baccouche et al. [4] proposed a deep model that is able to classify sequences of human actions into one action. Similar to the works in [44; 71; 76] the authors trained a CNN to extract spatial and temporal features from a video. Using these features, they trained a recurrent neural network in order to the action being undertaken.

In their paper, Simonyan and Zisserman proposed a two-stream convolution neural network that combines spatial and temporal information retrieved from videos. The first network (spatial stream) is trained on the raw images of the videos where it sees just one video frame at a time whereas the second network (temporal stream) is trained on the motion of the video – i.e. the optical flow of the frames within a video [78]. In more recent work, Peng and Schmid [63] outperformed the current state-of-the-art by proposing a multi-region two-stream R-CNN model for human action detection in videos. Their method, unlike Simonyan's was based on Faster R-CNN, two-stream CNNs (with optical flow) and multi-region CNNs [63].

3.2.1 Skeleton-based approaches

Skeleton-based human representations have gained a lot of attention recently [31; 53]. Thanks to their robustness to variations of different viewpoints and human body scale, space-time features can be extracted to generate human representation. Whilst 2D visual data can be enough to generate a human posture, 3D data allows for additional depth information [31]. On the other hand, the low dimensionality of 2D data allows for a simpler and faster learning process [56]. Trained models that are able to estimate pose or skeleton data can be used to automate feature extraction for video data [43].

Recent work by Song et al. [81] presents a human action recognition model based on spatio-temporal 3D skeleton data using RNNs with LSTM gates. The authors extract the position of human joints from a set of given frames and use them as input to recognise human actions. Furthermore, they proposed both spatio and temporal attention mechanisms on the extracted joints to capture discriminative joints for certain actions. Their model managed to achieve remarkable performance when compared to other state-of-the-art techniques. Other work by Ji et al. [40] shows the use of 3D CNN for action recognition which is also similar to the work in [100].

Liu et al. [53] also explored the use of 3D skeleton joints for recognising human actions. In their paper they show that CNNs "can lead to state-of-the-art action recognition performance using skeletal time-series data alone". The authors argue that in CNN architectures, the up-sampling process to images can add noise to and therefore propose an atomic visual unit called *Skepxel* that constructs skeletal images ready for processing. In their experiments, they use 3D joints from existing datasets and structure the data into 2D grids in order to take advantage of the 2D kernels in CNNs. Their method, which is claimed to have outperformed existing results, was evaluated on three standard datasets for human action recognition namely the NTU RGB+D Human Activity Dataset [75], the UTD Multimodal Human Action Dataset [11] and the Northwesthern-UCLA Multiview Dataset [91].

3.3 Pose Estimation

In a recent study, Papandreou et al. proposed a technique for detecting multiple persons in a frame together with a 2-D pose estimation of each person. This can be done from an RGB image without the need for a depth sensor. Using a 2-stage top-down approach, first they predict the bounding boxes likely to contain human beings and then, for each bounding box (human), the model estimates the positions of 17 human body keypoints (12 body joints and 5 face landmarks). Some examples of extracted keypoints include face features (eyes, nose and ears), elbows, shoulders, hips and ankles. The Faster R-CNN method [65] using the ResNet-101 [34] CNN architecture was used to detect people from a given frame whereas a fully convolutional network was used to predict the keypoints. They also trained their model using the MobileNet architecture [37] in order to allow for real-time processing. The models were trained using the COCO keypoints dataset[1] as well as an additional in-house labelled dataset. They managed to achieve an average precision (AP) of 0.685 on the *test-dev* set. The authors report that their model is capable of performing well even in heavily cluttered scenes. Furthermore, it is also able to predict keypoints of occluded areas [61].

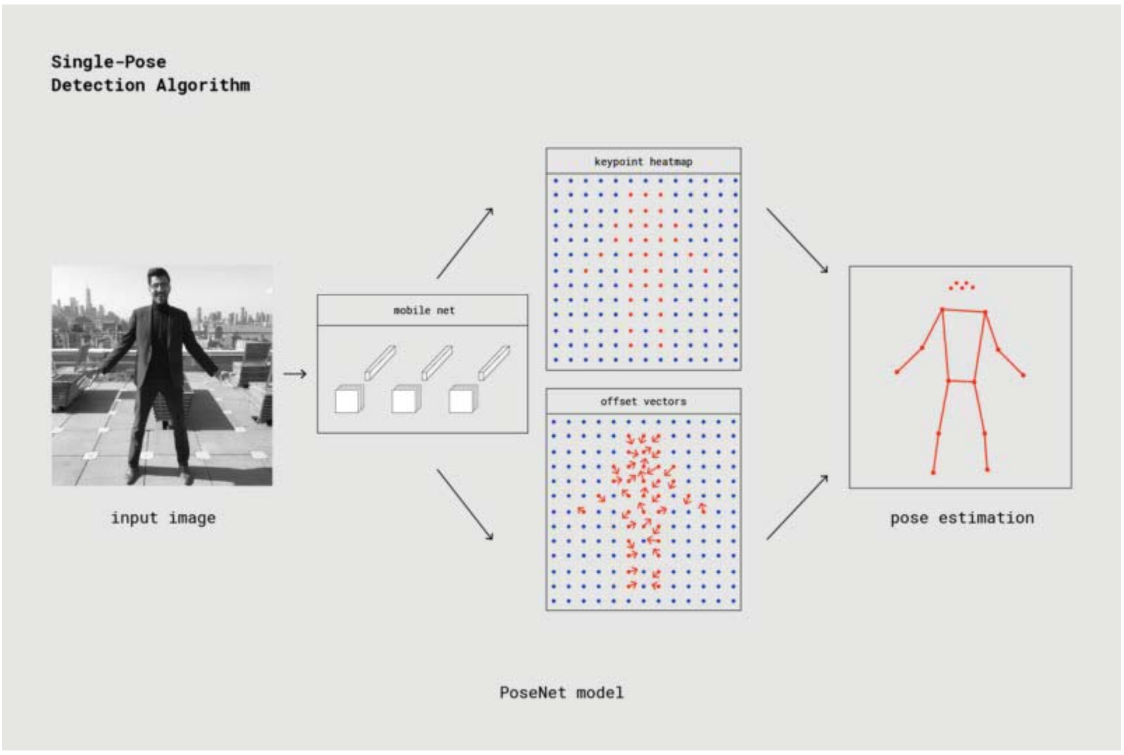


Figure 3.2: PoseNet - A pose estimation algorithm that detects multiple persons in a frame and outputs an estimation of 17 key body joints.
Source: Papandreou et al. [61]

Figure 3.2 depicts a high-level architecture for estimating the various keypoints of a person's pose. As already outlined above, in the first step the authors used a Faster

[1]`http://cocodataset.org/#keypoints-2017`

R-CNN architecture to detect persons in a given image. The second step deals with cropping each person from the image and estimate all 17 keypoints. This is done for all persons detected in the frame. Since the detector needs to recognise humans only, the Faster R-CNN detector was trained on the *person* category only from the COCO dataset. The other 79 categories were not required and were therefore ignored. In order to estimate the overall pose, the authors combined a classification and regression approach. For every spatial position, they first produce a heatmap to classify whether it is close to any other keypoints or not. Afterwards, a 2D offset vector for each position is predicted independently in order to get a precise keypoint location. Figure 3.3 shows the network target outputs for one keypoint only namely the left-elbow keypoint. The left and middle images show the heatmap target whilst the right image shows the offset field L2 magnitude (in grayscale) and the 2D offset vector (in red). The whole process is depicted in Figure 3.4. The authors report that predicting separate heatmaps and offset vectors together with aggregating them in a weighted voting process pinpoints the individual keypoints in a precise manner [61]. In their paper, the authors also claim that their revised model achieved better higher accuracy than the work of [9].

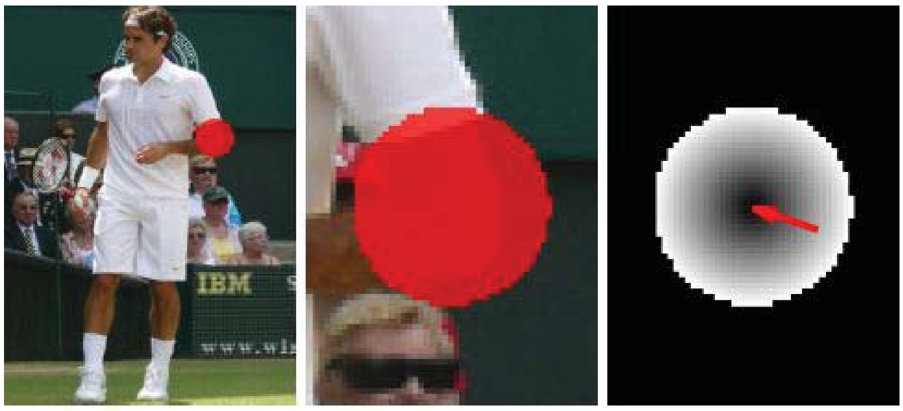


Figure 3.3: Network target outputs
Source: Papandreou et al. [61]

The team at Google released their model codenamed *PoseNet* in a form of TensorFlow.js[2] library. Such library allows use of the pre-trained model for inference purposes. The output of the model consists of a vector of 17 body keypoints (x and y values) together with a confidence score for each keypoint (for every person).

A similar multi-person 2D pose estimation model was proposed by Cao et al. [9] in their paper titled "Realtime Multi-Person 2D Pose Estimation using Part Affinity

[2]`https://js.tensorflow.org/`

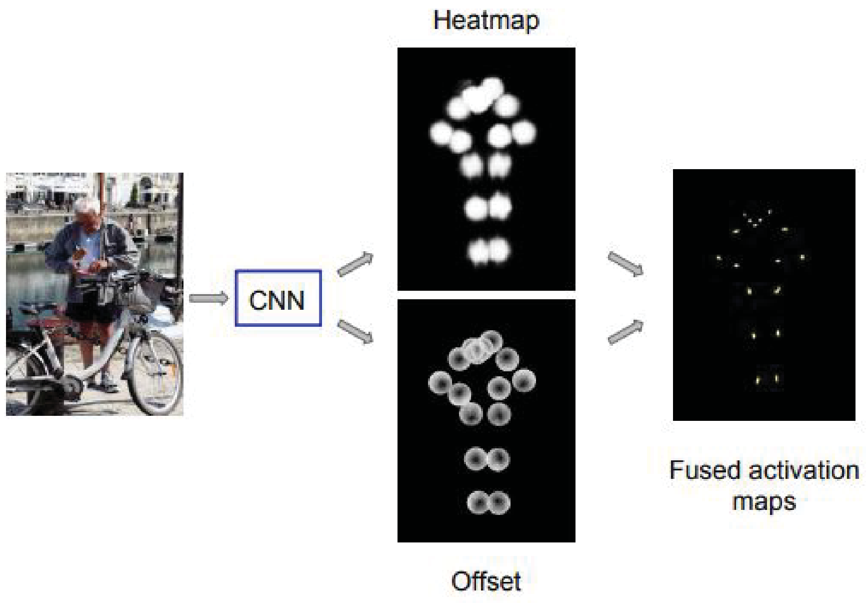


Figure 3.4: Super-imposed network outputs from all keypoints
Source: Papandreou et al. [61]

Fields". Unlike the approach in [61], the authors propose a bottom-up approach instead. They argue that systems that employ a top-down approach can fail if the first step fails. In this case if the person detector fails to detect any persons in a given frame, there is no way to recover. Furthermore, they argue that the computational cost of top-down approaches is proportional to the number of persons in the frame - i.e. the more people in the frame the higher the cost. Cao et al. [9] propose a novel feature representation called Part Affinity Fields (PAF). Essentially, PAF is comprised of a set of vectors that encodes information (location and orientation) about body limbs. The overall pipeline of the method is depicted in Figure 3.5. A two-branch CNN model takes an input image (a) and predicts both the body part confidence maps (b) as well as the part affinity fields (c) in a simultaneous manner. Bipartite matching is then performed in order to associate body parts to the correct candidates (d) followed by the last step which joins all body parts for all persons in the frame into full body poses (e). Despite the fact that the model fails in rare poses or overlapping parts, results show that the model is able to provide real-time performance for multiple people in images. Furthermore, they conclude that a greedy parsing algorithm is able to estimate high quality body poses even as the number of persons in the frame increases.

The authors also released their pre-trained models on their GitHub[3] for public use

[3]`https://github.com/CMU-Perceptual-Computing-Lab/openpose`

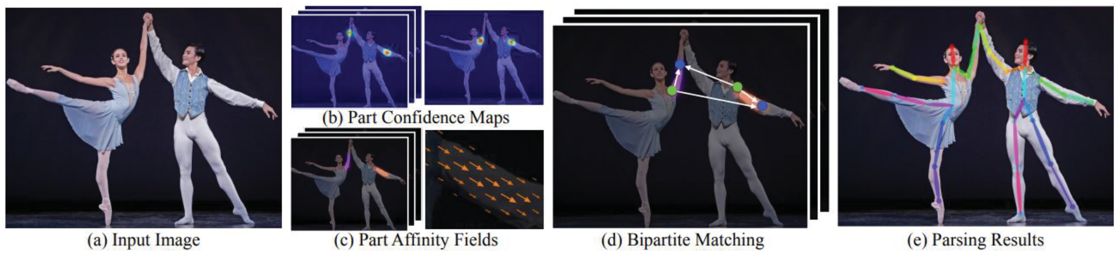


Figure 3.5: 2D Pose Estimation using Part Affinity Fields: Overall pipeline
Source: Cao et al. [9]

and packaged their work in a Python API and a CLI tool.

3.4 Encoder-Decoders for Anomaly Detection

Using a Variational AutoEncoder (VAE), An and Cho propose a way of detecting anomalous data points. In their experiments, they used the MNIST and KDD network intrusion dataset to determine the effectiveness of using the reconstruction probability from a VAE. They claim that their proposed method outperforms the PCA and the general autoencoder methods [2].

In their paper titled *'Unsupervised Anomaly Detection via Variational Auto-Encoder for Seasonal KPIs in Web Applications'*, Xu et al. [94] trained a VAE in an unsupervised manner that is able to detect anomalies in KPIs (Key Performance Indicators) such as number of page views, orders and subscribers. Since VAEs are not ideal for sequence-based data, the authors use a sliding window over their time-series data to create a suitable input. Apart from the baseline VAE, the authors trained a VAE with a *modified evidence lower bound (M-ELBO)* loss function to avoid learning the abnormal patterns of the data. Furthermore, they used a technique proposed by [66] to approximate missing data points with reconstruction values. Results reported by the authors indicate that their VAE variant (with *M-ELBO*) outperform supervised and anomaly detection based on VAE.
Typically, a sequence-to-sequence model is used for sequences where the time dimension is important. As outlined in [85], such model consists of an encoder that extracts a latent representation of the input and a decoder that decodes the target sequence. This is also similar to how an AutoEncoder for anomaly detection works, as outlined in Section 2.8.4. In machine translation, Sutskever et al. present an end-to-end sequence-to-sequence model consisting of stacked LSTMs that act as encoders and decoders. Due to the nature of RNNs (LSTM in this case), such model is able to handle variable-length

sequences. Figure 3.6 shows an example of this. They report that their architecture is able to handle both short and long sentences. They also discover that reversing the order of the input data improved the performance of the model significantly.

Srivastava et al. extended the idea of modelling machine translation sequences presented in [85] to videos. In a paper titled “Unsupervised Learning of Video Representations using LSTMs”, Srivastava et al. present an unsupervised approach of learning video representations using LSTMs. They argue that videos, when compared to single images, contain much higher dimensional data and therefore an unsupervised approach is only natural. In their experiments, they also utilise an LSTM Encoder-Decoder to learn video representations. In their experiments, they used image patches and high-level ’features’ extracted from the last hidden state of a pre-trained CNN. The authors propose a composite model that performs two main tasks: (1) reconstruct the input frame(s) (described in Section 3.4.1) and (2) predict the future frames (see Section 3.5). In their experiments they compared the AutoEncoder as a reconstruction model (more on this in the next section), the future predictor and the composite model. The authors report that the composite model performed the best and conclude that LSTMs can indeed learn good video representations even to predict future motion frames [83].

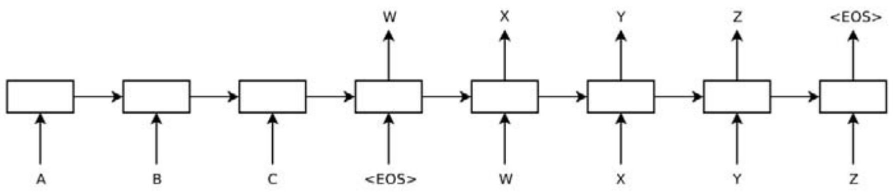


Figure 3.6: Sequence-to-sequence model. Input: *ABC*, Output: *WXYZ*. (EOS) is a token used to indicate the end of the sentence
Source: Sutskever et al. [85]

3.4.1 AutoEncoders as Reconstruction models

LSTMs have been proposed to deal with unsupervised anomaly detection. In their paper, Malhotra et al. [55] propose an LSTM based on Encoder-Decoder networks that is able to learn the data representation of normal multi-variate time-series events. Using the same network, new events (possibly anomalous) are fed to the trained network and using the reconstruction error, the authors detect abnormal events that do not ’belong’ to the normal time-series data. The approach is similar to the sequence-to-sequence model described in this chapter but in this case, the target sequence is the same as the source.

The authors use 2 LSTM modules in total - one for encoding the data into fixed length vector representation and the other to act as the decoder and reconstruct the input. To train their model, they employ the semi-supervised approach outlined in Section 2.2 by feeding in the normal time-series data only. The reason behind this is to force the encoder-decoder model to learn to reconstruct normal data but fail to reconstruct data that deviates from the normal set. This, therefore, allows the network to act as a reconstruction model in order to be able to detect anomalous data samples [55]. Such approach is very useful for anomaly detection tasks where abnormal instances are limited and expensive to collect.

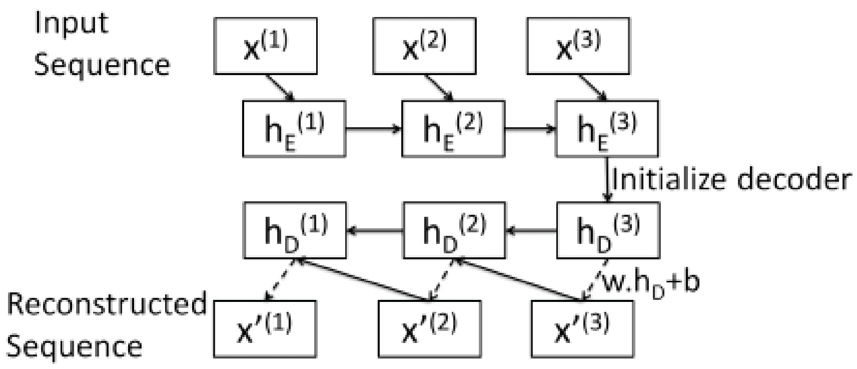


Figure 3.7: LSTM Encoder-Decoder inference for input with length L = 3
Source: Malhotra et al. [55]

Similar to the approach undertaken in [54], the authors split their normal data into various sets: one containing normal data for training the model, one containing normal data to act as the validation set for early stopping and for estimating μ and Σ (more on this in the next paragraph), another normal validation set for estimating model parameters and finally a normal test set. Figure 3.7 shows the an inference example of the LSTM encoder-decoder model proposed by [55] with input sequence of length L 3. The input $x^{(i)}$ together with ${h_E}^{(i-1)}$ are used to compute the hidden unit ${h_E}^{(i)}$ whereas ${h_E}^{(L)}$ is used to initialise the decoder ${h_D}^{(L)}$. A linear layer consisting of weight matrix w and bias vector b is then used to output the target sequence.

The reconstruction error vector for t_i is given by $e^{(i)} = |x^{(i)} - x'^{(i)}|$ [55]. Such error vectors are then utilised to approximate the parameters μ and Σ using Maximum Likelihood Estimation (MLE). Using these parameters, an anomaly score could be computed. The authors tested their model on 5 real-world datasets with various window length size ranging from 30 to 500 depending on the domain. Furthermore, they experimented with converting multivariate time-series data to univariate using PCA for some datasets. They also applied a sliding window with different step sizes and set the hyper-parameters that maximize the F-score on the validation sets. These parameters include the number of hidden units in each LSTM module (ranging from 40 to 90) and the anomaly score threshold τ. In their results they claim that their technique works on both short (≈ 30 time steps) and long (≈ 500 time steps) time-series data and show that the method can detect anomalies from both unpredictable and predictable data.

In their paper titled 'Learning Temporal Regularity in Video Sequences', Hasan et al. deal with learning normal sequences from videos by training an AutoEncoder to act as a reconstruction model. Similar to the approach in [3; 16; 30; 84; 98], the authors also employ a semi-supervised approach as they trained their models on the regular (i.e. free from anomalous sequences) scenes only. Initially, the authors trained an autoencoder with handcrafted spatio-temporal features extracted from the footage. However, they later found out that handcrafted features was not the optimal solution for detecting anomalous sequences so they trained a fully convolutional autoencoder (as it contains no fully connected layers at the end) that processes short video clips with a temporal sliding window. To detect anomalous events, they compute the reconstruction error in a similar way as explained by [55]. When the model is able to reconstruct the scene with a low error then it is considered to be regular. However, when the error is high and exceeds a threshold, the scene is considered to contain anomalous sequences. The authors tested their method on various datasets and claim that it is able to generalise quite well.

3.5 Forecasting models

Other similar work by Malhotra et al. also uses LSTMs to detect anomalies in time-series data. However, unlike the approach in [55], the authors uses stacked LSTMs to build a forecasting model of the current data. In this case, past time-series values are fed to the model and the next time-steps are predicted. The prediction error is used to assess the likelihood of anomalous events. A high prediction error indicates an anomaly whilst a low error indicates a normal data instance. Although the results of this technique were promising, such method requires predictable time-series data such as sensor data [54].

In [71] Sainath et al. present an architecture codenamed *CLDNN* which is a combination of CNNs, LSTMs and DNNs. They argued that since these different architectures were proven to have different modelling capabilities, the combination of them should be fruitful. In fact, through their experiments related to speech recognition the proposed model is reported to have a 4 to 6% reduction in Word Error Rate (WER) when compared to the strongest individual LSTM model.

On a parallel track, in their paper titled "*Convolutional LSTM network: A machine learning approach for precipitation nowcasting*", Shi et al. propose a new technique for modelling spatio-temporal dependencies. They create a model that is able to predict the rainfall in a given area. They leverage the power of CNNs and LSTMs in order to be able to extract important spatial features as well as model the temporal dependencies within the spatial features. Since a fully connected LSTM is not able to model spatial data, the authors introduce convolutions in both the input-to-state and state-to-state transitions of an LSTM. Similar to the work in Sutskever et al. [85], the authors use sequence-to-sequence learning where a stack of ConvLSTM encoders compresses the input sequence to a hidden state tensor and another stack of forecasting ConvLSTM modules decodes the hidden state to provide a predicted output. The authors tested this architecture on two datasets: a synthetic one called *Moving-MNIST* and another containing radar weather maps. In their experiments, Shi et al. compared a fully connected LSTM with 2 layers containing 2048 hidden units each as well as 5 ConvLSTM architectures with different number of layers and kernel sizes. Results reported by the authors show that the ConvLSTM model outperforms the fully-connected LSTM model as well as an optical flow based algorithm named *ROVER*. Although this was a classification task, the method can be also utilised for detecting anomalous data patterns. This could be done by measuring the error between the predicted outcome (forecast) and the ground truth. During training, a threshold could be set which allows new data points to be classified accordingly. In this case, a high prediction error would indicate an anomaly. This is similar to the work done in [54] and [55].

In time-series classification tasks, literature usually focuses on processing data in one-dimension. In a recent study researchers Hatami et al. [33] explore the use of CNN with 2D kernels on time-series classification tasks by transforming time-series data into 2D images. More specifically they use recurrence plots to represent a time-series signal in a 2D image. In their experiments they show that the their approach is competitive with other deep architectures and state-of-the-art classifiers. A similar approach is presented in Jiang and Yin [41] where the authors treat raw time-series signals from a set of accelerometer and gyroscope sensors as a signal image. Such images are then used to train a deep CNN in an effort to recognise different human activities.

3.6 Sensor-based techniques

Yin et al. [98] use wearable wireless sensors attached to a human in order to detect abnormal events. The authors argue that other approaches to the problem are affected by the issue of high false positives. To overcome this problem they train a one-class support vector machine (SVM) on the normal data only which in itself is able to act as an outlier detector. They perform a kernel non-linear regression analysis in order to reduce the false positive rates. Yin et al. argue that it is very difficult to obtain a dataset with abnormal activities however relatively easy to do for normal events and thus this allows a model to be trained on the normal events which in turn will be able to detect abnormal activities. They defined abnormal instances as activities that (1) occur rarely and (2) were not expected in advance (not seen). In order to evaluate their work, they generated a dataset containing both normal and abnormal instances. For the abnormal instances, the dataset consisted of simulated actions such as *"failing down"*, *"slipping on a wet floor"* and others. They used both the *detection rate* (TN/(TN+FN)) and *false alarm rate* (FP/(FP+TP)) as evaluation metrics. The AUC (area under the ROC curves) were computed to compare the performance between three different algorithms; OneSVM, SVM+MLLR and SVM+KNLR. Reported results indicate that the SVM+KNLR outperformed the other algorithms.

Other researchers also make use of various sensors to detect abnormal events [77; 87; 88]. The use of sensors on the subjects has its benefits such as continuous monitoring in an un-intrusive way as well as disadvantages which includes rigorous set-up and configuration, limited power and communication issues. In contrast, the method proposed for this study extracts human joint key points from a small and cheap camera without the need of recording the actual footage. The set-up in itself is non-invasive and respects the individual's privacy as only the skeleton data is used for processing.

3.7 Camera-based techniques

Several work also deal with fall detection in supportive home nursing or elderly people. Zhong et al. [101] present an efficient and stable unsupervised technique model that is trained on features such as moving objects trajectory, speed and the shape descriptor. They choose to model the features using one of the graphical models - a Hidden Markov

Model (HMM). The same approach as [3; 16; 30; 84; 98] is adopted here, the authors detect unusual events by matching all event segments with the normal model. Anything not fitting the normal model is considered to be abnormal. Similar work was also done by Hung et al. [38] with the main difference being using RFID-based sensors and a combination of HMM and SVMs to detect abnormal events of an elderly person.

Using multiple cameras, Antonakaki et al. [3] proposed a method of detecting normal and abnormal behaviours by dividing the classification problem into two separate problems. The first is detecting the human action (e.g. Walking, running, sitting down, etc.) and the second involves focusing on the trajectory. Using two one-class classifiers (SVM and cHMM) they report good results in the task of behaviour recognition [3]. Other related studies also seem to focus on monitoring and detecting falls. Some approaches include the use of image depth sensor to detect falls without the use of wearable technology [23], while Rougier et al. aim to detect falls by tracking the head of a person and classify the fall by velocity characteristics [68].

Debard et al. [19] propose a fall detection algorithm by using data from a camera system. In their approach, the authors used background subtraction together with a particle filter to locate and track the person from the video footage. After detecting the human subject from the footage, they constructed a feature vector consisting of 5 elements: aspect ratio, change in aspect ratio, fall angle, head speed and center speed. Such features were then used to train an SVM to classify events as falls and non-falls. To evaluate their proposed algorithm, they used a dataset [5] which consists of various human subjects individually performing different activities in a room. The dataset contains realistic fall activities as well as other activities that might cause false alarms. The authors of this dataset created such abnormal activities by re-enacting real-life incidents that were discovered in other studies. Since the dataset contains footage from 5 different cameras, Debard et al. trained 5 models separately. To evaluate their models, they calculated the precision-recall curve for every model (camera) and calculated the area under the curve (AUC). The average precision for all cameras was 41% whereas the average recall was 62%. They concluded that the particle filter used to enhance foreground segmentation yielded the best results. However, their algorithm still generated a large amount of false alarms and 24% of the falls were not detected. They claim that the main reasons were caused by room light changes and by the person leaving or entering the camera field of view.

More recent work in activity recognition make use of depth sensors to collect depth information. In particular, Yang et al. [96] proposed a spatio-temporal context tracking over 3D depth images. The authors used the Kinect sensor that is able to collect various data such as depth information and skeleton data. The method they employed calculates the distance from the floor level to the person's head position as well as the centroid height and finally use an adaptive threshold to classify the event.

A recent approach to detecting anomalous human behaviour events such as falls was proposed by Yao et al. [97] in their paper titled 'A New Approach to Fall Detection Based on the Human Torso Motion Model'. Their approach is similar to [96] as they are extracting specific features from a 3D representation of a depth image. Using the joint positions of the hip and shoulder centres, the person's torso angle is calculated. When such angle exceeds a set threshold, the changing rates together with centroid height are computed for a given sequence of time. These rates are then used to classify a given sequence as normal or abnormal. Testing on their own dataset, the authors claim that their approach achieved a detection accuracy rate of 97.5%.

However, some of the literature reviewed above have shortcomings as they only focus on detecting falls. Whilst it is reported that falls are one of the major health hazards especially in elderly persons, there are other types of behaviour that could show that a person is in some sort of difficulty or distress [60]. For example, in [96] and [97], specific features related to just falls are being used to train a model. Other behaviour that a person does not usually follow is not detected using these methods. Some examples include odd trajectory motion such as waving hands or feet, limping and others.

3.8 Conclusion

In this chapter, an overview of the sequence-based techniques used in anomaly detection was given. In recent works, researchers discovered that deep convolutional neural networks, autoencoders and other variants achieved remarkable performance on learning difficult tasks. In particular, deep LSTMs have shown excellent performance in learning predictable and unpredictable sequences [55]. Furthermore, results reported by researchers in the field show that the use of deep neural networks and their variants outperforms the traditional techniques (such as SVM and HMM) [43]. Recent studies also show that CNNs and similar architectures have established themselves as very powerful techniques in learning useful representations from raw data without the need for hand-crafted features [33; 44; 76; 86].

Various studies on skeleton-based human representations were also outlined in this chapter. Researchers have extracted 2D and 3D space-time features from images to generate human representations and be able to classify sequences into actions. Others used sensors and special thermal cameras to detect falls and/or irregular behaviour. The approach investigated in this study uses human representations extracted from videos in order to detect abnormal behaviour such as falls. Specifically, AutoEncoders will be used to learn the normal activities of the individual and be able to detect abnormal sequences. This is described in detail in the next chapter.

4

Methodology

4.1 Introduction

This chapter provides a detailed description of the methodology and the implementation adopted to develop a technique that aims at detecting abnormal behaviour, such as falls, from video footage. An overview of the modules of the system and how these will be combined together are also outlined in this chapter.

This chapter is made up of two main parts. The first part aims at presenting the proposed method of detecting abnormal behaviour from video data. In the second part, the method used to compare different human action representations, consisting of pose estimated and sensor data, is given. Both parts present the model architecture designs used in the experiments.

4.2 Proposed Method

The greater part of the work in anomaly detection focuses on general scene understanding such as pedestrian tracking, crowd control, unrestricted areas and irregular object detection [86; 100; 101]. Other work in abnormal human behaviour uses multiple cameras [3], depth sensors [96] or extract specific features that could contribute to abnormal behaviour [97].

This study focuses on individual-human based activities and the detection of irregular body posture is given. In contrast to the work outlined in Chapter 3, a semi-supervised approach with little to no supervision is proposed. This method extracts human joint keypoints from a regular camera without the need for sensors or for recording the actual footage. Furthermore, the models are trained solely on the extracted keypoints, without any further feature engineering. In this dissertation, body keypoints refer to the eyes, ears, nose, shoulders, elbows, wrists, hips, knees and ankles of an individual.

The set-up in itself is clean, non-invasive and respects the individual's privacy because only the body keypoints are required to train a model. Although depth cameras provide more information [31], this study will focus on 2D data which can be obtained from any regular camera and existing setups, ranging from a cheap USB-powered webcam (or a Raspberry Pi) to a digital single-lens reflex (DSLR) camera. In this study, irregular or abnormal human behaviour refers to abnormality in the body posture. This means that, from a sequence, consideration is given only to human representation, specifically the body keypoints.

As outlined in Chapter 1, the main aim is to determine whether a set of human body keypoints extracted from RGB 2D images can be used to detect abnormal human behaviour, such as falls. This study also seeks to determine whether human body keypoints estimated from 2D images are comparable to data generated by sensors such as depth cameras and wearable inertial sensors (see Section 4.6).

The general system overview of the proposed method is depicted in Figure 4.1. The figure depicts the data flow diagram (DFD) of the proposed method. In this section, each and every process depicted in the DFD is explained on its own merits.

The first step deals with processing the video footage. This involves extracting all the frames from every video. Afterwards, the body keypoints such as eyes, nose, elbows and hips are extracted from the footage using two pre-trained pose estimation models [9; 61]. As outlined in Chapter 3, Section 3.3, these machine learning models were trained to detect humans in an image and estimate the 2D location (x and y) of key body joints. The output of such models is then processed and used to train a model that is able to learn a generalised representation of the normal daily activities. Such model is then used as a reconstruction model to classify sequences as normal or abnormal.

Inspired by the work done by Malhotra et al. [55] and Hasan et al. [32], a semi-supervised approach is employed throughout this study. By using the said approach, the model is only exposed to the event sequences containing 'normal' behaviour. The data used to train and test the model is made up of time-series data containing the x and y values of every body keypoint. After training the model, test sequences consist-

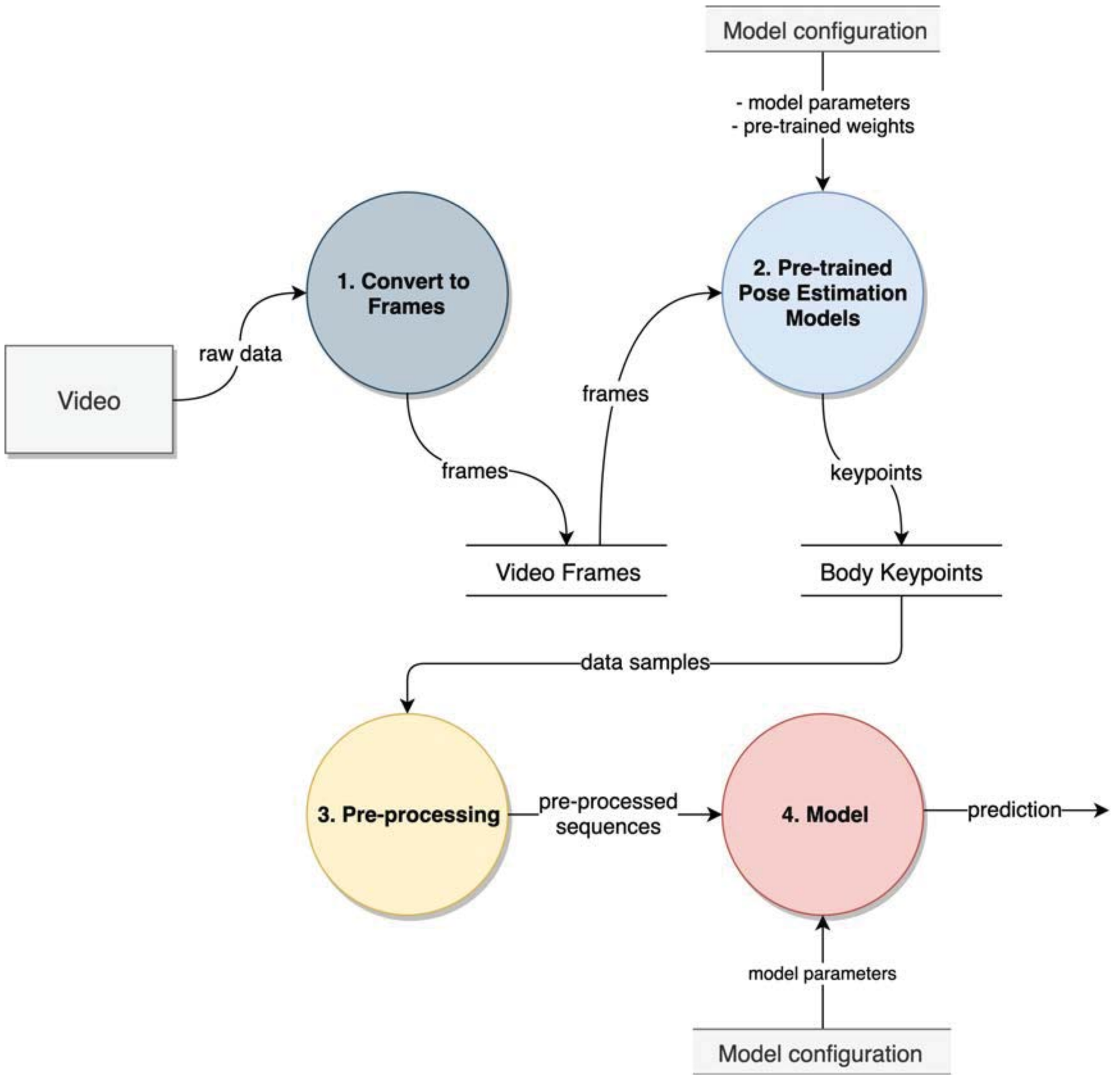


Figure 4.1: Data Flow Diagram: Level 1 - General System Overview

ing of both 'normal' and 'abnormal' are processed and fed to the model. Using a defined threshold for the reconstruction error (based on the output of the model), a sequence is classified as normal or abnormal. The main reasoning behind the model is similar to how human beings detect abnormal activities. An activity that rarely happens or is not common is usually considered as abnormal. Hence, the model is trained to learn and reconstruct the daily normal activities only so as to detect activities that deviate from normality (what it was trained on).

The rest of this chapter aims at providing the reader with a detailed description of each module of the system depicted in Figure 4.1. In the next section, an overview of how the videos were converted to frames (Module 1) is given. In Section 4.4, a detailed description on how the body keypoints are extracted from video datasets (Module 2) is presented. Finally, Section 4.5, provides a detailed description on how the extracted body keypoints were used to train different model architectures in order to detect anomalous body posture (Module 4). This also includes the pre-processing steps conducted on the data (Module 3) together with a justification of the model architectures (Section 4.5.2).

4.3 Converting videos to frames (Module 1)

The first step involves extracting all the frames from every video. This allows each video frame to be processed separately. FFmpeg[1], which is a free-ware tool used to record, convert and stream both audio and video was utilised. A Python script was written to load all the folders and process all the videos. To retain image quality, the *-qscale:v* (quality scale for video) option was used. A value of 2 (with 1 being the highest) was used for extraction as it was observed that the default setting produced noisy output.

4.4 Extracting features: Pose Estimation (Module 2)

Generating hand-crafted features typically requires domain expertise. Recent studies in the field show that deep CNNs managed to extract useful data representations from raw video data [32; 43]. As outlined in Section 3.2.1, skeleton-based human representations have gained a lot of attention recently. For example in [53] and [81], models were trained on 3D skeleton data to recognise human actions. In both studies, the authors relied on data generated by depth camera sensors such as the Kinect. In contrast, in this study, the human skeleton will be extracted from an ordinary RGB video with pre-trained pose estimation models. As outlined in 3.3, such models are trained to detect humans from a frame and estimate the location of the body joint key-points. This approach allows the extraction of human representations from existing footage without the need for further equipment, body-worn sensors or installations. Furthermore, it allows the model to be trained solely on the actions performed by the human subject. This helps to discard any background noise in the video which could lead to the model learning features that do not contribute to the action itself as seen in 3.1.1.

[1]FFmpeg - `https://ffmpeg.org/` - a tool used to record, convert and stream audio and video

Some of the techniques described in the previous chapter require a supervised approach based on labelled data [96; 97]. As outlined in Section 2.2, annotated data is very scarce because the annotation process itself is time-consuming and expensive. In view of this, a semi-supervised approach is employed, whereby the model is trained on the normal sequences only, similar to [16; 31; 55]. This method, unlike the one in [98], extracts human joint key points from a camera without the need for sensors or for recording the actual footage.

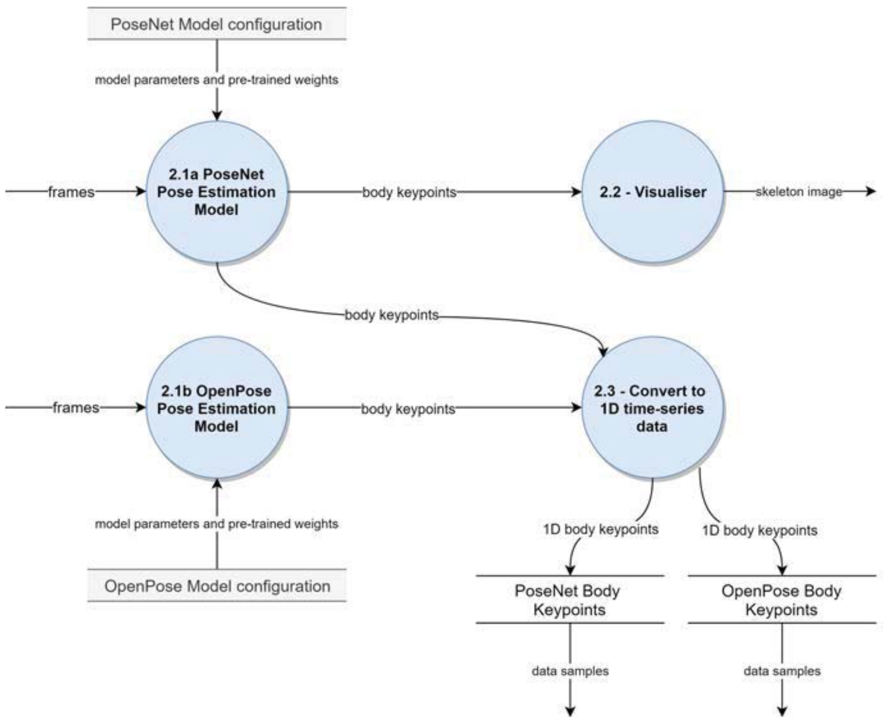

PoseNet Model configuration
model parameters and pre-trained weights
frames
2.1a PoseNet Pose Estimation Model
body keypoints
2.2 - Visualiser
skeleton image
body keypoints
frames
2.1b OpenPose Pose Estimation Model
body keypoints
2.3 - Convert to 1D time-series data
model parameters and pre-trained weights
OpenPose Model configuration
1D body keypoints
1D body keypoints
PoseNet Body Keypoints
OpenPose Body Keypoints
data samples
data samples

Figure 4.2: Data Flow Diagram: Level 2 - Module 2 (Pose Estimation)

In this study, two pre-trained pose estimators namely PoseNet [61] and OpenPose-CMU [9] are used and compared. Outlined in 3.3, these models are both available for public use and are ranked well in dataset leader-boards such as COCO[2]. To date, the PoseNet model is only available as a TensorFlow.JS[3] model and as such, the estimation process for extracting body keypoints has to be done in the browser using JavaScript.

Figure 4.2 depicts the sub-modules of Module 2 shown in Figure 4.1. Frames are processed by both algorithms (2.1a and 2.1b) and their output is then converted to 1D time-series data where the order of the video sequence is preserved. The next two sections describe in detail how both algorithms were utilised to extract body keypoints.

[2]`http://cocodataset.org/#keypoints-leaderboard`
[3]`https://js.tensorflow.org/`

4.4.1 PoseNet

As stated, the pre-trained PoseNet model is only available as a TensorFlow.JS model to date. In view of this, a JavaScript program responsible for loading and processing all the images was developed. After converting videos to frames (Section 4.3), folder batches of 800 images were created. For instance, if a video contained 1600 frames in total, two folders (named '1' and '2') are created whereby the first 800 frames are moved to folder '1' and the rest are moved to folder '2'. This is was done due to a browser memory limitation (in this case Google Chrome) that limits the amount of canvas items that could be loaded into memory.

Initially, all folders are loaded by an asynchronous call that retrieves the list of folders ready to be processed. Then, for every folder, a *Canvas* element for every image is created and the image is loaded. The model is then initialised and the '*estimateSinglePose()*' method is called. This method returns a total of 17 body keypoints consisting of x and y values depicted in Figure 4.3. For every keypoint, a confidence score between 0 and 1 is generated. All the frame keypoints are then saved in the original sequence as a JavaScript Object Notation (JSON) object. A sample of the JSON object for 3 keypoints (out of 17) is shown in the appendix, in Listing A.1.

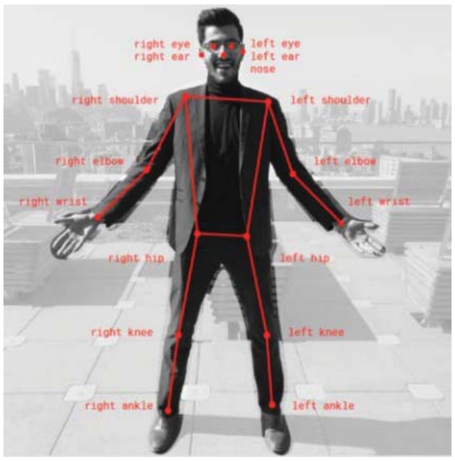


Figure 4.3: PoseNet body keypoints. (Source: [61])

In order to increase the accuracy of the model's output, the parameters used to estimate the body keypoints were tuned as suggested by the authors in [61] and the official GitHub repository for *tfjs-models*[4]. Furthermore, the *single-person* pose estimation,

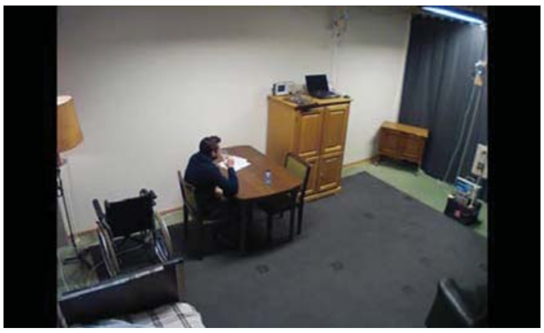
[4]`https://github.com/tensorflow/tfjs-models/tree/master/posenet`

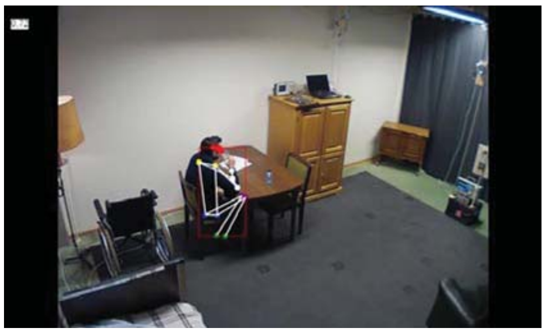
which is simpler and faster than the *multi-person*, was used because the footage used in this study was recorded containing one person only (except for one scenario in the ADL dataset that was excluded). The key parameters in question include *imageScaleFactor*, *outputStride* and *multiplier*. The *imageScaleFactor* scales the image down before it is fed to the model. A value of 1 would not scale the image down. Due to the resolution of the images used in this study, this value was set to 0.7 as a higher value would result in memory overflow error (this is a browser limitation and the amount of memory available does not matter). The *outputStride* parameter was set to 8, which is the lowest possible value for highest accuracy, at the expense of performance. This parameter determines the scale factor of the output relative to the input image size. In turn, it determines the size of the layers and the model's outputs. Finally, the *multiplier* parameter is passed during initialisation of the model. This specifies the *MobileNet* architecture to be used. Specifically, it determines the float multiplier for the number of channels for all convolutions. In this case, the parameter was set to the highest value (1.01) - which allows for higher accuracy at the cost of speed.

When the whole folder is processed, every set of keypoints for every frame are concatenated and passed to a PHP script. The script is responsible for saving all the extracted keypoints in a JSON file (Module 2.3). A sample of the output file is shown in the appendix, Listing A.2. Every frame in this file is represented by a total of 68 elements ($17 * 4$). Each keypoint is represented by 4 elements in the following order (as shown in the listing): (1) the keypoint confidence score, (2) the x position, (3) the y position and, (4) the frame file name. As an example, if 100 frames were processed, the total number of elements in the final JSON file amounts to 6,800 ($100 * 17 * 4$).

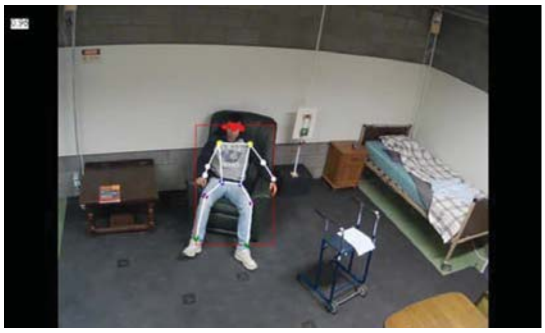
In order to visualise the estimated body keypoints (Module 2.2), the canvas drawing tools within JavaScript were used to: (1) draw a bounding box marking the detected person, (2) plot the keypoints using the circle marker in different colours (where each colour represents different body parts) and finally (3) draw the skeleton of the individual which joins adjacent key points together. Figure 4.4 shows a sample of 3 images from different camera positions within the ADL dataset (outlined in Section 5.3.5) before and after processing. It is interesting to note that the model is also capable of estimating keypoints that are occluded by other objects. For example, in Figure 4.4(f), both the knees and ankle points were correctly estimated even though they were occluded by the table.

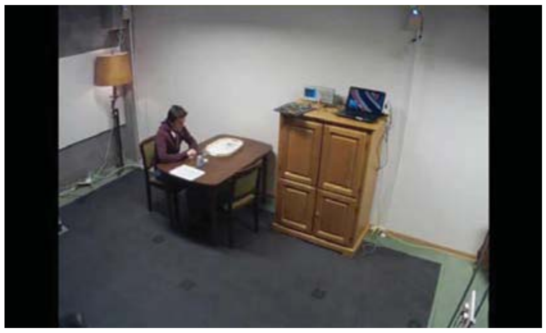
After manually observing the output of the algorithm, it was interesting to observe that in some cases where a person is lying on the floor, PoseNet fails to recognise some


(a) Cam1 Before


(b) Cam1 After

(c) Cam2 Before


(d) Cam2 After


(e) Cam3 Before

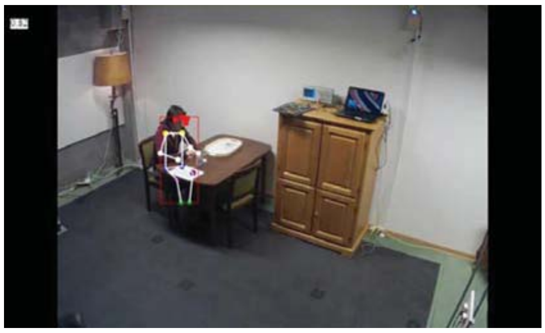

(f) Cam3 After

Figure 4.4: Body keypoint visualiser (Module 2.2). Before shots: screen shots from ADL dataset, After shots: developed PoseNet visualiser

of the keypoints. In view of this, it was decided to rotate the input image to various 90° rotations to determine whether an improved score could be predicted. The idea behind the rotation was to feed the model an upright position version of the human subject. Figure 4.5 illustrates the data flow of this sub-module. In total, four versions of the input image were fed to the model. These include: the original image (no rotation), 90° rotation, 180° rotation and 270° rotation. Figure 4.6 depicts the output for 2 versions of

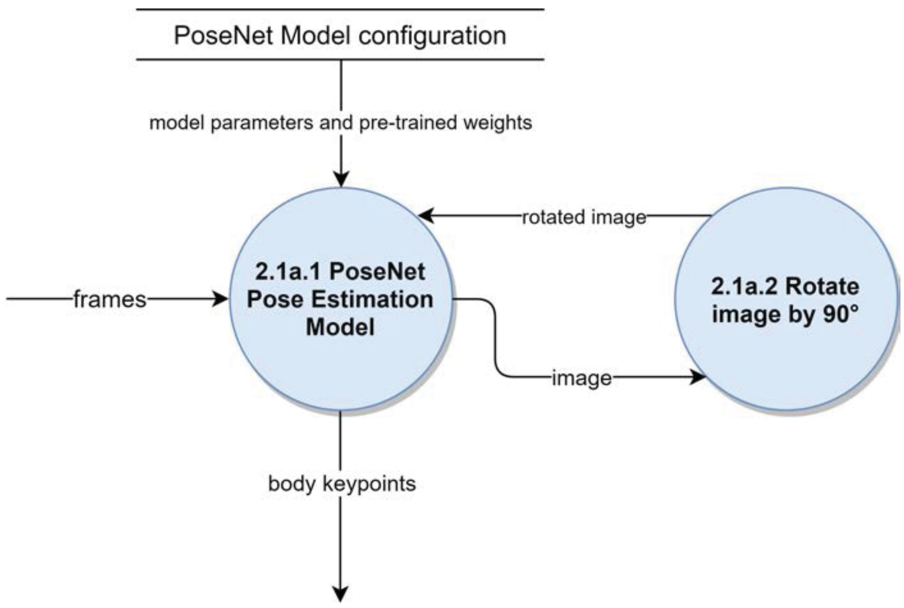
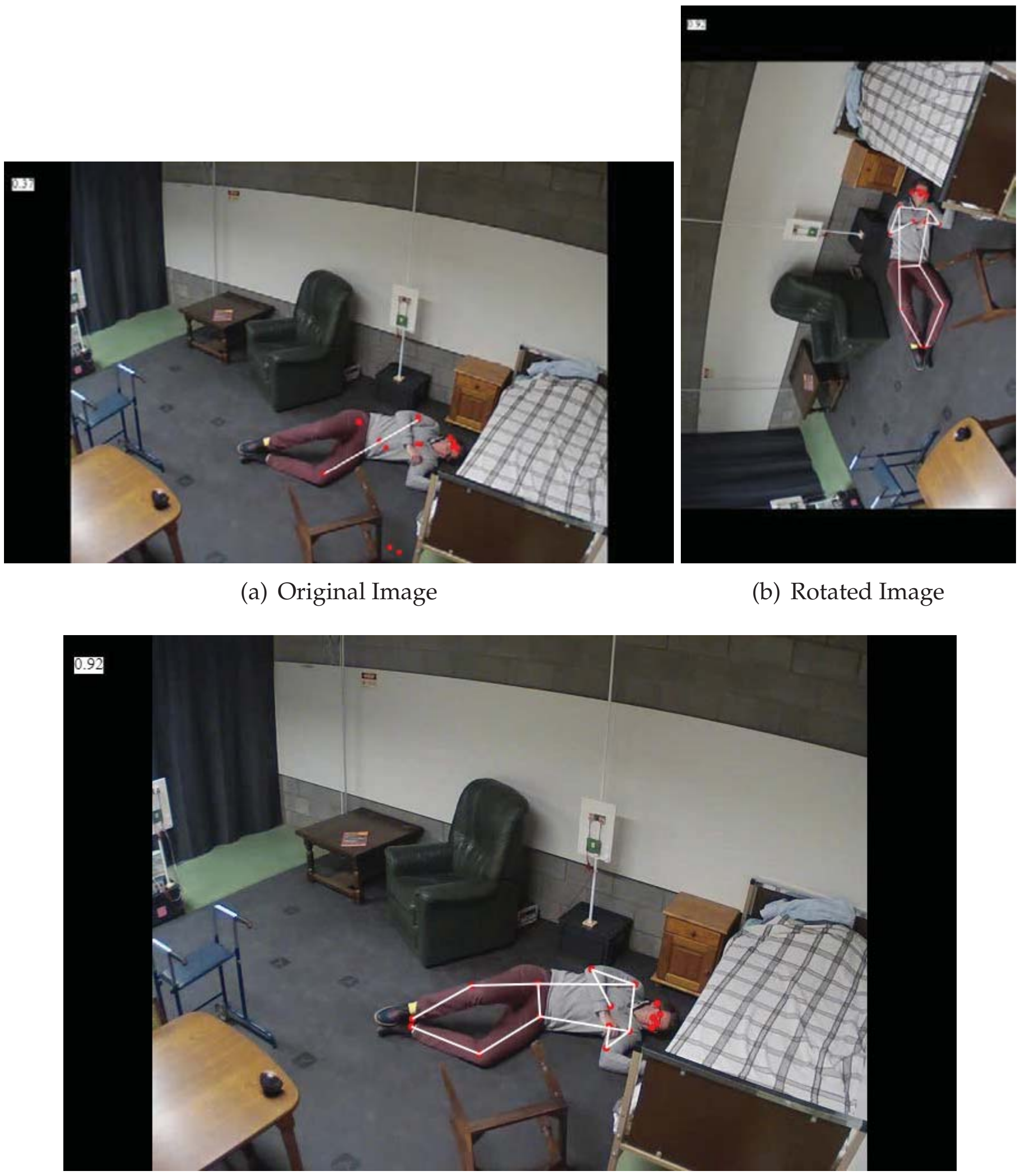

PoseNet Model configuration
model parameters and pre-trained weights
rotated image
frames
2.1a.1 PoseNet Pose Estimation Model
2.1a.2 Rotate image by 90°
image
body keypoints

Figure 4.5: Data Flow Diagram: Level 3 - Module 2.1a. Image rotations

the input image - the original and the image rotated to 270°. In the original image, the algorithm scored 37%, whilst for the rotated image it managed a score of 92% - an outstanding increase of 55% in confidence score. The highest scoring keypoints were then mapped to match the original image and saved in the final JSON file. This was done to preserve the original pose of the human subject. Such process is depicted in Figure 4.6(c). This experiment was conducted on 23,989 frames which contained various types of falls. It was observed that 67% of the frames obtained a better score when rotated, namely 47% when rotated 90 degrees, 43% when rotated 270 degrees and 10% when rotated 180 degrees. The score difference between the original and the rotated image ranged between 10% and 70%. Such difference can be visualised in Figure 4.7, which plots the scores for all image versions for the first 300 frames.

The pose estimation process was computed on Windows 10 using Google Chrome. The machine was equipped with an Intel Core i7 (6 cores) 8700k clocked at 3.70Ghz, 16GB RAM and an NVIDIA GeForce GTX 1080 as the graphical processing unit (GPU). Overall, the GPU managed to process $\approx$ 60 frames per second if ran in simultaneous mode.



(a) Original Image

(b) Rotated Image

(c) Best keypoints mapped to the original image

Figure 4.6: PoseNet output for Original (a) and Rotated image (b) and how the best keypoints were mapped to the original image (c)

4.4.2 OpenPose

Outlined in Section 3.3, OpenPose is another 2D pose estimation pre-trained model that employs a bottom-up approach. The authors proposed a novel feature representation called Part Affinity Fields that encodes information such as location and orientation

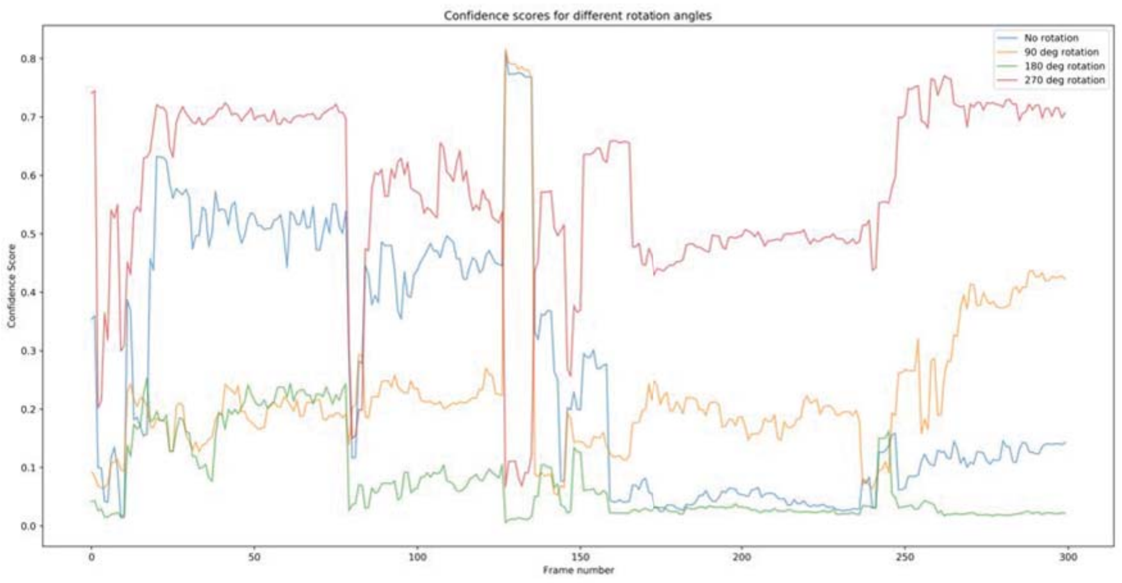


Figure 4.7: Confidence scores for different rotation angles. Higher scores are better. The blue plot represent the confidence scores for the original image whilst the other plots represent various rotations.

about the body limbs [9]. Figure 4.8 (source: OpenPose Github repo[5]) shows the output of the 25 body keypoints estimated by the model.

In this study, the OpenPose CLI tool was used. The tool accepts both video and image input and also comes with options such as *write_json*, *display* and *render_pose*. The *write_json* flag, when enabled, writes a JSON file with all the estimated pose for every frame whereas the *display* and *render_pose* flags are both used for visualisations purposes. All the footage for the video datasets were processed by the CLI tool. Due to the large amount of video files, a Python script was written to load all the videos and issue the command. A sample of the JSON output for one frame is shown in the appendix, listing A.3. After all footage was processed, another Python script was created to merge all JSON files into one for every video. Unlike PoseNet, OpenPose estimates 25 keypoints in total (as opposed to 17). The model, codenamed *BODY_25*, was used in this study as the authors claim that it is faster and more accurate than *COCO*[6]. Furthermore, the parameters suggested by the authors for highest possible accuracy were used.

[5] `https://github.com/CMU-Perceptual-Computing-Lab/openpose`

[6] `https://github.com/CMU-Perceptual-Computing-Lab/openpose/blob/master/doc/faq.md#difference-between-body_25-vs.-coco-vs.-mpi`

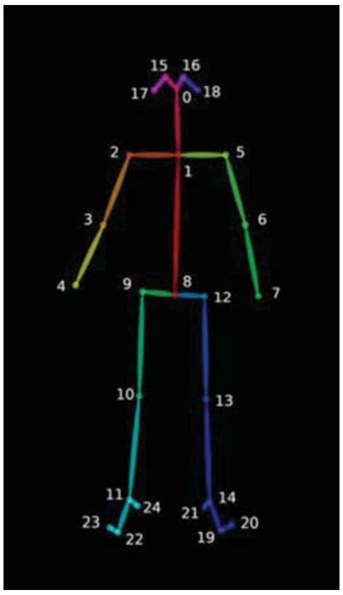


Figure 4.8: OpenPose body keypoints. (Source: OpenPose Github page)

4.5 Detecting abnormal behaviour

This section aims at presenting the technique employed in this study to detect abnormal human behaviour through body posture. The first two modules (marked 1 and 2) shown in Figure 4.1 were outlined in Sections 4.3 and 4.4 respectively. In this part, modules marked 3 and 4 are described in greater detail. Furthermore, the theories presented in the previous chapters are coupled with the techniques proposed.

4.5.1 Pre-processing techniques (Module 3)

A video dataset is different from traditional numerical datasets and typically complex to process. This is due to the fact that video frames are very dynamic and thus, such changes need to be detected and processed. Figure 4.9 illustrates a data flow diagram of how the data was pre-processed before fed to the models.

4.5.1.1 Computing confidence score statistics

The first sub-module (3.1) deals with computing the confidence score statistics for each set of frames. Initially, the files containing the sequential body keypoints are loaded in a Python script. Then, the mean score for a set of frames is computed in order to determine whether the frames should be omitted or not.

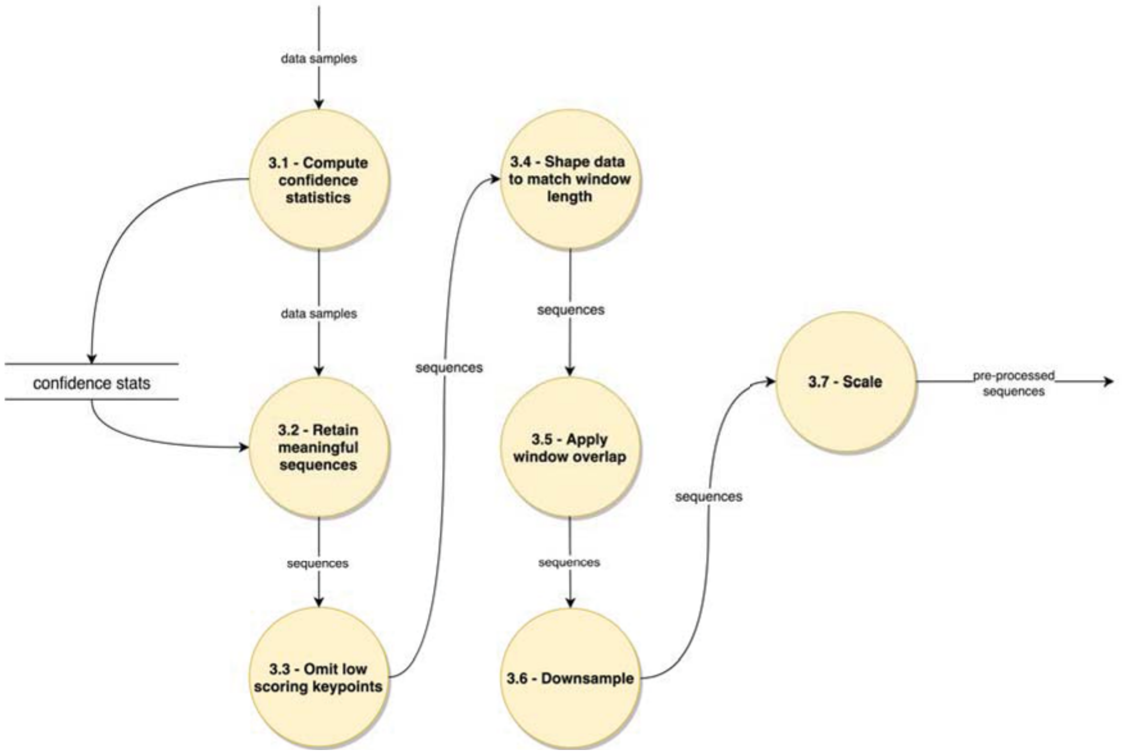

data samples
3.1 - Compute confidence statistics
data samples
confidence stats
3.2 - Retain meaningful sequences
sequences
3.3 - Omit low scoring keypoints
sequences
3.4 - Shape data to match window length
sequences
3.5 - Apply window overlap
sequences
3.6 - Downsample
sequences
3.7 - Scale
pre-processed sequences

Figure 4.9: Data Flow Diagram - Level 2 - Pre-Processing

4.5.1.2 Retaining meaningful sequences

The second sub-module (3.2) aims at retaining only the meaningful sequences and omit the ones not informative enough. Both pose estimation algorithms (PoseNet and OpenPose) return a very low confidence score between 0 and 0.09 when no human subject is detected. These frames are naturally omitted as they carry no information. However, there are cases like the ones depicted in Figure 5.12 where a human subject is only partially invisible for a few frames (1 second or so) and thus, in order to preserve the sequence, these cases were processed as shown in Algorithm 1.

Algorithm 1: Extracting meaningful sequences

```
  Input: frame body keypoints, min_conf_score, win_length
  Output: data sequences framed in win_length
1 compute confidence score mean for $L$ frames
2 foreach $L$ mean do
3     keep $L$ frames if mean conf_score > min_conf_score
4     preserve index of $L$ frames
```

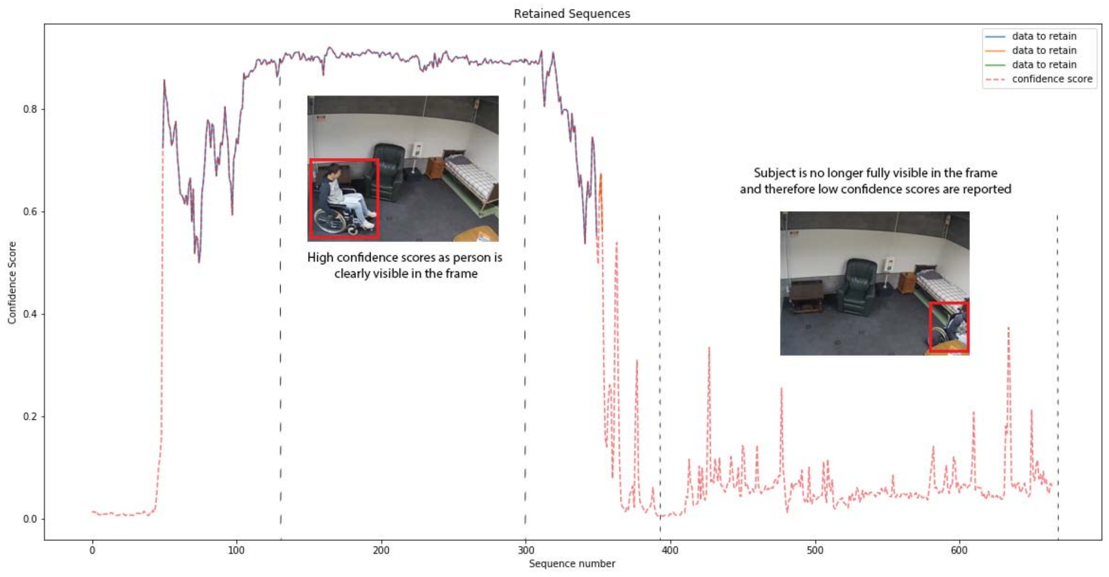


Figure 4.10: Graph showing confidence scores for every sequence at $L = 30$ and the retained sequences. Sample images represent one frame out of 30. Dotted black line indicate the range of the high and low confidence scores.

The algorithm ensures that sequences as a whole are preserved in the way they were captured. Additionally, this prevents sequences from having frames with mixed time-steps; i.e. the case of having a sequence of 30 time-steps made up of 10 time-steps from the current scene and the other 20 time-steps from a later scene is prevented. Figure 4.10 provides a visual representation of what was explained above. The graph shows the mean confidence score for every sequence at $L = 30$ (1 second) in dotted line and the extracted sequences in fixed colour lines (with min_conf_score set to 0.5). In this specific video footage, the first few frames report 0% confidence scores; this is due to the fact that no person was detected in the scene. A spike in the confidence score is seen at $\approx$ sequence number 70 - this represents the moment when the person entered the scene (who happened to be on a wheelchair). At sequence numbers > 350, the person moves to the lower right of the camera view-point having only the shoulders and hands visible. This explains the low confidence scores from the pose estimation algorithm. In this specific case, the retained sequences consist of the person entering the scene and moving around. The scene in which the person is outside the view-point was not retained. Such scene is however included in other camera views and therefore still processed. Different confidence level cutoffs (min_conf_score) that best represent the data were explored.

This is outlined in detail in Section 5.3.5.2.

4.5.1.3 Removing low scoring body keypoints

The next sub-module (3.3) aims at removing body keypoint locations that were poorly scored. Body keypoints not visible in the scene due to occlusions or simply because they are not visible in the camera's view-point are estimated with a low confidence score. To cater for this issue, points with low scores using the same minimum confidence cutoff are set to 0. This approach is similar to the work done in [94] where missing values are replaced with zeros. This technique only omits low scoring body keypoints, i.e. the rest of the points are not altered.

4.5.1.4 Window Length and Overlapping

Sub-module 3.4 is responsible for framing the data sequences into windows. This is done because in this problem, the temporal aspect is vital and thus the model is exposed to multiple time-steps of data. According to Baños et al. [6], the window length which best models a human activity is when windows are framed at 2 seconds or less. With specific regard to the ADL dataset (outlined in Section 5.3.5), 30 frames represent 1 second and therefore, setting L to 30 frames is equivalent to modelling 1 second of video footage. The authors also argue "that large window sizes do not necessarily translate into a better recognition performance". In view of this, the data was pre-processed with 3 different window lengths (1s - 30 frames, 2s - 60 frames and 3s - 90 frames). Such data configurations were selected in order to empirically pick the best window length for this problem. Similar to the approach undertaken in [19], in window lengths larger than 1 second, information from before the start time of abnormal event was included. This includes capturing 1 or 2 seconds earlier than the start time of the event.

On a parallel track, when the data is framed to sequences, the transition of an activity to another may be missed. Researchers in the field argue that one way to tackle this problem is by overlapping the window (sub-module 3.5) with the end of the previous window [46].

In line with this, a 'downsampling' technique used by Malhotra et al. [55] in their experiments is implemented. This technique (Sub-module 3.6) aims at downsampling window lengths of sizes 60 and 90 frames to 30 frames. The idea behind this is to reduce the amount of data processed while still retaining an approximation of the sequence.

4.5.1.5 Scaling the data

The final sub-module (3.7) of the pre-processing module is responsible for scaling the data between 0 and 1. According to [26], this process speeds up learning and convergence as the model is not forced to learn large weight values. Such practice also prevents the training process from getting stuck in local optima.

4.5.1.6 Inspecting the pre-processed data

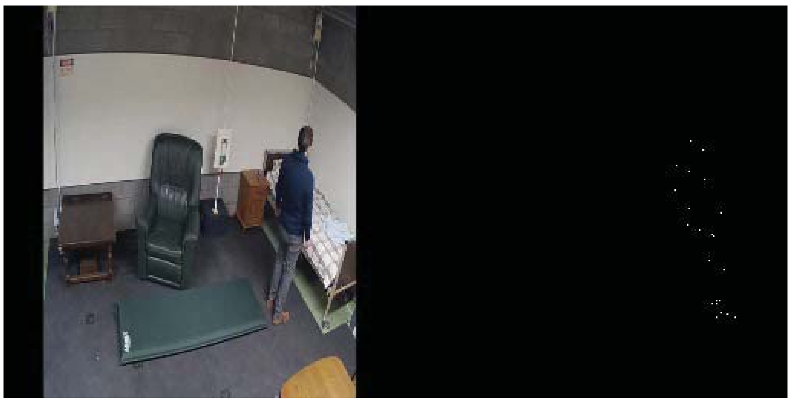
In order to inspect the pre-processed data and verify that the correct sequences were generated, a frame-keypoint visualiser was implemented. The visualiser was set to convert the keypoints vector to an image by transforming the data back to 2D, and rescale the data between 0 and 255, where a value of 255 denotes a white pixel. The original frames representing the keypoint mapping were also included in the visualiser as this allows one to manually verify and compare the output. Figure 4.11 illustrates a sample output of the visualiser. The image on the left represents the original frame whereas the image on the right shows the pre-processed pose data that is fed to the model.

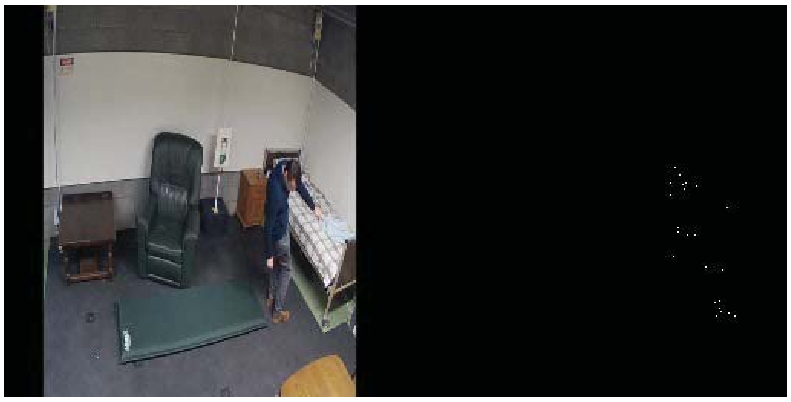
4.5.1.7 Data Parameters

In order to find the ideal combination of the dataset parameters, the models outlined in Section 4.5.2 are trained on different combinations of training sets. In this case, the dataset parameters refer to the window length, whether to overlap the windows or not, whether to remove low scoring points or not and whether to implement the downsample technique or not. Further details on how such data configurations were set are explored in the next chapter in Section 5.3.3.

4.5.2 Model architectures (Module 4)

For the task of detecting abnormal behaviour using solely video footage, two types of AutoEncoder models have been selected. Recent work in anomaly detection have shown that AutoEncoders achieved remarkable performance on learning to detect anomalous sequences in semi-supervised or unsupervised manners [32; 43; 55; 100]. The idea of training a model in a these manners is important because in nature, abnormal instances occur very rarely and therefore the acquirement of such data is expensive. Research shows that LSTM AutoEncoders and similar models, led to promising results in detecting anomalous events in sensor data [55] and video footage [32]. In this study, the idea is to extend the method of detecting anomalous sequences on human pose body keypoints instead.


(a) Fall frame number 1


(b) Fall frame number 10

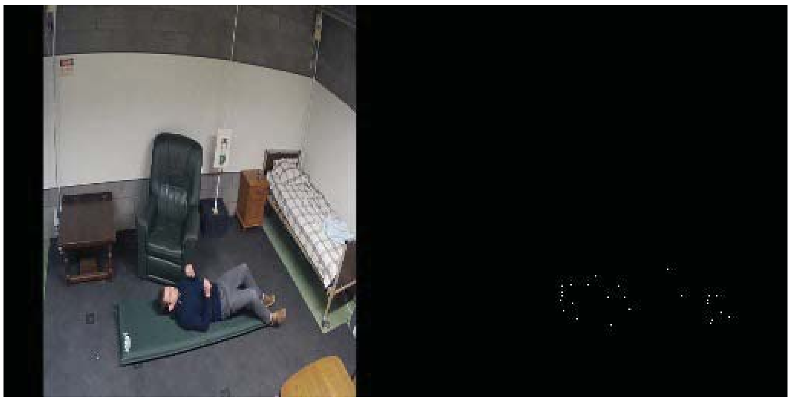

(c) Fall frame number 20

Figure 4.11: Frame-Keypoint Visualiser for one of the abnormal scenarios. Left: original frame, Right: pre-processed pose data (body keypoints)

The diagram in Figure 4.12 shows the sub-modules for Module 4 shown in the first

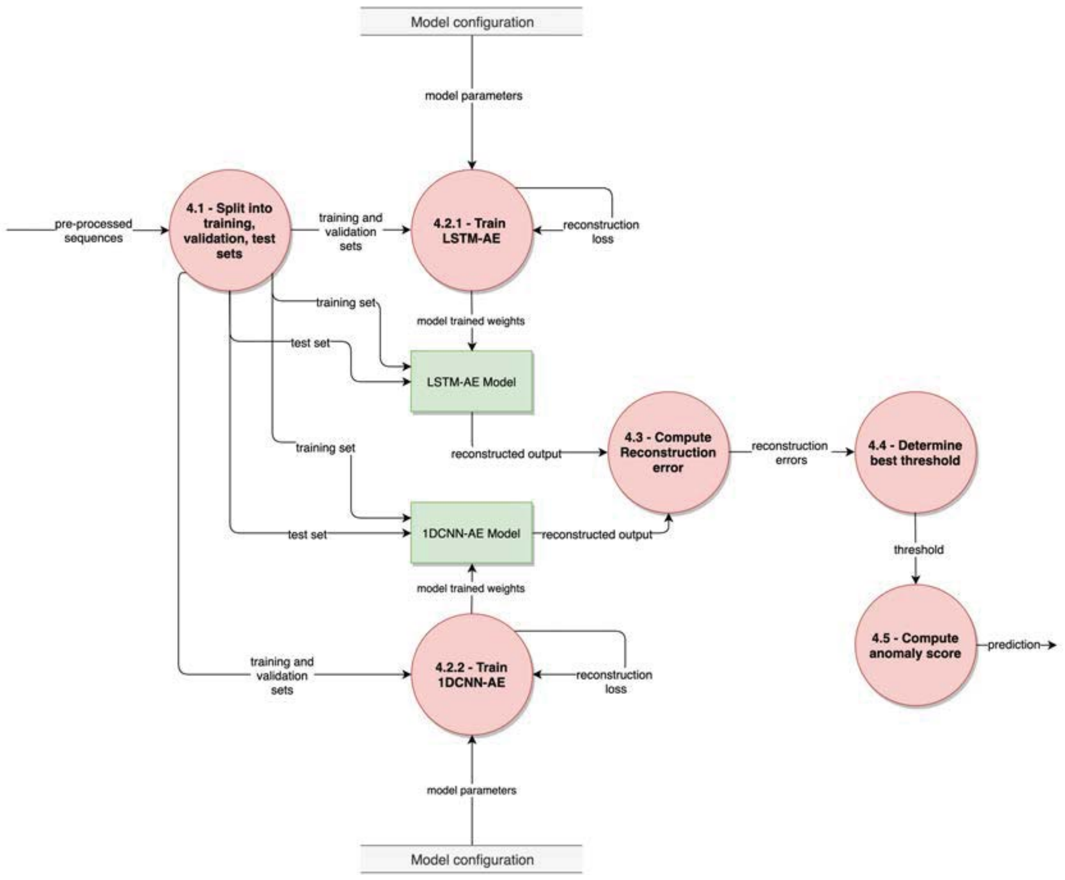


Figure 4.12: Data Flow Diagram - Level 2 - Models

DFD presented in Section 4.2. This section aims at providing a detailed description of the two types of AutoEncoder models (shown in green) designed to answer the questions of this research. Since a semi-supervised approach is employed, the models are only exposed to the normal instances of data. Furthermore, both models act as reconstruction models where their goal is to reconstruct the input. The reconstruction error has been widely used for detecting abnormal activities in various data representations [32; 55]. Such error is used as an indication to determine whether the data sequence is abnormal or not. For training the models, the data samples are fed into the model in multiple time-steps of length L where each time-step consists of x and y body keypoint values (as discussed earlier in this chapter). For example, a tensor shape of $(200, 30, 34)$ represents a total of 200 samples (S), where each sample contains 30 time-steps (L) and each time-step contains 34 features (F). The value for F depends on the pose estimation algorithm being used (PoseNet: 34, OpenPose: 50) and the value for L is determined empirically after a number of experiments as outlined in Section 4.5.1.7.

4.5.2.1 LSTM AutoEncoder

The first type of model is inspired by the work done by Malhotra et al. [55], which comprises of a number of LSTM units based on Encoder-Decoder networks. According to the authors, the model is able to learn the data representation of normal time-series events in a semi or unsupervised manner. The model was also proven to work on both short and long time-steps and can deal with both predictable and unpredictable data. Similar to the architecture proposed in [55] and [32], the architecture was set to consist of 3 LSTM layers where the first layer acts as the data encoder, the second is used to store the latent space representation, and finally, the last layer acts as the decoder (Sub-module 4.2.1). Figure 4.13 illustrates the architecture of the LSTM AutoEncoder (LSTM-AE). As outlined in Section 3.4, a bottleneck layer is typically used to force the network to learn a compressed representation of the data. The number of LSTM units of the encoder and the decoder within the architecture is the same, but the number of units of the bottleneck layer is half the number of units in the encoder/decoder layers. For instance, if the number of LSTM units in the encoder/decoder layers is 128, then the number of units in the bottleneck layer is 64 [93]. Since the model was trained to reconstruct the input itself, and therefore considered as a regression problem, the mean squared error loss function was used. This is a common loss function used in encoder-decoder models [43] as its aim is to calculate the mean squared difference between the input x and reconstructed output x'. For optimising the loss function, *Adam*, which is a popular choice amongst researchers [13; 16; 55; 94] was used. The parameters of the optimiser follow the default ones provided in the original paper [42], i.e the learning rate was set to 0.001 and β_1 and β_2 were set to 0.9 and 0.999 respectively.

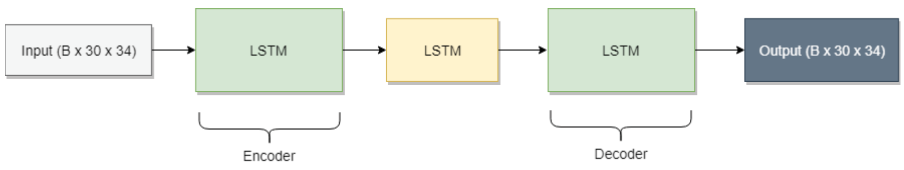


Figure 4.13: LSTM AutoEncoder: Model Architecture. Adopted from [55]

The hyper-parameters of this model were based on similar work done by [55] - LSTM units: 128, 64 and 128 respectively. However, in the interest of finding the ideal number of LSTM units, various experiments with different number of units were conducted. The values included 64, 128 and 256.

4.5.2.2 1DConv AutoEncoder

In a 2D image, a CNN is capable of identifying patterns by performing a couple of convolutions. Similarly, in a 1-dimensional vector, CNNs were also proven to work well with extracting discriminative features from sequences such as electrocardiogram signals [44] and sensor data from smartphones [99]. The CNN's ability to model data sequences is similar to a RNN, however with the advantage of being able to process data in parallel (unlike sequentially) [48]. Inspired by such abilities, an encoder-decoder model that aims to derive features that model the normal behaviour was proposed (Sub-module 4.2.2). The 1DConv AutoEncoder (1DConv-AE) is illustrated in Figure 4.14 and consists of three 1D convolution layers. The architecture is similar to the LSTM-AE model described earlier, in that the first layer is used to decode the sequence data, the middle one to force the network to learn a compressed representation of the data and finally the last layer is used to decode the input back to sequence data.

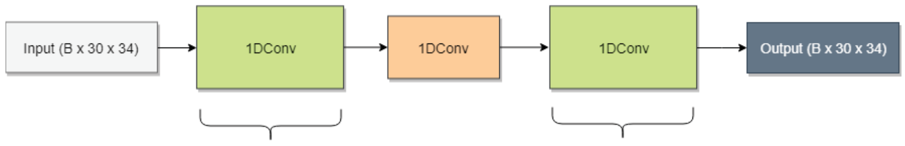


Figure 4.14: 1DConv AutoEncoder: Model Architecture

As for the loss and optimisation functions, the same approach described for the LSTM AutoEncoder model is employed for this network. Initially, the hyper-parameters of this model were based on similar work done by Chong and Tay [16] - filter sizes were set to 64, 32 and 64 respectively and the kernel size was set to 3. Different filter sizes were also tested.

4.5.3 Computing the Anomaly Score

In AutoEncoders, the difference between the input and the output (reconstructed input) is called the reconstruction error. When an AutoEncoder is trained, it is expected to produce low reconstruction errors if a normal event is fed to the network and a high reconstruction error if the sequence data is seen by the model [43]. The anomaly score α is determined by the reconstruction error (Sub-module 4.3). After training the model on the 'normal' data, the reconstruction error vector is computed using equation 4.1 where $X^{(i)}$ is the input sample and $X^{(i)\prime}$ is the reconstructed output of the model. The mean error for each vector is then computed in order for each data sample to be represented by one scalar value.

$$e^{(i)} = \left\| X^{(i)} - X^{(i)\prime} \right\|_2 \tag{4.1}$$

In order to find the best threshold τ that separates both classes, the Precision-Recall curve is plotted for various thresholds (Sub-module 4.4). Such curve is used to summarise the trade-off between the true positive rate (recall) and the positive predictive value (precision). The threshold τ that maximises the F-Score F_β is used as a threshold to determine whether a data sample is 'abnormal' or not. An instance is classified as 'abnormal' if the anomaly score α is greater than τ (Sub-module 4.5). The choice of β, the thresholds selected, and how the F-score is calculated is explored further in the next chapter in Section 5.3.2.

4.5.4 Training the models

All the architectures outlined in this chapter were constructed using Keras[7] as frontend and Tensorflow[8] as backend. Being the official frontend for Tensorflow, Keras is a high-level API for creating and running artificial neural nets written in Python. According to a ranking computed by Jeff Hale[9], Keras is the #2 most mentioned framework in scientific papers. It is a favourite among researchers in the field as its consistent and simple API allows a researcher to implement model architectures in a productive manner.

All models were trained on the latest version of Windows 10 using Python3, Keras and Tensorflow. The machine was equipped with an Intel Core i7 8700k (consisting of 6 cores) clocked at 3.70Ghz, 16GB RAM and an NVIDIA GeForce GTX 1080 as the graphical processing unit (GPU). The GPU version for Keras and Tensorflow were installed to take advantage of the multiple cores of the GPU.

In the next chapter, an analysis on the results together with how the models were evaluated is presented.

[7] `https://keras.io`
[8] `https://www.tensorflow.org/`
[9] `https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a`

4.6 Comparing sensor data with estimated pose data

In human action recognition and anomaly detection, models trained on data generated from sensors such inertial wearable sensors and depth cameras produced promising results [13; 38; 97; 98; 99]. But how do models perform if they were trained on data estimated by another model (in this case by estimating the human's posture), without the use of sensors? How do they compare with each other? This section aims at outlining the method employed to answer these questions.

In order to compare between sensor data with pose estimated keypoints (from RGB footage), a dataset that recorded both the RGB footage and the sensor data at the same time is required. The UTD-MHAD dataset is a good fit as it comprises of 3 modes that were recorded in a simultaneous manner: RGB footage, skeleton (3D) data and inertial sensor data. Figure 4.15 illustrates the process involved to compare sensor data with estimated pose data on human action recognition. In this section, an overview of the dataset is given. Furthermore, a description on each process shown in the flow chart is outlined.

4.6.1 UTD-MHAD Dataset

A publicly available[10] dataset named UTD-MHAD [11] consists of a total of 27 different human actions recorded in four types of modes: RGB video, Depth video, skeleton positions, and finally inertial signals. The video and skeleton data were acquired from a Kinect camera and the inertial signals were collected from wearable inertial sensors. All the modes were recorded at the same time; i.e. the subjects were recorded using the four mentioned modalities. The footage was recorded in an indoor environment and the actions were performed by 8 subjects (4 males and 4 females). Each action was repeated 4 times by each subject. A total of 861 data sequences (after removing 3 corrupted sequences) were recorded. Some of the 27 actions that were recorded include daily activities (e.g. sitting, standing, walking, and jogging), sport actions (e.g. boxing, bowling, and baseball swings), training exercises (e.g. lunges, squats, and arm curls) and hand gestures (e.g. swiping, clapping, waving, and crossing arms). Figure 4.16 depicts a single frame example of the first 5 actions that were recorded.

For recording the inertial sensor signals, an inertial sensor as depicted in Figure 4.17 was worn on the subject's right wrist or thigh (depending on the action being per-

[10]`http://www.utdallas.edu/~kehtar/UTD-MHAD.html`

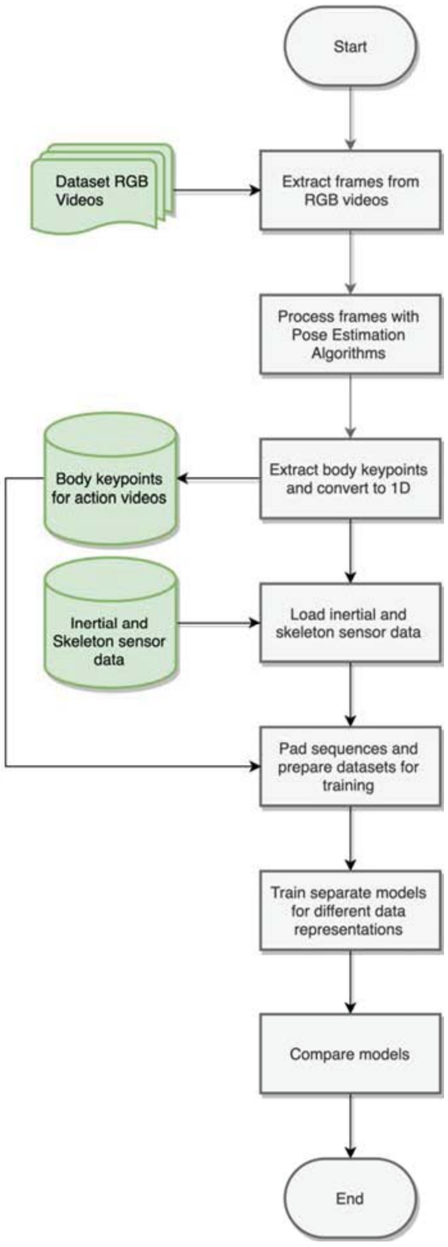

Start
Dataset RGB Videos
Extract frames from RGB videos
Process frames with Pose Estimation Algorithms
Body keypoints for action videos
Extract body keypoints and convert to 1D
Inertial and Skeleton sensor data
Load inertial and skeleton sensor data
Pad sequences and prepare datasets for training
Train separate models for different data representations
Compare models
End

Figure 4.15: Flow chart showing the processes involved to compare sensor data with pose estimated data in a human action recognition task

formed). Such sensor is capable of capturing 3-axis acceleration, 3-axis angular velocity and 3-axis magnetic strength. The magnetic readings were not considered due to the lack of magnetic fields in practice. A micro-controller together with a low energy bluetooth device were used to record and stream the data wirelessly to another device.

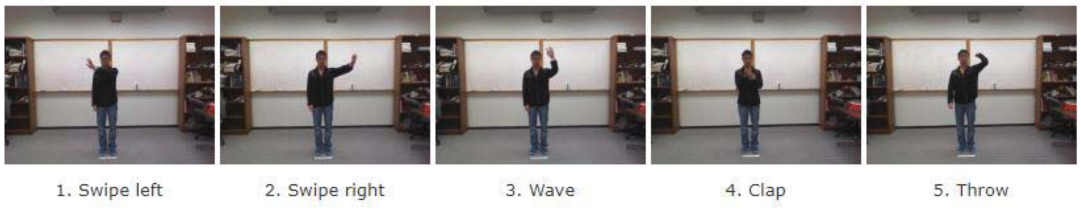


Figure 4.16: UTD-MHAD Dataset: examples of actions.
Source: [11]

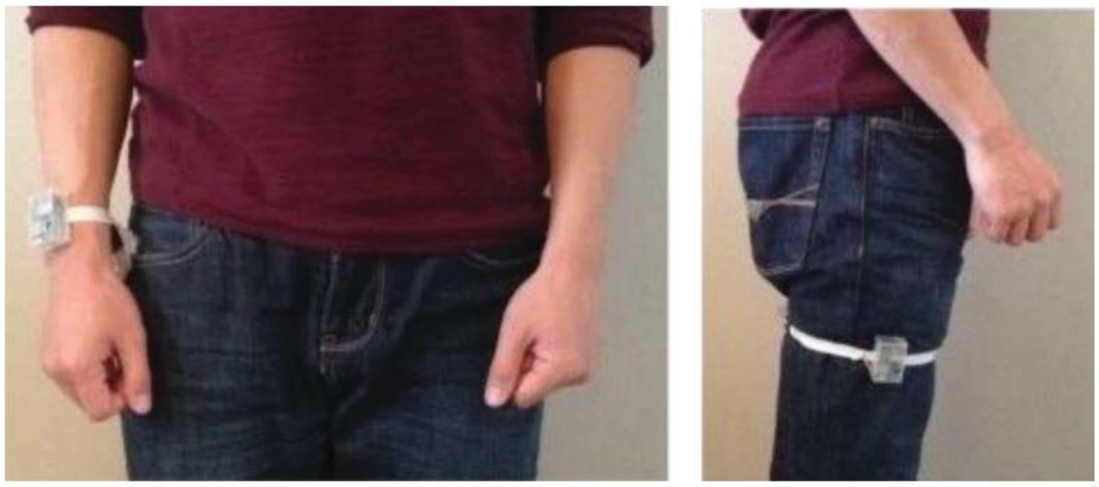


Figure 4.17: UTD-MHAD Dataset: Wearable sensor placement.
Source: [11]

The RGB video footage and the skeleton joint data were captured by one Kinect camera, which was placed 3 meters in front of every subject to ensure that the whole body appeared in the frame. Since the sampling rates of the wearable sensor and the frame rate of the camera were different, a time-stamp for each sample (for every action) was recorded in order to synchronise all the modes. In their paper, the authors also presented a diagram (Figure 4.18) which shows all four modes recorded for the action 'basketball-shoot'.

Since the RGB videos of all the actions covering the full body were made available, the body keypoints can be extracted using a pre-trained pose estimation model. This allows comparison between data gathered from the wearable sensor and the body keypoints estimated by the pose estimation model. Moreover, such extracted body keypoints can be compared with skeleton (both 3D and 2D) extracted from sensors.

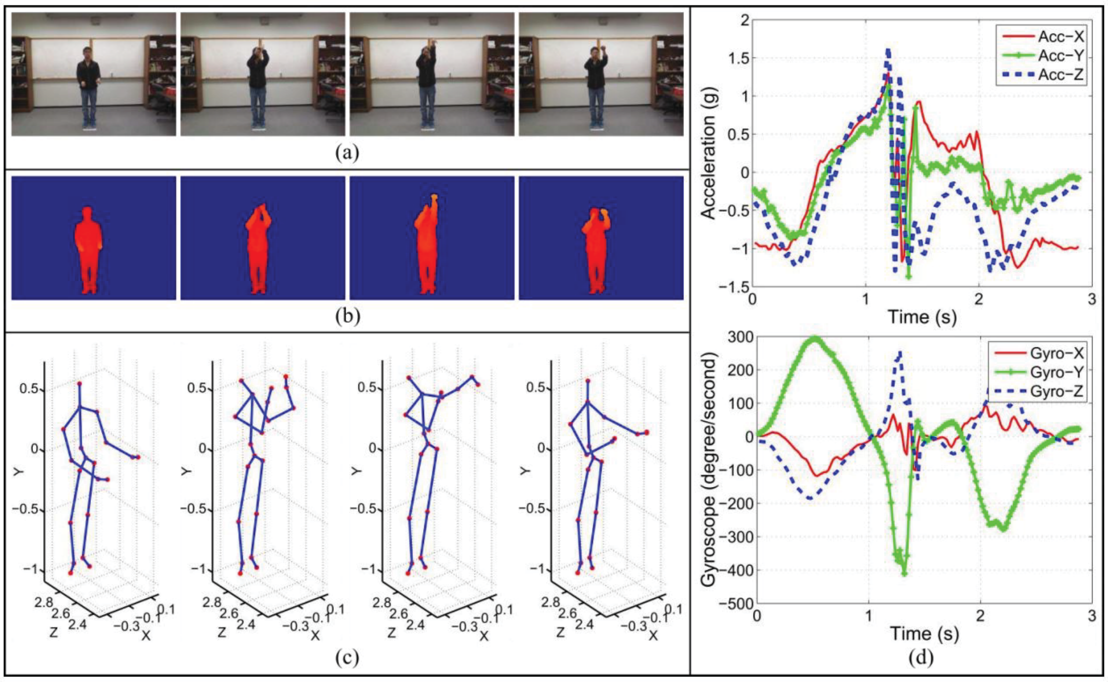


Figure 4.18: UTD-MHAD Dataset: Example of all four modes recorded for the action 'basketball-shoot': (a) RGB video footage, (b) depth images, (c) skeleton joint frames, (d) inertial sensor data
Source: [11]

4.6.2 Data Pre-processing

Initially, the RGB footage from the UTD-MHAD dataset was processed as explained in Sections 4.3 and 4.4 in that all the frames for every video were extracted and processed with both pose estimation models (PoseNet and OpenPose). In total, 44,982 frames were extracted and processed.

Table 4.1 summarises the data used in this experiment. In total, each dataset contains 861 sequences. The first dataset in the table refers to the RGB footage from the UTD-MHAD dataset that was processed by PoseNet. The model returns the x & y position of 17 keypoints. Such keypoints were converted to 1D and thus the number of features amount to 34. In this experiment, the whole action sequence (list of frames) is fed to the model as one sample. This allows the model to learn the temporal features of every action. However, the action sequences have variable length. In view of this, due to the fact that samples are trained in batches, they need to have the same length. A common approach is to pad the sequence with a padding value (typically 0) in order to transform

Dataset	Format	Description	Tensor shape
RGB Videos (->) PoseNet	Video -> JSON	17 body keypoints for every frame (x,y)	(861, 96, 34)
RGB Videos (->) OpenPose	Video -> JSON	25 body keypoints for every frame (x,y)	(861, 96, 50)
Inertial sensors	MATLAB Array	3-axis acceleration and 3-axis rotation signals	(861, 326, 6)
Skeleton data (2D)	MATLAB Array	20 body keypoints (x,y)	(861, 126, 40)
Skeleton data (3D)	MATLAB Array	20 body keypoints (x,y,z)	(861, 126, 60)

Table 4.1: List of datasets used to compare between pose estimated data and sensor data

the data to a fixed-length shape [47]. This was done to each dataset listed in Table 4.1 by finding the maximum sequence length and padding the rest of the sequences. The tensor shape shown in the table follows the following format:

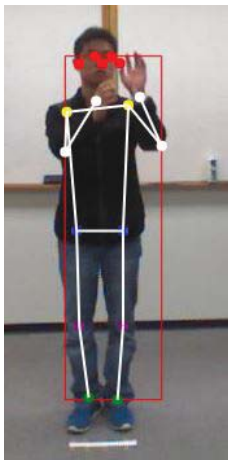
$$(\textit{number of samples}, \textit{number of time steps}, \textit{number of features})$$

where the number of time steps refer to the sequence length and the number of features indicate the amount of features in every time step (or sequence).

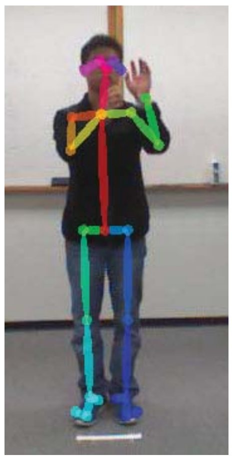
The skeleton data provided in the UTD-MHAD dataset was provided as a 3-dimensional array of shape (20 x 3 x n_frames). The first dimension represents the number of body keypoints, the second represents the x, y and z of every point and the third dimension refers to the number of frames that make up the sequence. In order to include a 2D representation of the data, it was decided to drop the z axis from the 3D skeleton data. This allowed for an additional dataset in the comparison experiment. Figure 4.19 illustrates the four types of skeleton representations (apart from the inertial dataset) that were trained and compared.

4.6.3 Model Architecture

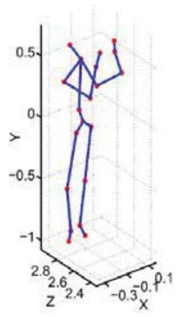
The model architecture used to compare the aforementioned representations is a supervised model that aims at classifying given sequences as human actions. The model architecture is depicted in Figure 4.20 and was inspired by recent work in human action recognition that used 1D Convolutions to identify human actions [13]. The use of 1D CNNs was also explored by Kiranyaz et al. [44] and proven to work well with extracting discriminative features from sequences. The input shape of the model depends on the



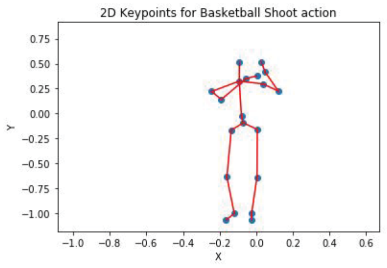
(a) 2D Estimated Skeleton, PoseNet



(b) 2D Estimated Skeleton, OpenPose



(c) 3D Skeleton, Kinect



(d) 2D Skeleton, Kinect (removed z axis)

Figure 4.19: The four types of skeleton representations used to train separate models. Action: 'Basketball Shoot' from UTD-MHAD dataset. Top images were generated by PoseNet and OpenPose respectively wheras the bottom images depict 3D and 2D plots respectively of the skeleton data.

dataset. For instance, the input shape depicted in Figure 4.20 refers to the PoseNet estimated skeleton data where the first value (marked *B*) refers to the batch size, the second value refers to the amount of time-steps and finally the last value refers to the amount of features for each time-step. The rest of the architecture is inspired by the work in [71] where consecutive convolutional layers (2 in this case) are used to extract spatial and temporal features from the data. Such features are then flattened and fed to an LSTM layer to model the temporal aspect of the sequence. Finally, the output is passed to 2 fully-connected (dense) layers for final classification where the *softmax* activation function is used to normalise the predictions in a probability distribution. Such technique is common in classification tasks where multiple layers are used [45]. Dropout layers were also added to the network as a regularisation technique to reduce over-fitting [82].

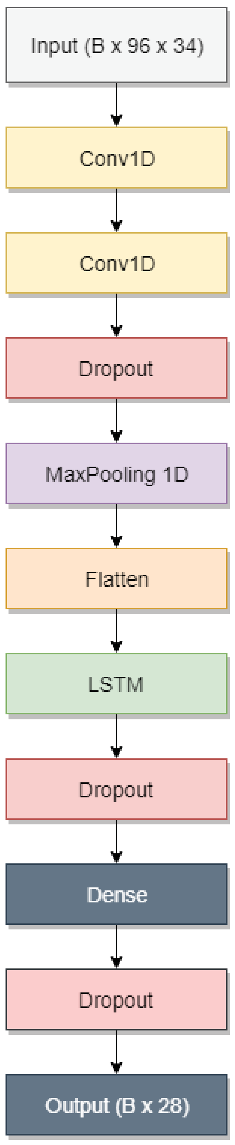


Figure 4.20: Model architecture used to compare different data representations (through action recognition). Convolution layers adopted from [13].

The activation functions for the convolutional and the dense layers were chosen to be 'ReLU'. It was shown that such activation function makes the training process faster [45]. On the other hand, the activation function for the LSTM layer was set to 'tanh' because the output from such function can be both positive or negative (unlike ReLU), which allows the state to increase or decrease accordingly [52]. Since the task at hand is a multi-class classification problem, the categorical cross entropy function is used as a loss function. This function is able to handle multi-class probabilities and thus is appropriate for such problem [26]. Finally, to minimise the loss function and therefore optimise the network during training, an extension of stochastic gradient descent called *Adam* was used. According to [42], this method outperforms all other techniques. Moreover, recent studies show that *Adam* have shown capable results and therefore a very popular choice for optimising the loss function [13; 16; 55; 94]. The parameters of the optimiser follow the default ones provided in the original paper [42], i.e the learning rate was set to 0.001 and β_1 and β_2 were set to 0.9 and 0.999 respectively. The architecture was also designed using Keras with Tensorflow as the backend.

4.6.3.1 Training the models

In order to compare all 5 data representations tabulated in Table 4.1, 5 models were trained separately. This allows the different data representations to be compared with each other and therefore one could measure the effectiveness of estimated keypoints from a 2D image over sensor data. An overview of the techniques used to train the models as well as how these were evaluated and compared is given in the next chapter in Section 5.2.

4.7 Conclusion

This chapter aimed at providing the reader with the techniques employed to reach the aims and objectives of this study. In the first sections of this chapter, the proposed method is depicted in a block diagram which shows the main components of the system: processing raw video footage, extracting body keypoints, pre-processing the data, training the model and finally predicting data sequences. A more detailed overview of such components together with the installation instructions are presented in Appendix B.

The chapter also provided the reader with a detailed description on how the extracted body keypoints were pre-processed and prepared to train different model architectures. A detailed description of how experiments were conducted to determine whether human body keypoints extracted from RGB images are comparable to sensor data collected from humans performing various actions was also presented.

The next chapter aims at presenting the results of the conducted experiments as well as at evaluating the system to prove that the proposed technique is effective in the real-world.

5

Results and Evaluation

5.1 Introduction

This chapter aims at providing the reader with the results of the methods proposed in the previous chapter. Furthermore, an overview of the evaluation criteria used to evaluate the system is given.

This chapter is divided into three main parts. In the first part, the results and evaluation techniques for the comparison between sensor and estimated data are presented and discussed. Positive results from this experiment indicate that pose estimation data from a 2D image can be informative enough to describe a human action and thus can replace the use of sensors. The second part deals with the presentation of the results and evaluation metrics used to evaluate the performance of the models to detect abnormal behaviour from video footage. In this section, the model's output is used to indicate whether a given data sequence is abnormal or not. Lastly, in the final and third part, the results of this study are compared to similar work done on the same dataset.

5.2 Comparison between sensor and estimated data

This evaluation aims at evaluating the method outlined in Chapter 4, Section 4.6 and at answering the following research question:

How would human action classification models perform if they were to be trained on pose data estimated from a 2D image instead of sensor data?

This question begs the following hypothesis:

Hypothesis: Pose estimated data from a 2D image can be informative enough to describe a human action.

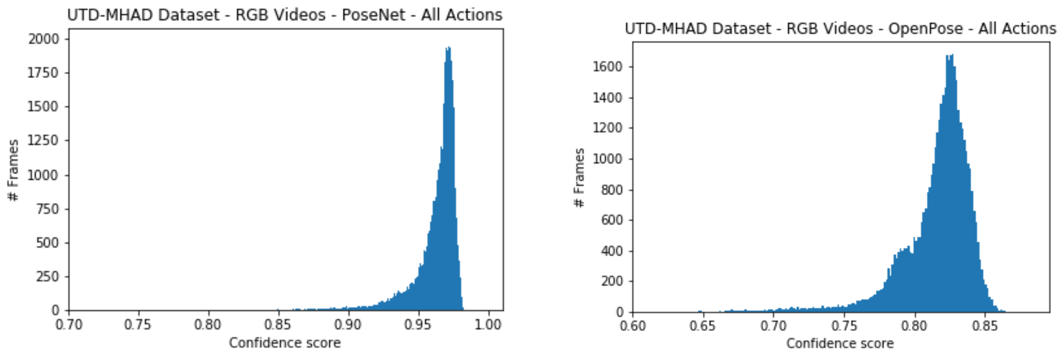
As outlined in the previous chapter, 5 different data representations were constructed to answer this question. All data representations represent the 27 human actions from the UTD-MHAD dataset. These include:

1. Body keypoints estimated from 2D images using PoseNet pose estimation algorithm
2. Body keypoints estimated from 2D images using OpenPose pose estimation algorithm
3. 3D Skeleton data extracted from a camera capable of determining depth
4. 2D Skeleton data created from the 3D Skeleton data by dropping the z axis
5. Data from a wearable inertial sensor

Both PoseNet and OpenPose pose estimation algorithms managed to estimate the majority of the body keypoints from the all video frames. Two histograms of the confidence scores reported by both models are depicted in Figure 5.1. This diagram includes all the scores for all actions (861) which in total amount to 44,955 frames. It was observed that on average, the overall confidence scores for PoseNet are higher than those of OpenPose. This is because the majority of the confidence scores reported by PoseNet are between 0.9 and 0.98, whereas the ones generated by OpenPose are between 0.7 and 0.88.

A model for each data representation was trained for comparison purposes. In order to assess the model performance and maximise the available data, stratified *K*-fold cross validation (CV) was utilised. In these experiments, stratification was used to ensure that each fold contains a good representation of the different actions (or classes). According to James et al. [39], *k* values of 5 or 10 have empirically shown that test error rate estimates do not suffer from high variance or high bias. In view of this and to reduce the cost of training, a *k* value of 5 was chosen to conduct the comparison experiments. Furthermore, the best model with the lowest loss value recorded was saved and used as the final model.

To find the best hyper-parameters for each model, a grid search for the parameters tabulated in Table 5.1 was conducted. The selected values are common ones usually



(a) PoseNet (b) OpenPose

Figure 5.1: Histogram showing confidence scores of extracted poses for UTD-MHAD RGB footage for both PoseNet and OpenPose

Hyper-parameter	**Values**
CNN filters	{64, 128, 256}
CNN kernel size	{1,2,3,4}
LSTM Units	{64,128}
Number of subsequences	{1,2}
Dropout rates	{0.5, 0.7}
Total	**96 combinations**

Table 5.1: The parameter grid used to tune the hyper-parameters

used as hyper-parameters in similar work in the area [13; 16; 55]. All 5 models were separately trained using 96 different combinations of hyper-parameters. Furthermore, since 5-fold cross validation was employed, each model was trained for 5 times. This amounts to a total of 2,400 trained models.

Since stratification was used, each action class is equally represented, both in the training and the testing sets. Such equal balance makes the accuracy metric (Equation 5.1) a proper fit for this problem [26].

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

Where TP refers to the True Positives, TN refers to True Negatives, FP refers to False Positives and FN refers to False Negatives.

The results for such experiments are illustrated in two ways. In Figure 5.2, a bar chart illustrates the mean accuracy score for the best model for each data representation. The bars coloured in blue represent the estimated pose data from RGB videos whereas the bars coloured in black represent sensor data acquired from a depth camera (skeleton data) and a wearable inertial sensor. The results are also tabulated in Table 5.3 where the mean accuracy and standard deviation (st.d) of the 5-fold CV are reported. In this table, only the results for the best performing model (for each data representation) are reported. Additionally, Table 5.2 shows a summary of the hyper-parameters used for each data representation which yielded the best results.

Dataset	**Batch Size**	**Dropout rate (%)**	**CNN Filters**	**CNN Kernel Size**	**LSTM Units**	**# Sub-sequences**
Posenet	32	0.5	64	1	128	1
OpenPose	64	0.5	128	1	64	1
Skeleton 3D	64	0.5	64	1	128	1
Skeleton 2D	64	0.5	256	1	128	1
Inertial	64	0.5	64	1	128	1

Table 5.2: Best hyper-parameter combination for each data representation

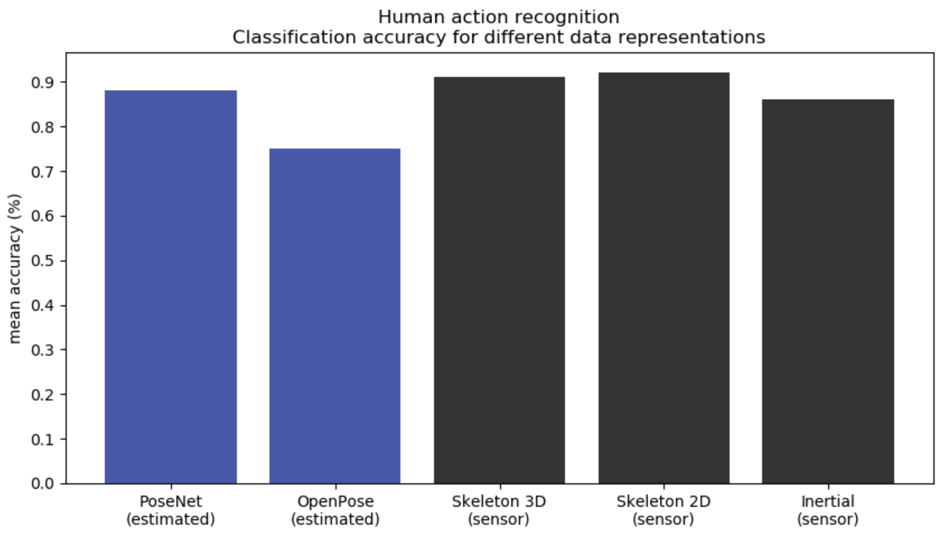


Figure 5.2: Human action classification accuracy for different data representations.

These results demonstrate that pose estimated data from RGB videos is indeed com-

Data Representation	**Mean accuracy**	**st.d**
PoseNet (estimated)	0.88	0.09
OpenPose (estimated)	0.75	0.1
Skeleton 3D (sensor)	0.91	0.26
Skeleton 2D (sensor)	0.92	0.14
Inertial (sensor)	0.86	0.04

Table 5.3: Human action classification accuracy and standard deviation (st.d) for different data representations for *k*-fold cross validation with *k*=5

parable to sensor data from depth cameras such as the Kinect. Moreover, this experiment shows that data estimated by PoseNet is indeed more informative than data gathered from an inertial sensor. This could be explained by the fact that data estimated from RGB videos captures the motion of all moving body parts whereas the inertial sensor only captures the movements of where it is attached. The results also demonstrate that the data representation estimated by PoseNet is better than that of OpenPose itself. This explains the difference in confidence scores reported by both algorithms.

The results also indicate that human action classification seems to perform best when skeleton data is generated by the depth camera. Taking into consideration that no extra hardware was utilised, the results from this experiment are quite promising, as they show that pose estimated data from a 2D image can be informative enough to describe a human action. The key finding from these results is that pose estimated data compares well to sensor data when it comes to human action classification from video footage. Thus, pose estimated data from a video can indeed be informative enough to recognise a human action being performed. This allows advanced understanding of existing video footage for various applications such as health care and surveillance.

5.3 Detecting abnormal behaviour

This section aims at presenting the evaluation techniques for the method outlined in Chapter 4, Section 4.5 and at answering the following research question:

Is it possible to train a semi-supervised model using solely body estimated keypoints to detect anomalous sequences?

In consideration of the above question, the following hypothesis is being set:

Hypothesis: Pose estimated data can be informative enough to represent normal activities and therefore flag abnormal activities otherwise

5.3.1 Evaluation Criteria

Many researchers in this area [93; 98; 100] suggest the use of ranking-based evaluation metrics when dealing with an unbalanced dataset. Instead of using accuracy, they recommend metrics such as the Area Under the Receiver Operating Characteristic curve (AUROC) or the Area Under the Precision-Recall curve (AUPR) [8]. In a dataset where a particular class has the majority of instances, an accuracy-based metric is not ideal mainly due to the fact that the class with the majority will be favoured.

Davis and Goadrich [17] and Saito and Rehmsmeier [72] argue that in a skewed dataset, the Precision-Recall curve is more informative than the ROC curve as it plots the precision and the recall of the classifier at various thresholds, which takes into account the data imbalance. This allows for a “more informative picture of an algorithm’s performance”. On the contrary, the ROC curve represents the relationship between the true positive rate (TPR) which also known as recall, and the false positive rate (FPR) - which is different from the precision score. The recall or TPR measures the fraction of positive samples that were correctly classified and the FPR measures the fraction of negative samples that were incorrectly classified as positives. Lastly, the precision metric measures the fraction of samples classified as positives that are truly positive.

In binary classification, the data is divided into two classes - positives (P) and negatives (N). In order to present the performance of a classifier, a confusion matrix is usually used. With reference to this study, positive samples refer to abnormal activities and negative samples refer to normal activities. Such matrix is usually formulated as a table that tabulates four types of outcomes: True Positives (TP), True Negatives (TN), False Positives (FP) and False Negatives (FN). The aforementioned metrics such as the precision and recall uses the outcomes from the confusion matrix to evaluate the performance of a given model. The equations for the recall, precision and false positive rate metrics are presented below.

$$\text{Recall/True Positive Rate} = \frac{TP}{TP + FN} \tag{5.2}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5.3}$$

$$\text{False Positive Rate} = \frac{FP}{FP + TN} \tag{5.4}$$

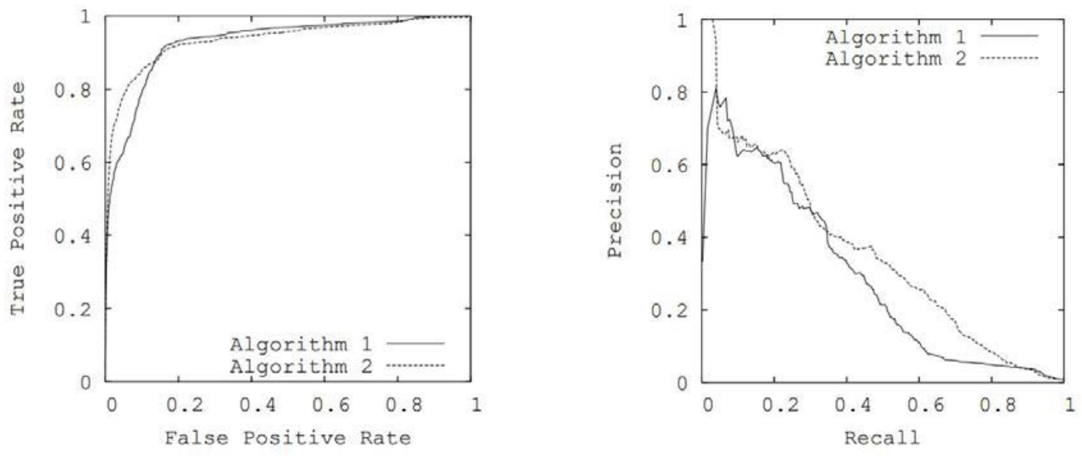


Figure 5.3: ROC Curve vs PR Curve. (Source: [17])

The visual representation of an ROC curve is different from a PR curve. In an ROC curve, the performance of an algorithm is best when the space is in the upper left-hand corner of the graph. On the other hand, an optimal PR curve would have its space in the upper right-hand corner of the graph. Figure 5.3 shows a sample for an ROC and PR curve that represent the same 2 algorithms. Such graphs represent two learned models on a highly unbalanced dataset conducted by Davis et al. in [18]. In the first ROC graph it is clear that both Algorithm 1 and 2 are comparable. However, the PR graph on the right shows that the predictive power of Algorithm 2 is higher than Algorithm 1. Such difference exists because the number of negative samples are larger than the number of positive samples. The Area Under the Curve (AUC) is then used to measure the performance of the model. An AUC value of 1 would indicate that the model is perfect.

In this study, the detection of true positive (abnormal) samples is vital. This is because predicting false positives (normal events classified as abnormal) is of less risk than classifying an abnormal event as normal (false negative). Therefore, more weight to the recall metric is to be given. On the other hand, the precision metric is also important as the model should be able to classify normal events correctly. In statistical analysis, the F-Score (equation 5.5) metric provides a weighted average of the precision and recall metrics [12]. The β parameter is used to control the balance between precision and recall. When β is equal to 1, the harmonic mean of the precision and recall is given. However, when $\beta < 1$, more weight is given to the precision score and when $\beta > 1$, more weight is given to the recall score. To evaluate the predictive power of the models a β value of 4 was chosen. This value allows the F-score to be more recall-oriented. Furthermore, it is the same value used by Debard et al. [19] in their paper on the same dataset that was used in this study.

$$F_\beta = \frac{(\beta^2 + 1) \cdot (\text{precision} \cdot \text{recall})}{(\beta^2 \cdot \text{precision}) + \text{recall}} \tag{5.5}$$

In light of the above, in the rest of this chapter, the AUC of the PR curve will be used to measure the performance of the trained models. This is because the number of samples for both classes is not equally balanced.

5.3.2 Computing and finding the best threshold

As outlined in Section 4.5.3, in order to find the best threshold τ that separates both classes, the PR curve is plotted for various thresholds. This section aims at describing the process undertaken to establish various thresholds. After an AutoEncoder model outlined in Section 4.5.2 is trained, the reconstruction errors of the training set (normal activities) together with the test sets (that contain both normal and abnormal samples) are plotted in a histogram. In an ideal situation, the reconstruction errors for the normal activities should be on the low side whereas the reconstruction errors for the abnormal activities should be on the high side. This is because the model was trained in a semi-supervised approach, i.e. to reconstruct the normal activities only.

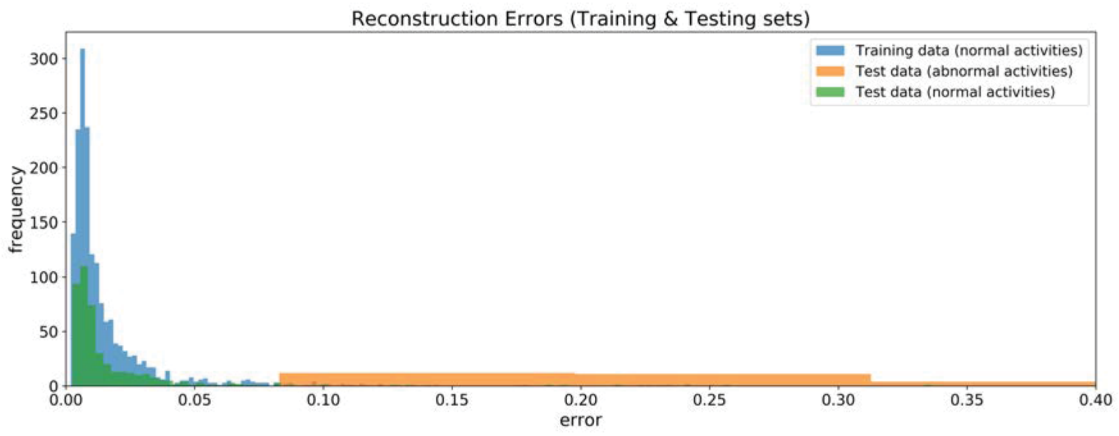


Figure 5.4: Reconstruction errors for Training and Testing Sets

A sample histogram for one of the models is shown in Figure 5.4. The histogram shows 3 distributions, each representing a set of reconstruction errors. The blue distribution represents the reconstruction errors of the training data itself, i.e. the same training data that was used to train the model, whilst the green distribution represents the reconstruction errors of unseen normal activities. Lastly, the orange distribution represents the reconstruction errors of the abnormal activities, which are also unseen by the model. In this case, the histogram demonstrates that the model was able to reconstruct the normal activities quite well as the error range is similar to that of the training data. Here, it is noticeable that the model reconstructed the abnormal sequences with higher error and thus this shows that the model can be used to detect normal and abnormal sequences. With direct reference to this histogram, the optimal anomaly threshold τ would be ≈ 0.09 where instances $> \tau$ are classified as abnormal and instances $< \tau$ are classified as normal.

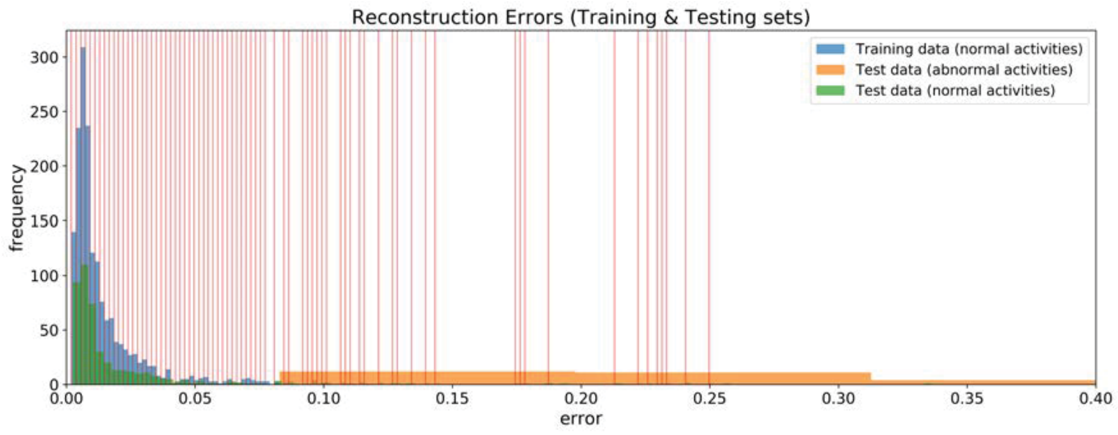


Figure 5.5: Various thresholds considered at different bins

In order to determine various thresholds, a non-parametric approach was employed. At every bin in the histogram, the area is calculated and considered as a threshold. At each threshold, the precision and recall metrics are calculated and used to plot the PR curve. This is depicted in Figure 5.5 where the various thresholds considered are represented in red vertical lines. The AUC of the PR curve is then used to measure the model's performance. Additionally, at each threshold, the F_β score is measured. The maximum F_β score is then used to find the threshold that best segregates the classes.

5.3.3 Evaluation

This section aims at outlining how the all 4 modules depicted in the Section 4.2 were linked together in order to train the models to detect abnormal behaviour. Since a preliminary analysis was conducted, this section is divided into two parts. The first part outlines the analysis done on the KTH dataset whilst the second part presents the method conducted on the ADL dataset. In each part, an overview of the dataset utilised is also given.

In both datasets, the LSTM-AE and 1DConv-AE models outlined in 4.5.2 are trained and compared. Moreover, the same method for preparing the training and testing sets was conducted for both datasets.

In order to evaluate the models, k-fold Cross Validation (CV) with $k = 5$ was used. According to James et al. [39], k values of 5 or 10 have empirically shown that test error rate estimates do not suffer from high variance or high bias. The training set, which consists of data from the normal class only, is randomly shuffled at sample level (i.e. the sequence order in each sample is preserved) and split into 5 folds. In order to be able to generate the same folds for every model, the *random_state* parameter was set to fixed number. This allows the ability for models to be compared with each other. In each fold, since k is set to 5, the training set is split into two sets ($\frac{k-1}{k} : \frac{1}{k}$): one used to train the model (80%) and the other (20%) to test whether the model is able to reconstruct 'normal' (or negative) actions correctly. The other test set containing the 'abnormal' (or positive) actions is pre-processed with the same configuration and added to the test set (which already contains the 'normal' sequences). Additionally, 20% of the 80% data used to train the model is used as a validation set. Such set is utilised to provide an unbiased evaluation of the model being trained. Similar to the approach undertaken by Chong and Tay [16], the models were trained for 200 epochs in mini-batches of size 64. Early stopping [64] with *patience = 5* was used to stop further training when the validation loss stops improving (in this case when it stops decreasing). Such practice is common amongst researchers [13; 16; 55; 94] as it prevents the model from over-fitting.

As outlined in the previous chapter, the models were trained on frame sequences rather than individual frames. In the rest of this document, 1 data sequence that consists of 20 frames is represented as $(1, 20, F)$, where F represents the number of features making up each frame. In line with this, the tensor shape generated after the pre-processing steps follows the following format:

number of samples (S) x window length (L) x number of features (F)

where S refers to the amount of data samples, L refers to the number of time-steps (frames) and finally F refers to the amount of x and y values that represent each body keypoint (34 for Posenet, 50 for OpenPose).

5.3.4 Analysis on the KTH Dataset

To evaluate the proposed method, a preliminary analysis was conducted on the KTH dataset [74]. The KTH dataset is a popular dataset for human action recognition. It contains a total of 2,391 sequences consisting of 6 different types of human actions: walking, jogging, running, boxing, hand waving and hand clapping. The actions were performed by 25 subjects in four different environments: outdoors, outdoors with scale variation, outdoors with different clothes and indoors. Figure 5.6 illustrates some examples of human action sequences in different environments. Having a length of 4 seconds in average, the footage was recorded with a static camera at 25 frames per second and the resolution was down-sampled to 160 by 120 pixels.

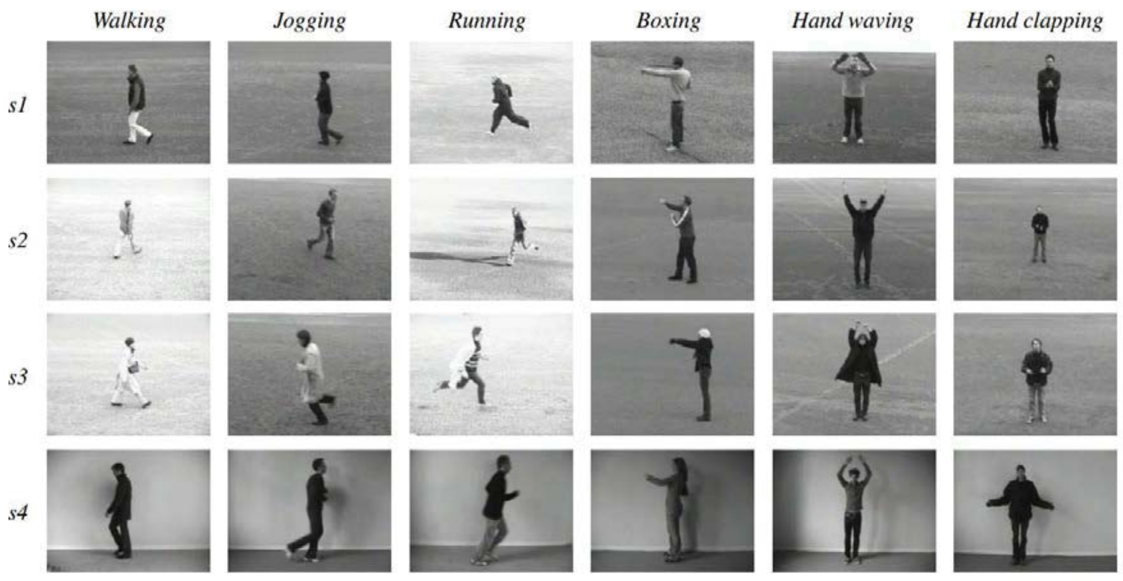


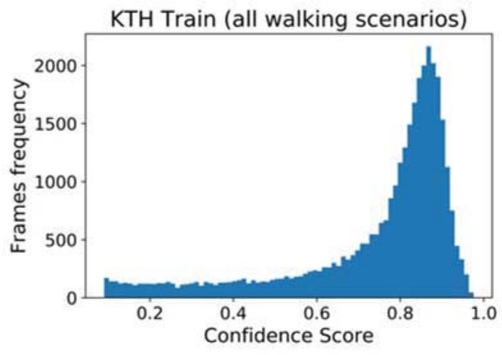
Figure 5.6: KTH Dataset: examples of sequences in different environments. Source: [74]

For this analysis, only the footage for actions *walking* and *running* were considered. The main idea behind this selection was to determine whether the model is capable of learning ‘walking’ as normal behaviour and ‘running‘ as abnormal. Initially, the footage was converted to frames as outlined in Section 4.3 and the body keypoints were extracted using PoseNet as described in Section 4.4. Such body keypoints were then preprocessed as outlined in Section 4.5.1. Table 5.4 tabulates the amount of frames extracted for each action.

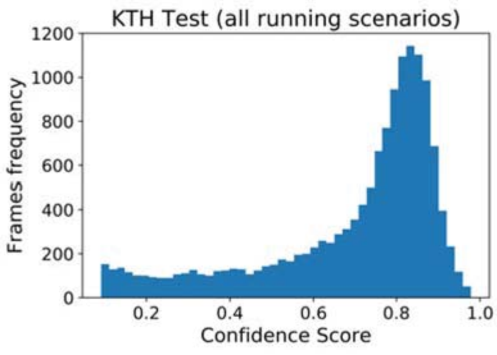
Dataset	Scenario	# Frames
KTH	Walking	65,795
KTH	Running	35,505
Total # frames extracted		**101,300**

Table 5.4: Total number of frames extracted for the KTH dataset.

Two histograms of the confidence scores reported by PoseNet are depicted in Figure 5.7. These histograms show the confidence score distribution for all the frames that scored between 0.1 - 1. The rest (0 - 0.09) were not included because they consist of empty frames (i.e. no human subject). The average confidence scores for ‘walking’ and ‘running’ were 0.74 and 0.70 respectively. This shows that the pose estimation algorithm managed to predict the majority of the frames with 0.70 confidence score.



(a) PoseNet: KTH Confidence Scores for Walking action



(b) PoseNet: KTH Confidence Scores for Running action

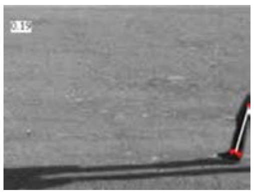
Figure 5.7: PoseNet confidence scores for KTH walking and running frames

5.3.4.1 Training the models

Both the LSTM-AE and 1DConv-AE models described in Section 4.5.2 were used to train to reconstruct the *walking* actions. A total of 4 data configurations were created for this experiment. Such configuration parameters are tabulated in Table 5.5. As depicted in Figure 5.8, PoseNet manages to predict the correct body keypoints even if the human subject is partially visible. However, when only two body parts or so are visible in the frame, the model fails to predict the keypoints. The minimum confidence score for these experiments was determined after manually observing the processed frames. It was observed that a score of 0.4 and upper produces promising results where the majority of the keypoints are correctly predicted. This also includes frames where the human subject is partially visible.

Parameter	**Values**
Minimum Confidence Score (MCS)	0.4
Window Length (WL)	20
Window Overlap (WO)	True, False
Remove Low Scores (RLS)	True, False

Table 5.5: Data configurations for KTH dataset


(a) KTH: Person entering the scene from the right. Score:0.29

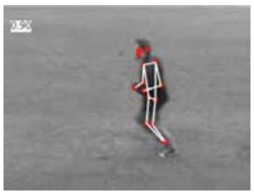
(b) KTH: Person leaving the scene. Score: 0.38

(c) KTH: Person left the scene. Score: 0.00

(d) KTH: Person entering the scene from the left. Score: 0.11

(e) KTH: Person running. Score:0.71

(f) KTH: Person walking. Score: 0.91


(g) KTH: Person running. Score: 0.90

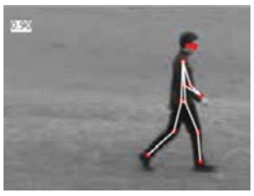

(h) KTH: Person walking. Score: 0.90

Figure 5.8: KTH Dataset. Upper row shows output from PoseNet for challenging scenes. Lower row shows scenes where subject is fully visible

In this experiment, the window length L of each sample was set to 20 frames. This models ≈ 0.6 seconds of the action which, in literature, seems to be an acceptable value to model human actions [6]. Furthermore, it was observed that both walking and running actions are represented well in 20 frames. The training and test sets were prepared as described in the introduction of this section (Section 5.3.3). Since a semi-supervised approach is proposed, the training data consisted of 'walking' instances only, i.e. the model was not exposed to the 'abnormal' instances. In light of this, the models were expected to classify any action not 'walking' as abnormal as the model was not exposed to it. The test set comprised of all the 'running' (abnormal) sequences and 20% of the 'walking' (normal) instances.

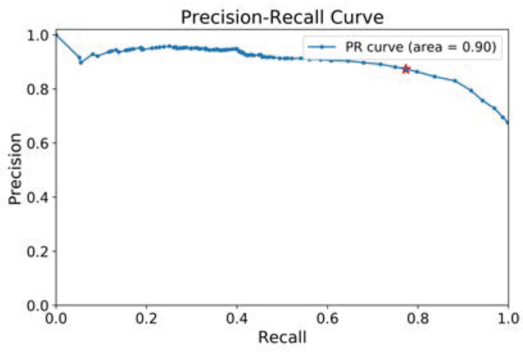
5.3.4.2 Results and their interpretation

Table 5.6 tabulates the results for the data configurations shown in Table 5.5. Both the PR AUC and the Fβ scores reported are the mean values of 5 models trained for each fold (since k was set to 5). In this table, the standard deviation (st.d) of the mean values applies to both the PR AUC and Fβ measures as they happen to be identical. One can note that both models performed very similar to each other. The number shown after the model name indicates the number of units or filters used in the models (Section 4.5.2).

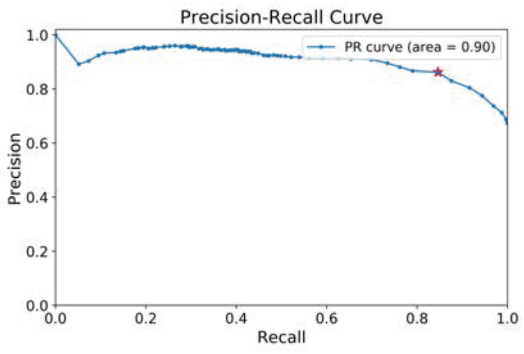
Model	Window Overlap	Remove Low Scores	Mean PR AUC	St.d	Mean Fβ Score
LSTM-AE 128	✓	✓	0.78	0.02	0.73
LSTM-AE 128	✓	✗	0.87	0.01	0.76
LSTM-AE 128	✗	✓	0.79	0.02	0.71
LSTM-AE 128	✗	✗	**0.90**	0.01	0.76
1DConv-AE 64	✓	✓	0.78	0.02	0.73
1DConv-AE 64	✓	✗	0.87	0.01	0.75
1DConv-AE 64	✗	✓	0.78	0.01	0.68
1DConv-AE 64	✗	✗	**0.90**	0.00	0.77

Table 5.6: Results for LSTM-AE and 1DConv-AE models on the KTH dataset. $\beta = 4$

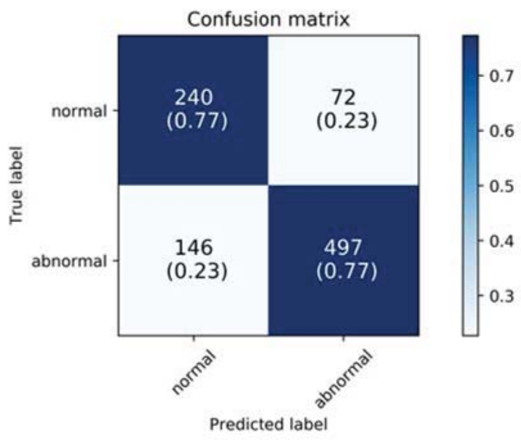
These results show that removing low scoring body keypoints (as outlined in Section 4.5.1.3) does not necessarily contribute to the model's classification abilities. Moreover, despite the fact that the window overlapping technique doubles the size of the training data, this did not prove to be very advantageous to this problem. The PR curves and confusion matrices for the best two models (marked in bold) are shown in Figure 5.9. Both the PR curves demonstrate that at various thresholds, the models are able to correctly classify normal and abnormal sequences of data. Both models were trained on (1247, 20, 34) samples and evaluated on (312, 20, 34) negative (normal) samples and (643, 20, 34) positive (abnormal) samples. In this case, since all 'running' actions were considered, the dataset contained more positive samples.



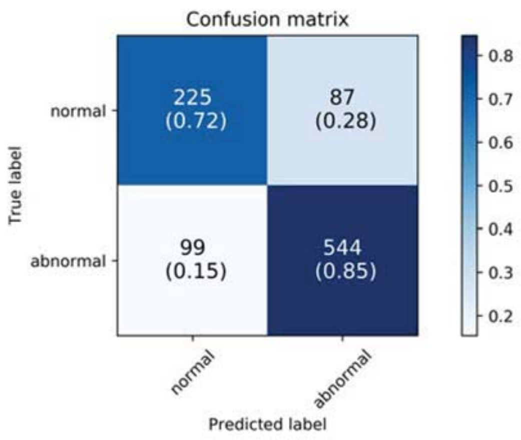
(a) PR Curve for LSTM-AE model



(b) PR Curve for 1DConv-AE



(c) Confusion Matrix for LSTM-AE



(d) Confusion Matrix for 1DConv-AE

Figure 5.9: KTH - PR Curves and Confusion Matrices for LSTM-AE and Conv1D-AE models

The key finding of this preliminary analysis is that both the LSTM-AE and 1DConv-AE models are able to learn to reconstruct the normal activities only for these to then be used for detecting anomalous sequences. Taking into account that both walking and running actions are very similar to each other in terms of body keypoints representation, the outcome of such approach is very promising. This also shows that the proposed model architectures are able to learn a compressed representation of the normal activities in a time-series fashion and use such output to 'transform' the model into a classifier (by using the reconstruction error as an anomaly score).

5.3.5 Analysis on a realistic and challenging dataset: ADL

The use of a realistic dataset is crucial for this study because it shows how the model reacts to real-life situations. In a real-world environment such as a nursing home, individuals are usually unaware of the camera location. Room objects such as chairs, tables and other furniture are typically in the scene. Such objects make detection and classification of human beings more challenging due to the possibility of occlusions. For example, a person sitting at a table might have his/her lower body occluded by the table itself. It is therefore desirable to have a dataset that simulates the real-world cases.

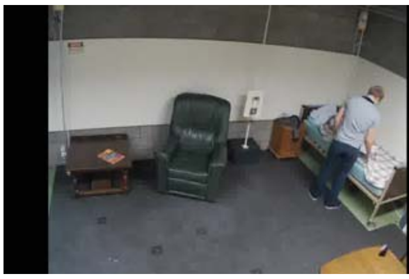
5.3.5.1 ADL Dataset

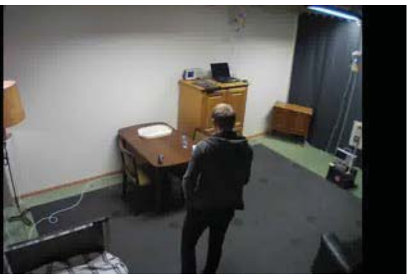
Activities of daily living (ADL) is a term that refers to a set of activities that are performed on a daily basis by an individual, usually alone. Such activities range from walking, sitting down, changing clothes, eating and more. Baldewijns et al. [5] present a dataset consisting of various subjects performing different activities in a room. This dataset bridges the gap between currently available datasets (which consist of simulated activities) and real-life datasets as it contains more realistic abnormal activities such as falls. The authors created such abnormal activities by re-enacting real-life incidents that were discovered in other studies. The authors claim that their dataset is more realistic as it contains a lot of 'non-fall' data which simulates the real-world; because typically, fall activities do not happen very often. Furthermore, the dataset contains a good number of different types of falls, was recorded in a realistic setting and contains a good balance between normal and abnormal activity data.

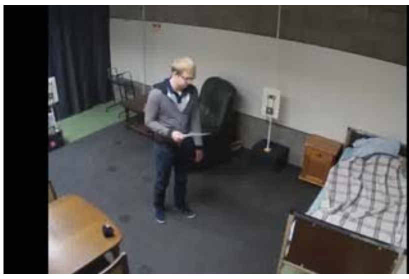
The dataset, which is publicly available[1], consists of a set of videos recorded in a realistic room from different angles. The videos are divided into two main classes: normal and abnormal activities. The normal activities consist of a human subject performing actions such as walking, sitting, eating, drinking, sleeping, changing clothes, reading and more. On the other hand, the abnormal activities contain various fall scenarios such as slow and fast falls and falls with different starting and ending poses. The room in which the footage was recorded was furnished to look similar to a nursing home room. In total, 5 web-cameras capturing 30 frames per second (fps) with a resolution of 800 by 480 were installed. In the paper it was reported that the cameras capture 12 fps with a resolution of 640 by 480, however, the footage reports otherwise. These were positioned in different locations within the room to capture various view-points. Figure 5.11 depicts

[1]`https://iiw.kuleuven.be/onderzoek/advise/datasets`

(a) Cam1


(b) Cam2


(c) Cam3


(d) Cam4

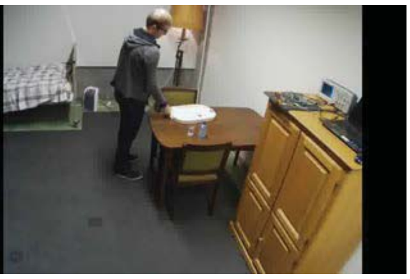

(e) Cam5

Figure 5.10: All camera positions for ADL Scenario 2

the room layout as well as the camera positions. The entrance to the room is located right of the first camera (marked C1). A frame from each camera for ADL Scenario 2 (ADL2) is shown in Figure 5.10.

A good balance between normal and abnormal data was recorded. Table 5.7 tabulates a summary of the recorded footage. Each scenario in the normal set consists of various set of activities. Furthermore, some alterations in the normal activity footage were induced by using different props such as walking aids, changing the order in which the activities were performed and changing the pace of the person. In both classes, real-life challenges such as illumination changes, occlusions and falling partially or completely out of camera view were also included. Every scenario (both normal and abnormal)

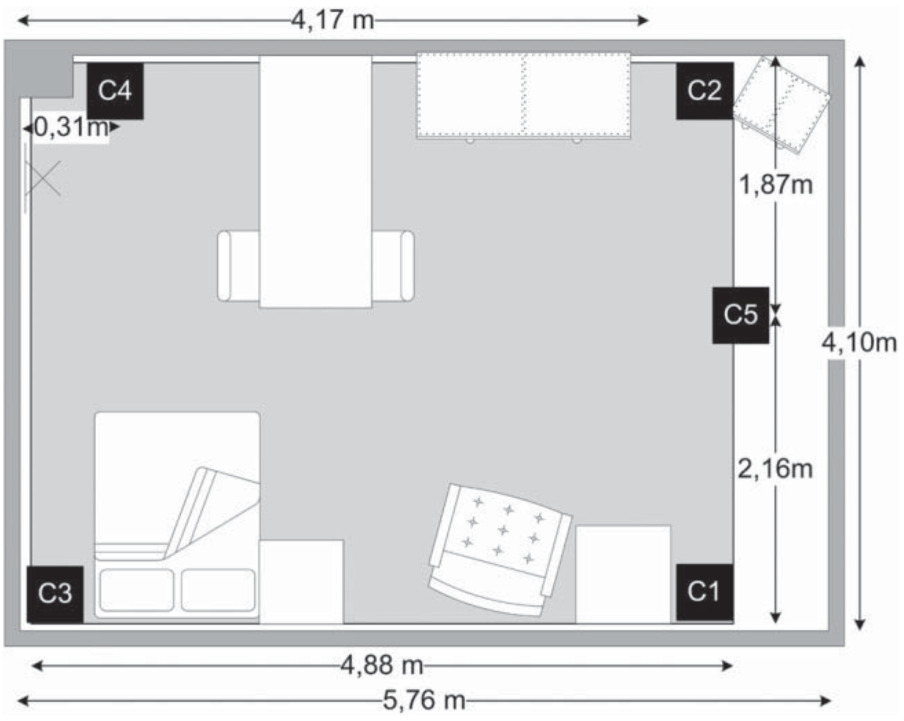


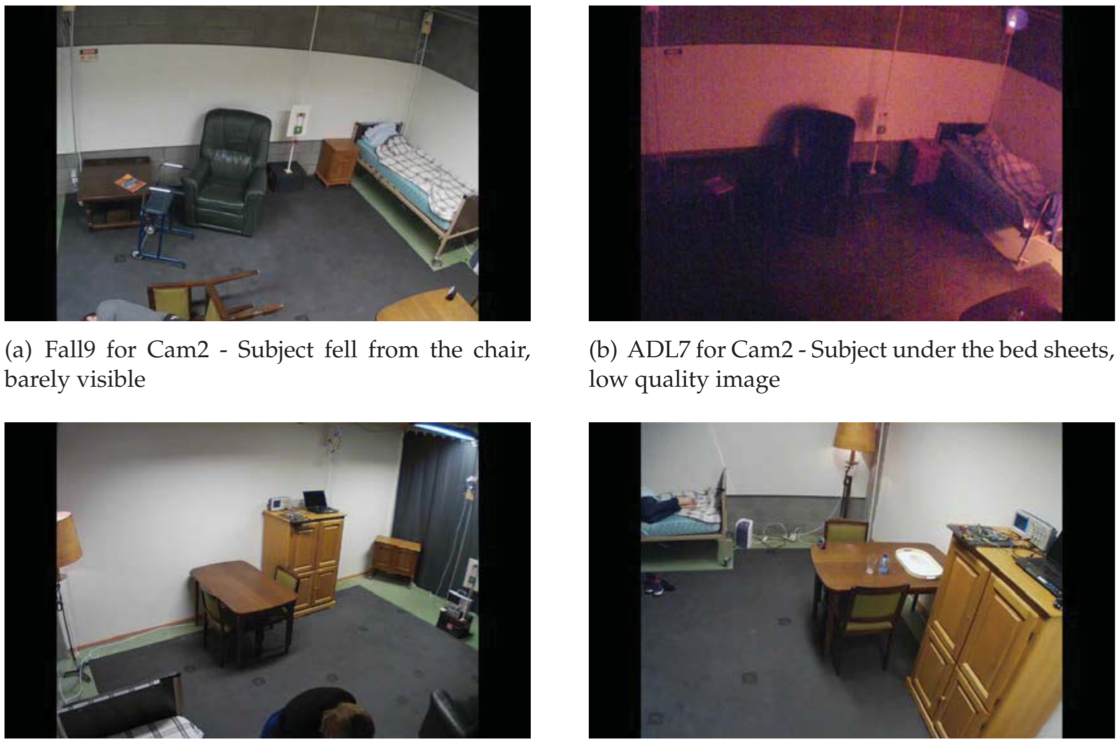
Figure 5.11: ADL Dataset: Room layout and camera positions (marked with C1 - C5)
Source: Baldewijns et al. [5]

Class	Number of Scenarios	Average length per scenario	Total length
Normal	17	20:39 mins	5:50 hours
Abnormal	55	2:44 mins	2:25 hours

Table 5.7: ADL Dataset Overview

was recorded with 5 different cameras. All video files were named in such a way that both the scenario and the camera can be identified. For example, the first fall scenario for camera one (C1) was named *Fall1_Cam1.avi* and the fifth normal scenario for C4 was named *ADL5_Cam4.avi*. A meta file was also provided with the dataset. This includes annotation details of the start and end time of the abnormal activity, together with the scenario description.

The nature of the ADL dataset is different from the KTH dataset outlined earlier due to a number of reasons. Firstly, the cameras installed in the ADL dataset are fixed to the ceiling and thus do not capture the human subject from a front angle. Such view makes the process for estimating the posture harder. Furthermore, in various camera angles (such as Cam1 and Cam2) the person might be in areas of the room where his/her body is only partially visible. This makes the pose estimation process more challenging as well. Another impediment in such dataset is the video quality, especially in low-light situations where the only source of light is infrared. Some examples of such challenges are depicted in Figure 5.12.


(a) Fall9 for Cam2 - Subject fell from the chair, barely visible

(b) ADL7 for Cam2 - Subject under the bed sheets, low quality image

(c) ADL9 for Cam3 - Subject bending to sit on a bed

(d) ADL2 for Cam5 - The only visible body parts are the subject's legs on a bed

Figure 5.12: ADL Dataset challenges

5.3.5.2 Dataset Statistics

All the footage from this dataset was pre-processed as described in Sections 4.3 and 4.4 in that all the frames were extracted from each video and then processed with PoseNet and OpenPose pose estimation algorithms. A list of frames extracted and processed for this dataset is tabulated in Table 5.8. Each camera folder includes all 17 scenarios (for normal) and all 55 scenarios (for abnormal). The footage for abnormal scenario 1 (Fall1) and normal scenario 3 (ADL3) contained more than one person. As outlined in Chapter 1, the focus of this study is on individual-human based activities and therefore these scenarios were omitted from the datasets. Since the abnormal data was annotated by the

Dataset	Scenario	# Frames
ADL Normal	Camera 1	660,565
ADL Normal	Camera 2	659,723
ADL Normal	Camera 3	657,707
ADL Normal	Camera 4	658,904
ADL Normal	Camera 5	659,711
ADL Abnormal*	Camera 1	13,680
ADL Abnormal*	Camera 2	13,680
ADL Abnormal*	Camera 3	13,650
ADL Abnormal*	Camera 4	13,920
ADL Abnormal*	Camera 5	13,920
Total # frames extracted		**3,365,460**

Table 5.8: Total number of frames extracted for the ADL dataset. * specific abnormal activity frames only

authors of the dataset themselves, only the frames where the abnormal activity occurs were extracted. Such data was then pre-processed as outlined in Section 4.5.1, and used to test the models. The tensor shape generated after the pre-processing steps follows the same format as described earlier in this section: *number of samples (S) x window length (L) x number of features (F)*.

Confidence scores histograms reported by both PoseNet and OpenPose for Camera 1 are depicted in Figure 5.13. Such histograms show the confidence score distribution for all the frames that scored in the range 0.1 - 1. The rest of the scores (0 - 0.09) were not included because, as outlined in Section 4.5.1, they consist of frames with no human subject and thus not informative to this problem. The rest of the histograms for cameras 2-5 for both algorithms can be found in Section A.2, in the Appendix. These histograms demonstrate that on average, the PoseNet algorithm managed to predict more frames with higher confidence scores than OpenPose. The median values for PoseNet and OpenPose for the normal data are 0.72 and 0.44 respectively. On the other hand, the median values for the abnormal frames are 0.53 and 0.44. Such difference is due to the fact that PoseNet is also able to estimate points that fall outside the frame. On the other hand, OpenPose only estimates the keypoints that are visible in the frame, setting the rest of the keypoints as 0. This difference explains why OpenPose reports a lower average score.

As underlined above, abnormal sequences consist of various types of falls. In some of the falls, a person might be fully visible in the frame at any given time, whilst a second

Algorithm	MCS Partial and Full Visibility	MCS Full Visibility
PoseNet	0.4	0.5
OpenPose	0.1	0.4

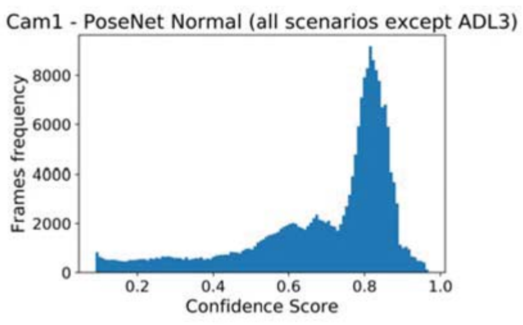
Table 5.9: Minimum confidence scores (MCS) that best captures partial and full visibility

later he/she might completely or partially be out of frame. After exploring the PoseNet and OpenPose data at various configurations for all cameras, it was determined that different confidence scores can be used to report partial and full visibility of human beings. Table 5.9 tabulates such confidence scores for both algorithms. At the minimum confidence scores (MCS) shown in the table, it was observed that 98% of the abnormal data is fully represented. The 'MCS Partial and Full Visibility' column in the table outlines the minimum confidence score required for each algorithm to capture both partial and full visibility of a human being. On the other hand, the other column shows the MCS required to capture full body visibility only. The only excluded sequences (2%) include those that only have 2 body parts or so visible in the frame as depicted in Figure 5.12. The rest of the frames with lower confidence scores were not used in the training set as due to heavy occlusions or limited visibility, the pose estimation model was unable to correctly estimate the body keypoints.

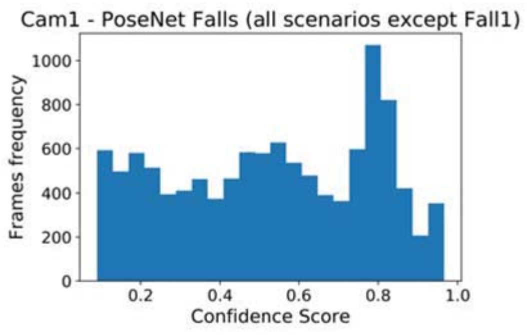
5.3.5.3 Survey

A survey[2] was also conducted in order to back such claims. 13 images from different cameras (both normal and abnormal) were randomly selected and processed with PoseNet. The reason for choosing PoseNet was because as demonstrated in Section 5.2, the algorithm performed better than OpenPose. In the survey, each processed image consisted of the original image itself together with the estimated body keypoints mapped onto the body. The participants were asked to inspect each image and mark the appropriate numeric response, ranging from 1-5, that best described the output. The confidence score originally output by the algorithm was also included in every image, where the main question was: 'How accurate was the pose estimation algorithm in its output?'. A value of 1 would describe the output as 'Not accurate' whereas a value of 5 would signify a 'Very accurate' output.

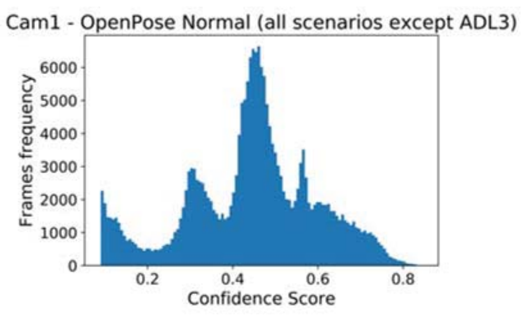
[2]`https://goo.gl/forms/4iXE1kaJAKGUUGKz2`



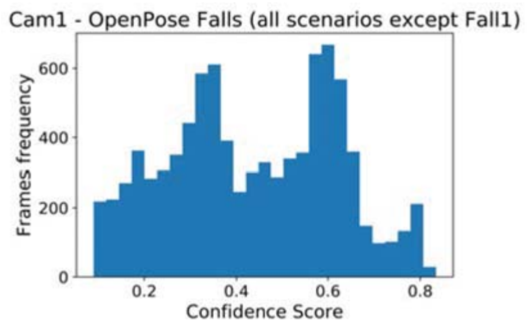
(a) PoseNet: Camera 1 - Normal



(b) PoseNet: Camera 1 - Abnormal



(c) OpenPose: Camera 1 - Normal



(d) OpenPose: Camera 1 - Abnormal

Figure 5.13: ADL - PoseNet and OpenPose confidence score histograms for Camera 1

The survey, which was created with Google Forms[3], was answered by 69 anonymous participants. Table 5.10 summarises the images used in the survey together with the participants' average scores. In general, it was noted that the average score for all questions was 3.90 which shows that overall, participants believe that the output from such algorithm on the ADL dataset is accurate. Specifically, in instances where the algorithm reports confidence scores $> 50\%$, the average score from participants was that of 4.09 (accurate to very accurate). In other cases where the person is heavily occluded (such as images 6, 7, 10, 11 and 13), the confidence scores were between 8% and 43%. For such scores, the participants, scored an average of 3.76. The lowest two scores reported by the participants were for images 10 and 11 (3.03 and 3.00 respectively). In both instances, the persons were barely visible in the frame and thus this explains the low confidence scores reported by the algorithm.

[3]`https://forms.google.com`

Image #	Description	Confidence Score	Participants' Average Score
Image 1	Person sitting on a sofa. All points were predicted.	96%	4.75
Image 2	No person in the scene.	5%	4.64
Image 3	Person walking using walking aid. Legs slightly occluded.	89%	4.43
Image 4	Person sitting on a bed. All points were predicted.	92%	4.19
Image 5	Person lying on the floor. Shoulder and right elbow keypoints not predicted.	68%	3.87
Image 6	Person partially visible. Visible body points were predicted	43%	4.40
Image 7	Person partially visible. Visible body points were predicted but left shoulder not predicted well.	42%	3.78
Image 8	Person sitting on a wheelchair. Slightly occluded. Majority of the keypoints were predicted.	83%	4.17
Image 9	Person lying on the floor with head slightly occluded. Legs not properly predicted.	72%	3.59
Image 10	Person heavily occluded with legs only visible. Visible body points not properly predicted.	16%	3.03
Image 11	A person's hand can be seen in the bottom right hand corner. No points predicted.	8%	3.00
Image 12	Person lying on the floor with the majority of the keypoints predicted except his right hands and legs, which were occluded.	56%	3.59
Image 13	Person partially occluded with a chair, lying on the floor. Some keypoints were not predicted.	41%	3.15

Table 5.10: List of images used in the survey together with the participants average score. Number of participants: 69.

5.3.5.4 Training the models

In light of the above, it was decided to train the algorithm with different data configurations. This was done to determine how the model reacts to (1) instances where the person is partially occluded and (2) when the person is fully visible in the frame (i.e. excluding partial visibility). The WL parameter values were manually checked to ensure that at least, part of the abnormal activity is captured. The list of data configuration parameters, which is tabulated in Table 5.11, shows a total of 40 combinations that were used to train the models. Each combination was used to train both the LSTM-AE and 1DConv-AE models for both PoseNet and OpenPose data representations.

Parameter	Values
Minimum Confidence Score (MCS)	{0.4, 0.5} (PoseNet) {0.1, 0.4} (OpenPose)
Remove Low Scores (RLS)	{True, False}
Window Length (WL)	{1s, 2s, 3s}
Downsample (DS)	{True, False}
Window 50% Overlap (WO)	{True, False}
Total	**40 combinations***

Table 5.11: List of parameter combinations used to train both models. * When window length is equal to 1s, the downsample value is automatically set to False.

> "Cross-validation is not a silver bullet. However, it is the best tool available, because it is the only non-parametric method to test for model generalization." - Varoquaux [89]

The training and test sets were prepared as described in the introduction of this section (Section 5.3.3). 5-fold CV was also used to test for model generalisation. Since as already outlined a semi-supervised approach is employed, the training data consisted of 'normal' ADL instances only. The test set comprised of all the 'abnormal' ADL sequences and 20% of the 'normal' instances. Initially, the data was randomly shuffled by the first axis, i.e. by the number of samples. This was done to ensure that the temporal order of the data is preserved. Furthermore, the random shuffle allowed for the data used to train and test the model to represent the overall distribution of the data [26]. Similar to the approach undertaken in the preliminary analysis, the models were trained to reconstruct sequences (i.e. multiple frames).

Similar to the work by Debard et al. [19], a separate model for each camera was trained. Both the LSTM-AE and the 1DConv-AE models were trained to reconstruct

Model	Algorithm	# Data configurations	# Cameras	# folds (CV)	# Models trained
LSTM-AE	Posenet	40	5	5	1000
1DConv-AE	Posenet	40	5	5	1000
LSTM-AE	OpenPose	40	5	5	1000
1DConv-AE	OpenPose	40	5	5	1000
Total # models trained					**4000**

Table 5.12: Total # models trained

the normal activities as described in Section 4.5.2. The models were trained for all 40 data configurations outlined in 4.5.1.7 with both skeleton representations extracted from PoseNet and OpenPose algorithms respectively. This was done to determine the ideal data configuration. The next step was to conduct the experiments to test all data configurations. A summary of such experiments is shown in Table 5.12, showing a total of 4,000 models.

The majority of the models were able to learn a compressed data representation after ≈ 35 epochs and both models seem to have generalised well on the validation set. Figure 5.14 plots the training loss of both the training and the validation sets over different number of epochs. This shows that this specific model managed to generalise well on the training data.

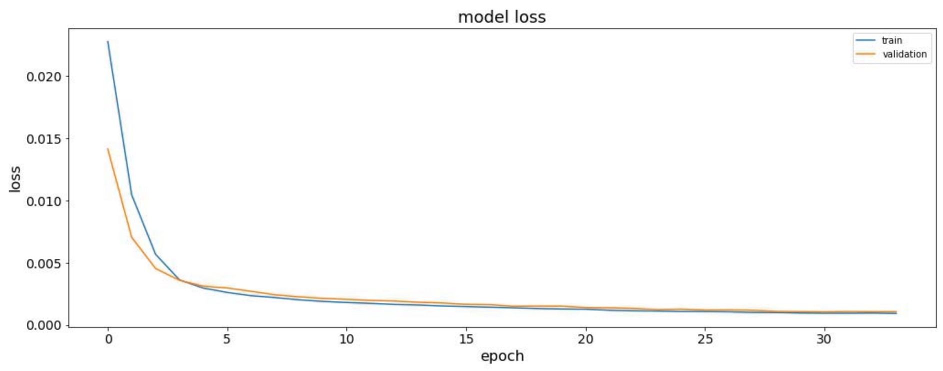


Figure 5.14: Train and Validation loss for LSTM-AE model

5.3.5.5 Results and their interpretation

In this section, the results for the experiments outlined above are presented and discussed. This section is divided into two main parts. The first part presents the results of the models trained on data that was extracted by PoseNet whereas the second part presents the findings on the OpenPose data representation. Specifically, Table 5.13 tabulates the top results for PoseNet and Table 5.18 (found in the next subsection) presents the top results for OpenPose. A summary is then presented in Section 5.3.5.6.

PoseNet data representation

Model	**WO**	**RLS**	**DS**	**WL**	**MCS**	**Mean PR AUC (all cams)**	**Mean F4 Score (all cams)**
LSTM-AE 128	✓	✗	✗	1 second	0.5	0.86 (0.10)	0.97 (0.02)
LSTM-AE 128	✓	✗	✗	3 seconds	0.5	0.85 (0.14)	0.97 (0.02)
LSTM-AE 128	✓	✗	✓	3 seconds	0.5	0.85 (0.10)	0.98 (0.01)
LSTM-AE 128	✓	✗	✗	2 seconds	0.5	0.85 (0.10)	0.96 (0.02)
LSTM-AE 128	✗	✗	✓	2 seconds	0.4	0.77 (0.15)	0.94 (0.03)
LSTM-AE 128	✓	✗	✓	2 seconds	0.4	0.76 (0.12)	0.94 (0.02)
LSTM-AE 128	✓	✗	✗	2 seconds	0.4	0.76 (0.11)	0.94 (0.02)
LSTM-AE 128	✓	✗	✗	3 seconds	0.4	0.75 (0.16)	0.94 (0.03)
1DConv-AE 64	✗	✗	✗	**3 seconds**	**0.5**	**0.91 (0.10)**	**0.98 (0.02)**
1DConv-AE 64	✓	✗	✗	3 seconds	0.5	0.90 (0.13)	0.98 (0.03)
1DConv-AE 64	✗	✗	✓	3 seconds	0.5	0.90 (0.07)	0.98 (0.01)
1DConv-AE 64	✓	✗	✗	1 second	0.5	0.87 (0.12)	0.96 (0.03)
1DConv-AE 64	✗	✗	✗	**3 seconds**	**0.4**	**0.85 (0.15)**	**0.95 (0.03)**
1DConv-AE 64	✓	✗	✗	3 seconds	0.4	0.83 (0.16)	0.96 (0.03)
1DConv-AE 64	✗	✗	✓	3 seconds	0.4	0.81 (0.16)	0.95 (0.03)
1DConv-AE 64	✗	✗	✗	2 seconds	0.4	0.80 (0.17)	0.94 (0.04)

Table 5.13: Top 4 results for the first two rows for every MCS (PoseNet) in the data configuration table.

Table 5.13 summarises the top 4 results for both models that were trained on the PoseNet data representation. Both scores in each row represent the average score for all camera models (that were each trained on 5 different folds). The PR AUC provides the overall predictive power of the classifier whilst the F4 score indicates the predictive skill for the best anomaly score threshold. The F4 score was calculated after finding the optimal threshold that best segregates normal from abnormal errors, as described in

Section 5.3.2. The values shown in brackets represent the standard deviation measure.

This table shows the parameters that best contribute to the skill of the model. With reference to the LSTM-AE model, results indicate that the WO parameter definitely helped to improve the reconstruction ability of the model. However, the 1DConv-AE model managed to perform quite well even without window overlapping. On the contrary, similar to the results seen in the preliminary analysis, removing low scoring (RLS) body keypoints did not prove to be effective to this problem. Furthemore, downsampling the data does not seem to effect the results significantly. However, the models performed better when the DS parameter was set to false.

The LSTM-AE model seemed to have performed best when the WL is set to 1 second (i.e. 30 frames) for both MCS values of 0.5 and 0.4. On the other hand, the 1DConv-AE model performed best when the WL is set to 3 seconds. A higher WL reduces the sample size but increases the number of time-steps. It was observed that at different WLs, both models produced promising results as the difference in the results was marginal. At a higher MCS (0.5), both models performed better. This can be explained by the fact that the data used to train and test the models included higher scoring keypoints with less noise. Overall, the performance of both models is similar for the same data configurations. However, the 1DConv-AE seems to outperform the LSTM-AE in both MCS values when WL = 3 seconds.

As outlined, the WL parameter plays an important role in such experiments as a higher value increases the time-step dimension of the data but decreases the sample size. Table 5.14 outlines the training and testing sizes for various data configurations for Camera 4 after data pre-processing. In this table, the RLS and DS parameters were not reported because they do not contribute to the sample size. For generating this table, both parameters were set to *false*. Overall, after data pre-processing, all cameras contained the same amount of data however, cameras 2 and 4 contained $\approx 15\%$ more data than the others. Such difference is due to the fact that both cameras were positioned in an area that captured more human activity data.

A breakdown for each camera model is given in Table 5.15. Such table tabulates the average PR AUC for every camera model (for all 5 folds) together with the standard deviation measure (shown in brackets). In this table, only the top performing model for each MCS is shown. It is noticeable that on average, all models performed quite well on cameras 1, 2 and 3. However, lower scores were reported by models trained on

Data Conf	Training Data (Normal)	Testing Data (Normal)	Testing Data (Abnormal)
WO: 1, MCS: 0.5, WL: 1	(14775, 30, 34)	(3694, 30, 34)	(248, 30, 34)
WO: 1, MCS: 0.4, WL: 1	(17212, 30, 34)	(4303, 30, 34)	(316, 30, 34)
WO: 1, MSC: 0.5, WL: 2	(7255, 60, 34)	(1814, 60, 34)	(138, 60, 34)
WO: 1, MSC: 0.4, WL: 2	(8423, 60, 34)	(2106, 60, 34)	(177, 60, 34)
WO: 1, MSC: 0.5, WL: 3	(4703, 90, 34)	(1176, 90, 34)	(62, 90, 34)
WO: 1, MSC: 0.4, WL: 3	(5440, 90, 34)	(1360, 90, 34)	(88, 90, 34)
WO: 0, MCS: 0.5, WL: 1	(7520, 30, 34)	(1880, 30, 34)	(148, 30, 34)
WO: 0, MCS: 0.4, WL: 1	(8788, 30, 34)	(2198, 30, 34)	(186, 30, 34)
WO: 0, MSC: 0.5, WL: 2	(3684, 60, 34)	(921, 60, 34)	(83, 60, 34)
WO: 0, MSC: 0.4, WL: 2	(4288, 60, 34)	(1072, 60, 34)	(102, 60, 34)
WO: 0, MSC: 0.5, WL: 3	(2406, 90, 34)	(602, 90, 34)	(47, 90, 34)
WO: 0, MSC: 0.4, WL: 3	(2799, 90, 34)	(700, 90, 34)	(64, 90, 34)

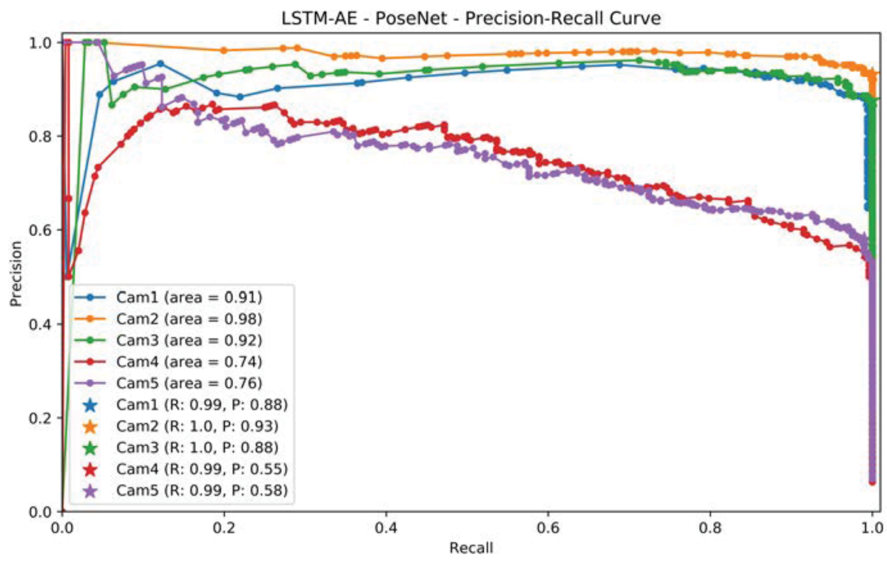
Table 5.14: Training and testing sample sizes with different parameters for Cam4 (PoseNet)

Model	Cam1	Cam2	Cam3	Cam4	Cam5
LSTM-AE 128 (MCS 0.5)	0.93 (0.01)	0.97 (0.01)	0.91 (0.02)	0.72 (0.05)	0.77 (0.01)
LSTM-AE 128 (MCS 0.4)	0.85 (0.05)	0.82 (0.02)	0.90 (0.01)	0.48 (0.02)	0.76 (0.03)
1DConv-AE 64 (MCS 0.5)	0.99 (0.02)	1.00 (0.00)	0.99 (0.01)	0.78 (0.06)	0.88 (0.03)
1DConv-AE 64 (MCS 0.4)	0.95 (0.03)	0.96 (0.01)	0.98 (0.01)	0.60 (0.05)	0.75 (0.03)

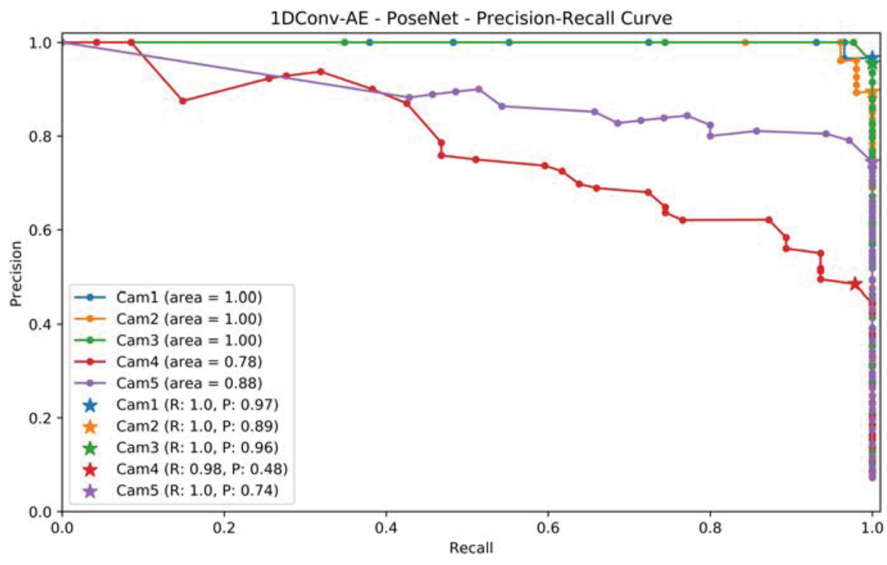
Table 5.15: Camera average PR AUC scores and standard deviation values for the top performing models (PoseNet)

cameras 4 and 5. The model for Cam4 also reports a higher standard deviation value, when compared to the others. This shows that the model performed slightly different with different data folds.

The PR curves and confusion matrices for the best LSTM-AE and 1DConv-AE models for MCS = 0.5 are shown in Figures 5.15 and 5.16 respectively. The PR curves demonstrate that at various thresholds, the model is able to correctly classify normal and abnormal sequences of data. The PR curve also shows the precision and recall scores for each



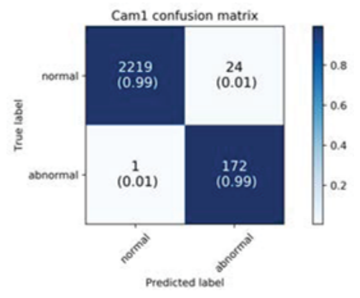
(a) LSTM-AE



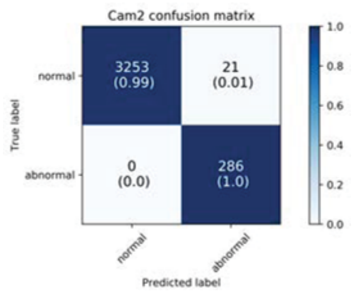
(b) 1DConv-AE

Figure 5.15: PR Curves for LSTM-AE and 1DConv-AE for top performing models (MCS: 0.5 - PoseNet)

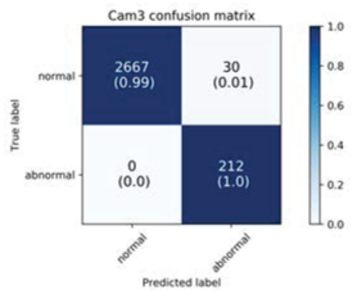
camera for the maximum F4 score recorded. These are marked with a 'star' marker and shown in the legend. The confusion matrices present a breakdown of the instances that were correctly and incorrectly predicted. On average, the model was able to correctly predict 99% of the abnormal instances and 97% of the normal instances from all cameras.



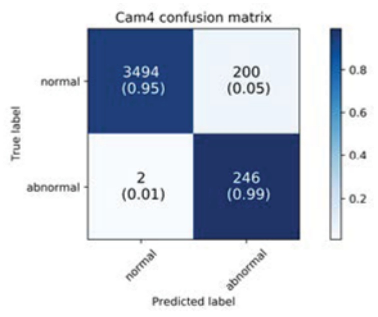
(a) Cam1



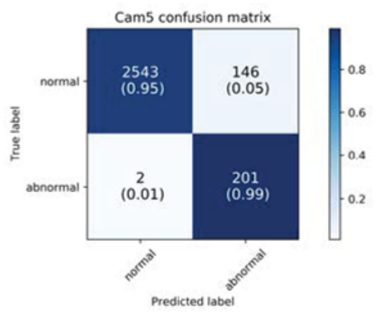
(b) Cam2



(c) Cam3



(d) Cam4



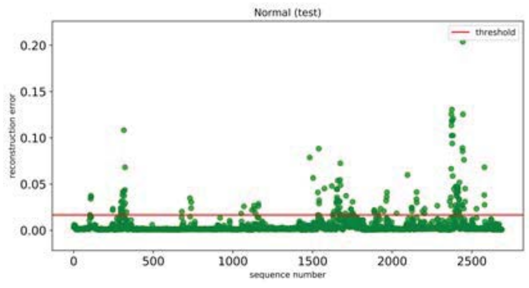
(e) Cam5

Figure 5.16: Confusion Matrices for LSTM-AE model. Conf.: WO: ✓, DS: ✗, WL: 1, MCS: 0.5 (PoseNet)

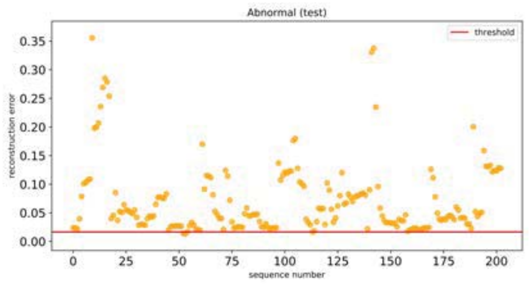
In cameras 4 and 5, 5% of the normal activities were incorrectly classified as abnormal. A visualisation of the normal and abnormal points together with the set threshold for Cam5 is depicted in Figure 5.17. These results demonstrate that both AutoEncoders were able to correctly learn how to reconstruct the normal activities. Furthermore, the technique of using the reconstruction errors to classify data sequences seemed to be very fruitful for this problem and shows that a semi-supervised approach fits well to this problem.

Comparing the models

It was also observed that on average, the 1DConv-AE model outperforms the LSTM-AE model at certain data configurations. However, one should note that in Tables 5.13 and 5.15 the top performing models are not using the same data configurations and thus cannot be directly compared. In light of this, a table which compares the top performing models using the same data configuration is tabulated in Table 5.16. For completeness sake, the 2 seconds WL configuration was also added to the comparison. Results demonstrate that the LSTM-AE performs better than the 1DConv-AE when $WL = 1$



(a) Cam1



(b) Cam2

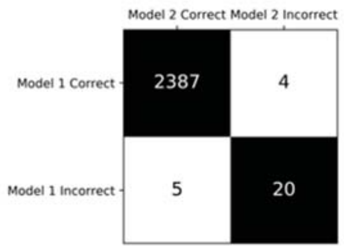
Figure 5.17: Normal and abnormal sequences visualisation for LSTM-AE. Conf.: WO: ✓, DS: ✗, WL: 1, MCS: 0.5 (PoseNet)

however, for other WL values especially when $WL = 3$, the 1DConv-AE outperforms the LSTM AutoEncoder model. Futhermore, since in a CNN weights are shared with other neurons (as outlined in Section 2.6), the network trains faster. In fact, on average, the 1DConv-AE model trains 70% faster than the LSTM-AE.

Model	**WO**	**DS**	**WL**	**MCS**	**Mean PR AUC (all cams)**	**Mean F4 Score (all cams)**
LSTM-AE 128	✓	✗	1 second	0.5	**0.86 (0.10)**	0.97 (0.02)
1DConv-AE 64	✓	✗	1 second	0.5	0.85 (0.13)	0.96 (0.03)
LSTM-AE 128	✓	✗	1 second	0.4	**0.74 (0.08)**	0.94 (0.02)
1DConv-AE 64	✓	✗	1 second	0.4	0.73 (0.14)	0.93 (0.03)
LSTM-AE 128	✗	✗	3 seconds	0.5	0.84 (0.13)	0.97 (0.02)
1DConv-AE 64	✗	✗	3 seconds	0.5	**0.91 (0.10)**	0.98 (0.02)
LSTM-AE 128	✗	✗	3 seconds	0.4	0.74 (0.17)	0.93 (0.03)
1DConv-AE 64	✗	✗	3 seconds	0.4	**0.85 (0.15)**	0.95 (0.03)
LSTM-AE 128	✓	✗	2 seconds	0.5	0.85 (0.10)	0.97 (0.02)
1DConv-AE 64	✓	✗	2 seconds	0.5	**0.87 (0.13)**	0.97 (0.03)
LSTM-AE 128	✓	✗	2 seconds	0.4	0.76 (0.11)	0.94 (0.02)
1DConv-AE 64	✓	✗	2 seconds	0.4	**0.79 (0.13)**	0.94 (0.03)

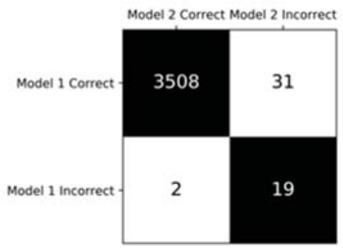
Table 5.16: Model comparison using the same data configurations (PoseNet)

Since in certain cases the difference in performance is marginal, the McNemar test was used to test whether such difference is statistically significant or not. Recommended by Dietterich [20], the McNemar test is a non-parametric statistical test used to compare the performance of two classifiers. The test uses a contingency table which contains 4 outcomes as shown in Figure 5.18. The outcomes include the number of predictions that each model got right or wrong when the other model got right or wrong. As an example, in Figure 5.18(a), the top left black cell shows the number of predictions that both Model 1 and Model 2 got right. On the other hand, the top right white cell shows the amount of predictions that Model 2 got incorrect that Model 1 got right. In this section, Model 1 refers to LSTM-AE whereas Model 2 refers to 1D-ConvAE. The contingency table can be used to pinpoint the best model. However, when the scores are very close, it becomes less conclusive about which model is the better and thus the test is used to determine if the difference is statistically significant or not [20].



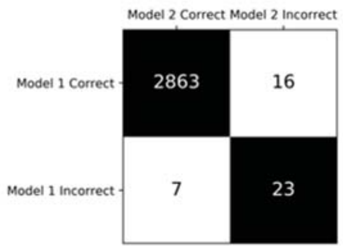
	Model 2 Correct	Model 2 Incorrect
Model 1 Correct	2387	4
Model 1 Incorrect	5	20

(a) Cam1 - p-value: 1.0



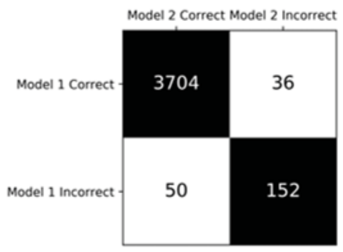
	Model 2 Correct	Model 2 Incorrect
Model 1 Correct	3508	31
Model 1 Incorrect	2	19

(b) Cam2 - p-value: 0.00



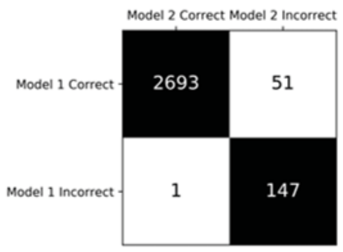
	Model 2 Correct	Model 2 Incorrect
Model 1 Correct	2863	16
Model 1 Incorrect	7	23

(c) Cam3 - p-value: 0.09



	Model 2 Correct	Model 2 Incorrect
Model 1 Correct	3704	36
Model 1 Incorrect	50	152

(d) Cam4 - p-value 0.16



	Model 2 Correct	Model 2 Incorrect
Model 1 Correct	2693	51
Model 1 Incorrect	1	147

(e) Cam5 - p-value 0.00

Figure 5.18: Contingency tables for for LSTM-AE (Model 1) and 1DConv-AE (Model 2). Conf.: WO: ✓, DS: ✗, WL: 1, MCS: 0.5 (PoseNet)

In the McNemar test, the null and alternate hypotheses are formulated as follows:

H_0 = The two models have the same error rate (i.e. no statistical difference)
H_A = The two models perform differently (i.e. different proportions of errors)

The test statistic only takes into consideration the discordant cells (white cells) from the contingency table. It outputs a statistic together with a p-value where a p-value $> \alpha$ would indicate that the null hypothesis cannot be rejected. On the other hand, if the p-value is smaller than α, the null hypothesis is rejected. For these tests, the $alpha$ threshold was set to be 0.05, which is a common value amongst researchers [20]. Table 5.17, which is linked to Table 5.16 shows the p-values for every camera for the data configurations compared in this section. The main aim of this test is to determine whether the mean PR AUC difference (for e.g. 86% vs 85%) between the models is statistically significant. Ultimately, the test is used to select the best model. The first row in the table shows that in cameras 1, 3 and 4 the p-value > 0.05. This means that the null hypothesis cannot be rejected and thus both models performed in a similar way. However, in cameras 2 and 5, the p-values are both < 0.05 which indicate that the models perform differently (and the null hypothesis has to be rejected). The contingency tables used in the first test (first row of the table) are illustrated in Figures 5.18(b) and (c) and shows that the LSTM-AE model (Model 1) is better than 1DConv-AE in cameras 2, 3 and 5. On the other hand, the 1DConv-AE model performed slightly better on Camera 4. This concludes that the LSTM-AE model is indeed better than the 1DConv-AE for this specific data configuration and the 1% difference is statistically significant (at least for Cam2 and Cam5). This conclusion can also be applied to the rest of the data configurations where it was observed that the models do indeed perform differently, especially for Cam3 and Cam4. An interesting observation is that both models seemed to have the same error rate in different data configurations for cameras 1, 2 and 5.

Configuration	**Cam1**	**Cam2**	**Cam3**	**Cam4**	**Cam5**
WO: ✓, DS: ✗, WL: 1, MCS: 0.5	1.00	0.00	0.09	0.16	0.00
WO: ✓, DS: ✗, WL: 1, MCS: 0.4	0.00	0.56	0.00	0.00	0.00
WO: ✗, DS: ✗, WL: 3, MCS: 0.5	0.25	1.00	0.06	0.00	1.00
WO: ✗, DS: ✗, WL: 3, MCS: 0.4	1.00	0.04	0.02	0.00	0.11
WO: ✓, DS: ✗, WL: 2, MCS: 0.5	0.11	0.07	0.00	0.01	0.38
WO: ✓, DS: ✗, WL: 2, MCS: 0.4	0.21	0.00	0.00	0.00	0.33

Table 5.17: P-values for every camera model reported by McNemar test (PoseNet)

The top performing models were also tested with different number of LSTM units (for LSTM-AE) and CNN filters (for 1DConv-AE) as outlined in Section 4.5.2. In general, however, it was noted that the results were not significantly different from the original hyper-parameters. The results are tabulated in the appendix in Table A.1.

OpenPose data representation

In this section, an interpretation of the results for OpenPose is given. Additionally, a comparison between both data representations (PoseNet and OpenPose) is presented. Table 5.18 summarises the top 4 results for both models that were trained on two different MCS values (0.4 and 0.1). Similar to the results table for PoseNet, both scores in each row represent the average score for all camera models. In this case, both models for both MCS values performed best with the same data configuration. In all top performing models, the WO parameter has shown to improve the reconstruction ability of the models. Furthermore, neither the RLS nor the DS parameters proved to be effective. Similar to the PoseNet representation results, these results indicate that the LSTM-AE model outperforms the 1DConv-AE in all configurations. On the contrary to PoseNet, both models seemed to have performed best when $WL = 1$. Interestingly, it looks like both models performed best when the MCS parameter was set to 0.1 (which includes partial visibility). This contrasts with the results obtained using PoseNet's representation and could be explained by the fact that OpenPose only outputs keypoints that are visible in the frame and thus, the data includes less noise.

Model	WO	RLS	DS	WL	MCS	Mean PR AUC (all cams)	Mean F4 Score (all cams)
LSTM-AE 128	✓	✗	✗	**1 second**	**0.4**	**0.79 (0.06)**	**0.94 (0.02)**
LSTM-AE 128	✓	✗	✓	2 seconds	0.4	0.74 (0.08)	0.91 (0.01)
LSTM-AE 128	✓	✗	✗	2 seconds	0.4	0.73 (0.08)	0.90 (0.02)
LSTM-AE 128	✗	✗	✓	2 seconds	0.4	0.73 (0.08)	0.90 (0.05)
LSTM-AE 128	✓	✗	✗	**1 second**	**0.1**	**0.87 (0.04)**	**0.92 (0.02)**
LSTM-AE 128	✗	✗	✗	1 second	0.1	0.80 (0.07)	0.89 (0.03)
LSTM-AE 128	✓	✗	✓	2 seconds	0.1	0.79 (0.09)	0.89 (0.03)
LSTM-AE 128	✓	✗	✗	2 seconds	0.1	0.79 (0.07)	0.89 (0.03)
1DConv-AE 64	✓	✗	✗	1 second	0.4	0.71 (0.08)	0.90 (0.02)
1DConv-AE 64	✓	✗	✗	2 seconds	0.4	0.69 (0.08)	0.87 (0.03)
1DConv-AE 64	✗	✗	✗	1 second	0.4	0.66 (0.09)	0.87 (0.03)
1DConv-AE 64	✓	✗	✓	2 seconds	0.4	0.66 (0.08)	0.86 (0.02)
1DConv-AE 64	✓	✗	✗	1 second	0.1	0.76 (0.09)	0.88 (0.03)
1DConv-AE 64	✓	✗	✗	2 seconds	0.1	0.71 (0.10)	0.86 (0.03)
1DConv-AE 64	✗	✗	✗	1 second	0.1	0.71 (0.09)	0.86 (0.03)
1DConv-AE 64	✓	✗	✓	2 seconds	0.1	0.70 (0.11)	0.85 (0.04)

Table 5.18: Top 4 results for the second two rows for every MCS (OpenPose) in the data configuration table.

Table 5.19, tabulates the training and testing sizes for various data configurations for Camera 4. It was also observed that the OpenPose data representation includes $\approx 38\%$

Data Conf	Training Data (Normal)	Testing Data (Normal)	Testing Data (Abnormal)
WO: 1, MCS: 0.4, WL: 1	(12840, 30, 50)	(3210, 30, 50)	(286, 30, 50)
WO: 1, MCS: 0.1, WL: 1	(23354, 30, 50)	(5839, 30, 50)	(445, 30, 50)
WO: 1, MSC: 0.4, WL: 2	(6282, 60, 50)	(1571, 60, 50)	(150, 60, 50)
WO: 1, MSC: 0.1, WL: 2	(11500, 60, 50)	(2875, 60, 50)	(246, 60, 50)
WO: 1, MSC: 0.4, WL: 3	(4052, 90, 50)	(1013, 90, 50)	(71, 90, 50)
WO: 1, MSC: 0.1, WL: 3	(7527, 90, 50)	(1882, 90, 50)	(131, 90, 50)
WO: 0, MCS: 0.4, WL: 1	(6557, 30, 50)	(1640, 30, 50)	(170, 30, 50)
WO: 0, MCS: 0.1, WL: 1	(11854, 30, 50)	(2964, 30, 50)	(249, 30, 50)
WO: 0, MSC: 0.4, WL: 2	(3195, 60, 50)	(799, 60, 50)	(88, 60, 50)
WO: 0, MSC: 0.1, WL: 2	(5815, 60, 50)	(1454, 60, 50)	(136, 60, 50)
WO: 0, MSC: 0.4, WL: 3	(2088, 90, 50)	(522, 90, 50)	(54, 90, 50)
WO: 0, MSC: 0.1, WL: 3	(3820, 90, 50)	(956, 90, 50)	(89, 90, 50)

Table 5.19: Training and testing sample sizes with different parameters for Cam4 (OpenPose)

more test samples than PoseNet's when MCS = 0.1. However, at higher confidence scores, PoseNet contains $\approx$ 15% more data. This shows that OpenPose as a pose estimation model managed to predict more frames when the human subject is partially visible. The same table for PoseNet was presented earlier in this chapter, marked 5.14. Similar to PoseNet, all cameras contained a similar amount of data but cameras 2 and 4 contained $\approx$ 15% more.

Since in Table 5.18 the mean scores are given, a breakdown for each camera model for the top performing models is given in Table 5.20. This table shows the average PR AUC for every camera model (for all 5 folds) together with the standard deviation (shown in brackets). It was observed that on average, the LSTM-AE model managed to perform quite well on all cameras but excelled in Cam5. As opposed to PoseNet, both models also managed to do quite well on cameras 4 and 5. This shows that the data representation for OpenPose is able to handle various camera angles better than PoseNet. On the other hand, the PR AUC scores achieved by the models trained on PoseNet's data representation were much higher than OpenPose's.

The PR curves and confusion matrices for the best models for MCS = 0.1 are shown in Figures 5.19 and 5.20 respectively. The PR curves demonstrate that at various thresholds, the models are able to classify normal and abnormal sequences of data. It also shows that the model for Cam3 performed worst out of all 5 cameras. On average, the

Model	Cam1	Cam2	Cam3	Cam4	Cam5
LSTM-AE 128 (MCS 0.4)	0.78 (0.04)	0.77 (0.03)	0.77 (0.06)	0.74 (0.04)	0.92 (0.02)
LSTM-AE 128 (MCS 0.1)	0.85 (0.02)	0.87 (0.02)	0.79 (0.03)	0.89 (0.02)	0.93 (0.01)
1DConv-AE 64 (MCS 0.4)	0.67 (0.06)	0.65 (0.02)	0.72 (0.03)	0.65 (0.02)	0.87 (0.02)
1DConv-AE 64 (MCS 0.1)	0.80 (0.02)	0.72 (0.04)	0.61 (0.03)	0.80 (0.03)	0.87 (0.02)

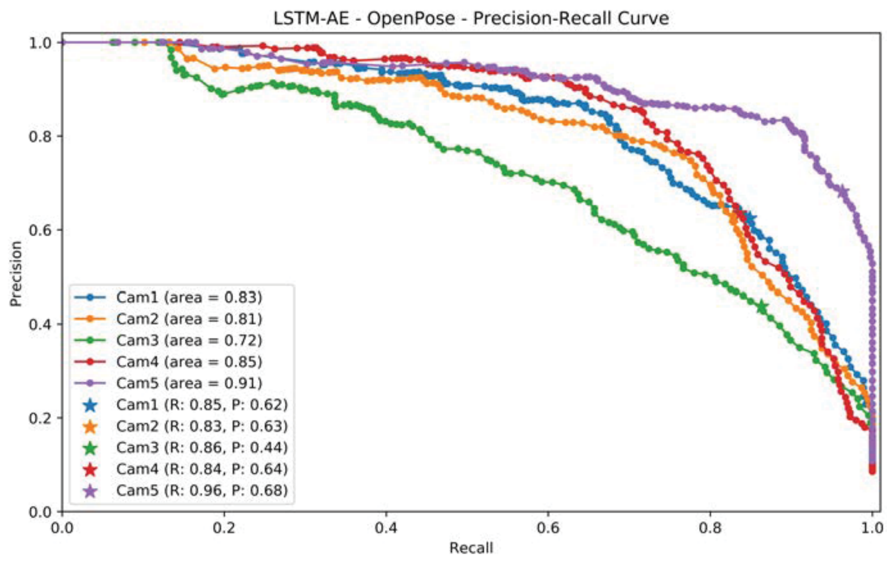
Table 5.20: Camera average PR AUC scores and standard deviation values for the top performing models (OpenPose)

LSTM-AE model outperformed the 1DConv-AE model in all data configurations. The graphs also show the precision and recall scores for each camera for the maximum F4 score recorded. These are marked with a 'star' marker and shown in the legend. The confusion matrices present a breakdown of the instances that were correctly and incorrectly predicted. On average, the LSTM-AE model was able to correctly predict 90% of the abnormal sequences and 91% of the normal sequences from all cameras.

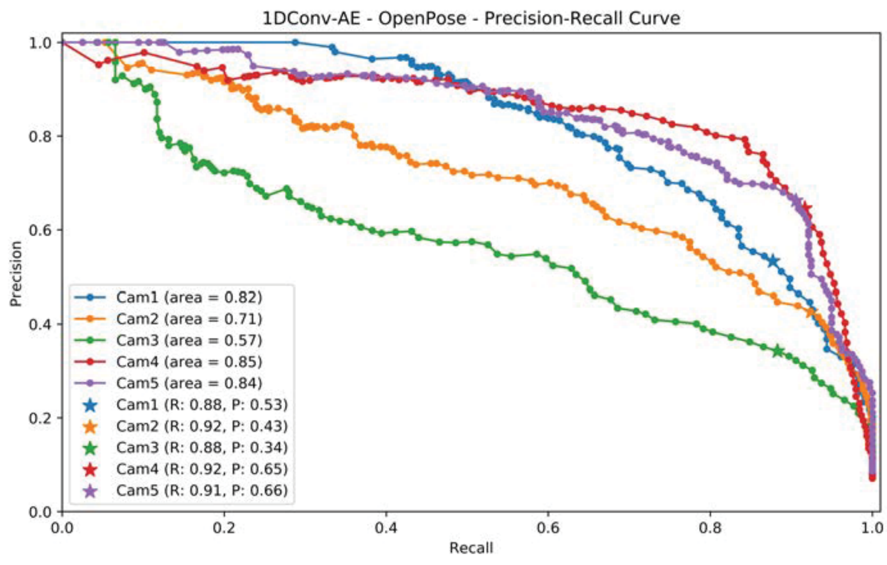
The top performing models were also tested with different number of LSTM units (for LSTM-AE) and CNN filters (for 1DConv-AE) as outlined in Section 4.5.2. In general, it was noted that the results were not significantly different for the LSTM-AE, which is the best performer on partial visibility MCS. However, improvements were observed for the 1DConv-AE model with different CNN filters. Such improvement however still did not outperform the LSTM-AE model. The results are tabulated in the appendix in Table A.2.

5.3.5.6 Summary

In this analysis, the ADL dataset was used to evaluate the proposed method on two data representations extracted by PoseNet and OpenPose. The LSTM-AE and 1DConv-AE models were trained to reconstruct the normal data at various data configurations. Specifically, one of the data parameters (MCS) was used to train the model on partial and full body visibility. In both data representations, both models produced promising results. Table 5.21 aims at providing a summary of the results that were presented earlier in this section.



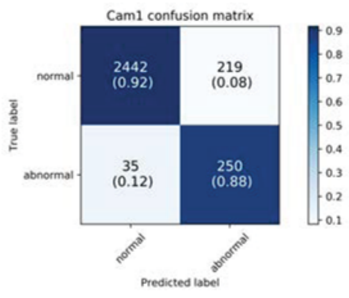
(a) LSTM-AE model



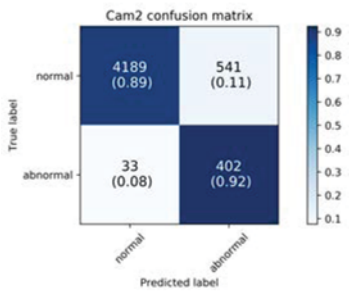
(b) 1DConv-AE model

Figure 5.19: PR Curves for LSTM-AE and 1DConv-AE models. Conf.: WO: ✓, DS: ✗, WL: 1, MCS: 0.1 (OpenPose)

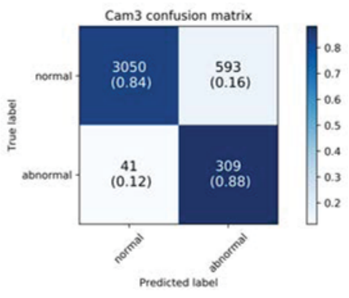
It was observed that the models trained on PoseNet data were able to handle full visibility better than OpenPose. On the other hand, the models trained on OpenPose data were able to handle partial visibility views much better. The LSTM-AE model per-



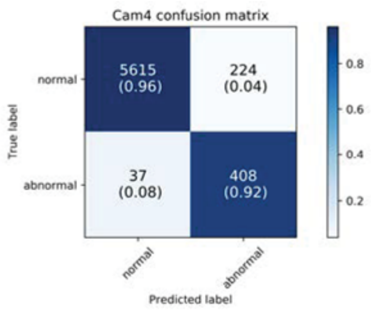
(a) Cam1



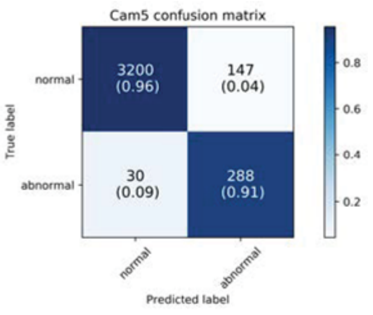
(b) Cam2



(c) Cam3



(d) Cam4



(e) Cam5

Figure 5.20: Confusion Matrices for LSTM-AE model. Conf.: WO: ✓, DS: ✗, WL: 1, MCS: 0.1 (OpenPose)

formed best when the WL was set to 1 second. On the contrary, the 1DConv-AE model was able to perform better when the WL was set to 3 seconds. In comparison with PoseNet's representation, the models trained on OpenPose's representation generated more false positives. This explains why the mean F4 score for PoseNet is higher.

Finally, these results demonstrate that both AutoEncoders were able to correctly learn how to reconstruct the normal activities. Furthermore, the technique of using the reconstruction errors to classify data sequences proved to be very fruitful for this problem and shows that a semi-supervised approach fits well to this problem. Additionally, the results also indicate that a semi-supervised model trained on solely body estimated keypoints is indeed able to detect anomalous sequences without using extra hardware such as depth sensors.

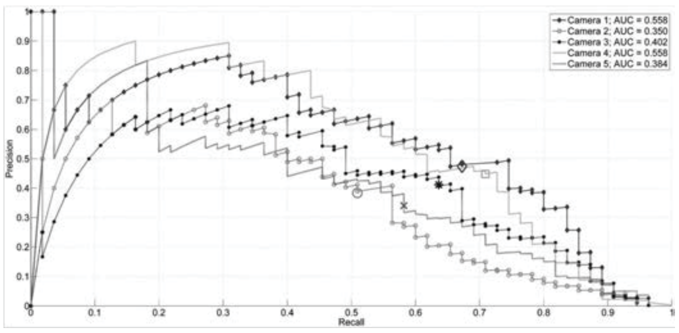
Data Representation	Model	Configuration	Mean PR AUC (all cams)	Mean F4 Score (all cams)
PoseNet	**1DConv-AE**	**Full visibility WL: 3 seconds**	**0.91 (0.10)**	**0.98 (0.02)**
OpenPose	1DConv-AE	Full visibility WL: 3 seconds	0.64 (0.12)	0.88 (0.03)
PoseNet	LSTM-AE	Partial visibility WL: 1 second	0.74 (0.08)	0.94 (0.02)
OpenPose	**LSTM-AE**	**Partial visibility WL: 1 second**	**0.87 (0.04)**	**0.92 (0.02)**

Table 5.21: Results summary for PoseNet and OpenPose data representations, for both MCS values

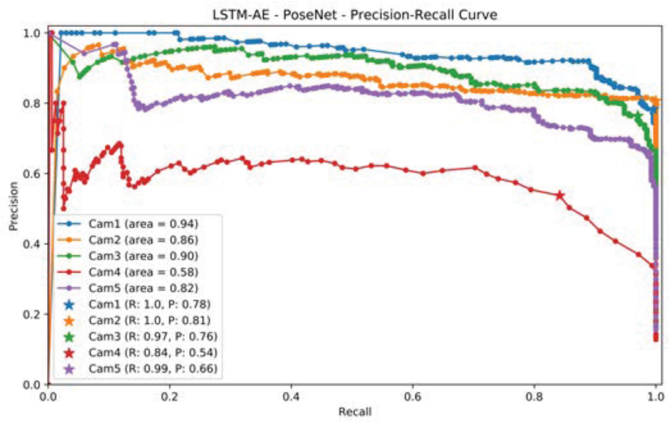
5.4 Evaluation against other work

The proposed method which was presented in Chapter 4 and evaluated in the previous section is compared with the models implemented by Debard et al. [19]. As outlined in Chapter 2, Section 3.7, the authors used background subtraction together with a particle filter to locate and track the person from the video footage. After detecting the human subject from the footage, they constructed a feature vector consisting of 5 elements and used them to train an SVM to classify events such as falls and non-falls. To evaluate their method, they used the same ADL dataset used in this study and trained their models using *k*-fold CV where *k* was set to 10.

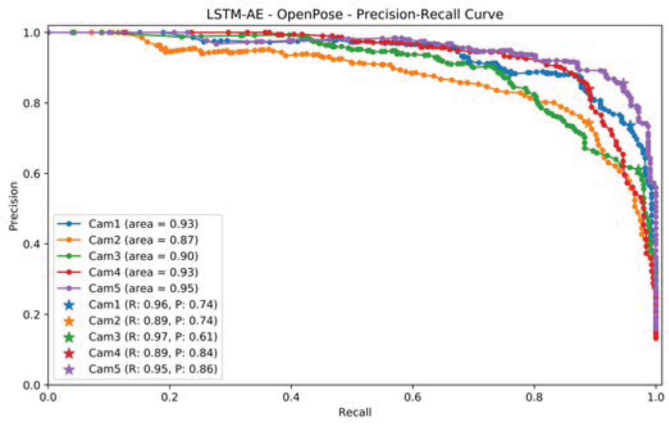
In view of this, the data configurations that achieved the best performance were used to re-train the models using 10-fold CV. Since in their approach the authors trained their models on all frames (i.e. partial and full visibility), only the MCS values that capture both partial and full visibility (0.1 for OpenPose and 0.4 for PoseNet) were used for comparison. To further ensure that the same testing environment is used, the same window length used by Debard et al. (1 second) is utilised. Furthermore, to reduce the false alarm rate, the authors removed single detections. Such approach was replicated as described in Section 4.5.1. In this comparison, the LSTM-AE model is used because, as outlined earlier in this chapter, it outperformed the 1DConv-AE model at such data configuration.



(a) Debard et al. [19]



(b) This study: LSTM-AE trained on PoseNet



(c) This study: LSTM-AE trained on OpenPose

Figure 5.21: PR Curves presented by Debard et al. [19] and this study

Figure 5.21 illustrates 3 PR curves. Figure (a) shows the results achieved by Debard et al. [19] that were presented in [5] whereas (b) and (c) represent the results carried

	Debard et al. [19]	LSTM-AE 128 PoseNet	LSTM-AE 128 OpenPose
Cam1	P: 0.47 R: 0.67, AUC: 0.56	**P: 0.78 R: 1.0 AUC: 0.94**	P: 0.74 R: 0.96 AUC: 0.93
Cam2	P: 0.38 R: 0.51, AUC: 0.35	**P: 0.81 R: 1.0 AUC: 0.86**	P: 0.74 R: 0.89 AUC: 0.87
Cam3	P: 0.41 R: 0.64, AUC: 0.40	**P: 0.76 R: 0.97 AUC: 0.90**	P: 0.61 R: 0.97 AUC: 0.90
Cam4	P: 0.45 R: 0.71, AUC: 0.56	P: 0.54 R: 0.84 AUC: 0.58	**P: 0.84 R: 0.89 AUC: 0.93**
Cam5	P: 0.34 R: 0.58, AUC: 0.38	P: 0.66 R: 0.99 AUC: 0.82	**P: 0.86 R: 0.95 AUC: 0.95**

Table 5.22: Comparison of Precision, Recall and AUC for all three methods. Higher values are better.

out in this study. From this quantitative evaluation it is evident that the method employed in this study outperforms the approach undertaken in [19]. The PR curves clearly demonstrate that the models trained in this study have a higher precision-recall ratio and thus the AUC scores are superior.

A summary of the precision (P), recall (R) and AUC for each camera and for all three methods is presented in Table 5.22. Both the precision and recall for all methods were selected after finding the maximum Fβ score (where $\beta = 4$) from the PR curve itself. Having slightly higher precision and recall values, the models trained on PoseNet seem to perform better for cameras 1, 2 and 3. However, in cameras 4 and 5, the models trained on OpenPose data outperformed the models trained on PoseNet significantly. It is evident that the employed LSTM-AE models were able to detect more normal and abnormal events than the work in [19]. It can also be observed that the LSTM-AE models for all cameras managed to score a considerably high recall score. This is vital because in anomaly detection systems an abnormal event that is not detected (false negative) is much riskier than an undetected false positive.

5.5 Conclusion

This chapter aimed to provide the reader with the techniques employed to evaluate the proposed method. The results together with their interpretation and key findings were presented and structured in 3 main sections.

In the first section, the results demonstrated that pose estimated data from RGB videos is indeed comparable to sensor data from depth cameras such as the Kinect for classifying human actions. This is because the accuracy scores were considerably high for both pose estimated and sensor data (88% and 92% respectively). Taking into consideration that no extra hardware was utilised, the results from this experiment are quite promising as they show that pose estimated data from a 2D image can be informative enough to describe a human action.

In the second section, the evaluation metrics used to evaluate the performance of the models to detect abnormal behaviour from video footage were presented. Furthermore, two different datasets were outlined and utilised to evaluate the effectiveness of the proposed method. The KTH dataset was used to conduct a preliminary analysis and the ADL dataset was utilised to determine whether the proposed method is capable of detecting abnormal behaviour in a realistic environment. In both datasets, it was determined that both AutoEncoder models (LSTM-AE and 1DConv-AE) were able to correctly learn how to reconstruct the normal activities. Furthermore, the technique of using the reconstruction errors to classify data sequences proved to be very fruitful for the resolution of this problem and showed that a semi-supervised approach fits well to this problem. Additionally, the results also indicated that a semi-supervised model trained solely on body estimated keypoints is indeed capable of classifying between normal and anomalous sequences as an average PR AUC of 0.86 was recorded.

In the final and third section, the results of this study were compared to similar work done on the same dataset. It was determined that the method employed in this study outperforms the approach undertaken in [19] with a significantly higher precision-recall rate.

6

Conclusion and Future Work

In this study, a method for detecting abnormal behaviour through video footage was presented. The main aim of this dissertation was to determine whether the human body keypoints extracted from RGB images can be used to train a model to detect abnormal activities.

In order to achieve this goal, a number of objectives were set up. The first objective involved extracting human body keypoints from a 2D image with the use of pre-trained pose estimation algorithms.

The extracted body keypoints were then used to:

- To train a number of supervised models in order to compare sensor data with estimated body keypoints from RGB images. This was presented in Chapter 4, Section 4.6.
- To train a number of semi-supervised models that should be able to detect abnormal human behaviour. This was presented in Chapter 4, Section 4.5.

The last objective was to measure the effectiveness of all the models, this was presented in Chapter 5, Sections 5.2 and 5.3.5. The rest of this chapter aims at providing the reader with a detailed description of how each objective was achieved.

In the first chapters, an overview of machine learning techniques employed in anomaly detection was given. This was followed by an in-depth background on recent techniques used in the field. In these chapters, the focus was on networks that are able to handle the temporal dimension of the data, such as RNNs. Furthermore, anomaly detection techniques employed in sequence-based data such as videos and time-series data were reviewed and compared. This includes techniques employed in detecting and recognising human activities as well as other techniques that use 3D convolutions and similar architectures to process the whole video frames. An overview of pose estimation algorithms and related work in the area was also presented.

In light of the principles that were explored and reviewed, the proposed solution was developed. Two pose estimation algorithms were used to extract body keypoints from each video frame. The data was organised to follow a time-series manner. Presented in Chapter 4, Section 4.4, two pose estimation algorithms were used to extract the human representation from video footage. As reported in Chapter 5, Section 5.3.5.2, the histograms show that both algorithms managed to estimate the majority of the keypoints even when the persons were partially visible or occluded by other objects. Such data was then pre-processed and used to train two types of AutoEncoder models (Section 4.5.2). The first model comprised of a number of LSTM layers, whereas the second model was designed with a number of 1D Convolution layers. A semi-supervised approach was employed whereby the models were trained to reconstruct the normal sequences only. This technique is useful in domains such as anomaly detection where the anomalous data is either scarce or expensive to acquire. In addition, the objective of determining the effectiveness of 2D estimated points from an image, when compared to data acquired from sensors (such as inertial and depth) was also evaluated after conducting a number of experiments (Section 5.2).

The final objective of measuring the effectiveness of the models was presented in Chapter 5 and was structured in three parts. In the first part, the results and evaluation techniques for the comparison between sensor and pose estimated data were presented and discussed (Section 5.2). The second part dealt with the presentation of the results and evaluation metrics used to evaluate the main aim of this dissertation (Section 5.3.5). And finally, in the third part, the acquired results were compared to similar work done on the same dataset (Section 5.4).

The results from the first part demonstrated that pose estimated data from RGB videos is indeed comparable to sensor data classifying human actions. This concludes that with a traditional camera (which is cheap and easy to acquire) and a pose estimation algorithm, one could collect human data as well as build models that recognise human actions, without relying on extra hardware, setup and/or wearable sensors. This shows that the objective of training a number of supervised models to compare sensor data with estimated body keypoints was achieved.

In the second part, both AutoEncoder models that were designed to learn the normal behaviour (and subsequently detect anomalous behaviour through the reconstruction error), were evaluated on two different datasets. Overall, in both datasets and with both pose estimation algorithms, the models managed to correctly distinguish between normal and abnormal sequences of data. On the challenging ADL dataset, the PR AUC scores for 5 cameras ranged between 0.86 to 0.95. When compared to the work carried out on the same dataset by Debard et al. [19], the obtained results contained an improvement of ≈ 0.30 on the PR AUC. This answers the main research question of this study and shows that the models trained on solely body estimated keypoints are indeed capable of detecting abnormal human behaviour from video footage.

6.1 Limitations and Future Work

This section presents a list of future improvements together with some limitations of this study, the latter due to time constraints.

Variable length activities - At present, data sequences that exceed the set confidence score are segmented into different window lengths of 1, 2 and 3 seconds. Distinct models are then trained on different window lengths as discussed in Section 4.5.1.7. Such window lengths capture fixed length activities and may not capture the whole activity taking place. An improvement to this would be to capture variable length activities. This could be done by using a weighted similarity metric on the estimated keypoints whereby a high value would imply a change in activity. Such metric was implemented using the same technique used by Google in their AI experiment called 'Move Mirror'[1]. This metric, which captures a weighted similarity between two poses, was applied to one of the ADL videos on the first 20,000 frames. Figure 6.1 illustrates the output of such metric in green. The blue and orange line plots show the x and y values respectively. The actions were manually marked on the plot where the change in posture is clearly indicated by the higher weighted distance value. In this case, the activities will be segmented when a change in posture is detected.

[1]`http://g.co/movemirror`

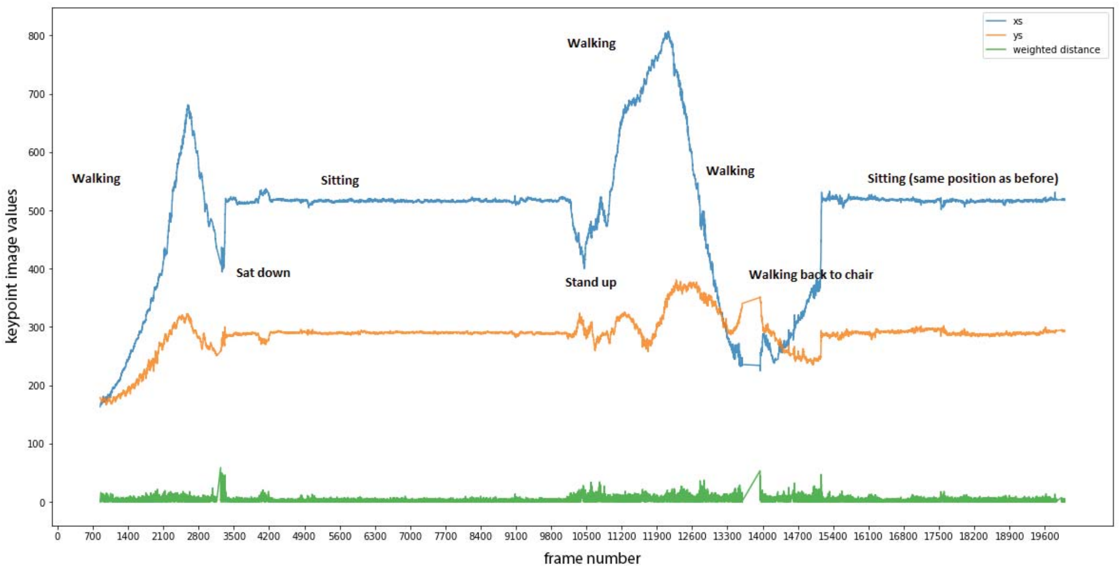


Figure 6.1: Detecting change in posture

Low quality scenes - To date, the available pose estimation models are unable to handle scenes which are very low in quality. As outlined in Section 5.3.5, the dataset contained low light scenes where the human subject is seen on the bed covered with sheets. Apart from the fact that a lot of body keypoints were 'hidden' by the bed sheets, the only available source of light was infra-red. Another limitation in realistic video datasets, as seen in the ADL dataset, is when human subjects are heavily occluded by objects or even if they are partially out-of-frame. Such conditions make the pose estimation process harder and, as a result, the algorithm fails to provide any detections. An improvement to this would be to manually collect and annotate challenging scenes and re-train the pre-trained models. This process, however, may be expensive and time-consuming.

3D Pose Estimation - The method proposed in this dissertation uses 2D pre-trained pose estimation model to extract raw features from 2D images. Such models, as outlined in Section 4.4 output a 2D vector containing the x and y positions for each body keypoint. An extension to this study would be the exploration of the use of pose estimation models that output a 3D representation. A recent model codenamed DensePose [29] aims at mapping all human body parts to a 3D surface. It is also able to provide a 3D representation of the body keypoints in segmented body parts. Since this provides more information, it allows a richer representation of the human subjects to be extracted which could potentially contribute to the model's skill.

2D ConvLSTM - Outlined in Section 3.5, a ConvLSTM layer is a variant of LSTM where unlike the fully-connected LSTM (FC-LSTM), the operations have convolution structures. Researchers observed that such a model is able to capture spatio-temporal correlations better than the FC-LSTM [76]. Although the authors trained a model to classify data, this method can also be utilised for detecting anomalous data patterns. In view of this, a further improvement to this study would be to transform the data to a 2D representation and then train an AutoEncoder model comprised of ConvLSTM layers.

Forecasting the next move - Both the LSTM-AE and the 1DConv-AE were trained to reconstruct the given normal input. An interesting extension to this research would be to train a forecasting model. In this case, instead of training the model to reconstruct the whole input, the model would be trained to predict the next time-steps values and to forecast the next values by feeding past time-series values as input and future time-series values as output. The prediction error would then be used to assess the likelihood of anomalous events. A low prediction error would indicate that the move was expected and thus considered to be 'normal'. On the contrary, a high prediction error would indicate an anomalous event.

Do nearby objects contribute to the outcome? - Another extension to this study would be determining whether context in a given environment, such as objects, contribute to detecting abnormal behaviour. A pre-trained CNN model trained to detect various objects can be used to output a vector representing the objects in the scene. A similarity based metric (such as cosine similarity) can then be used to measure the similarities between the training and the testing scene. Such similarity will act as a weight to the anomaly score and indicate whether context contributes to detect irregular behaviour or not. Such technique would be useful for transferring knowledge learnt from one camera to another.

6.2 Final Remarks

This research introduced a novel semi-supervised learning technique for detecting abnormal human behaviour through videos. The proposed solution was extensively evaluated on two datasets, one of them including very challenging scenes. The outlined results demonstrated the effectiveness of the proposed AutoEncoder models, that were able to correctly distinguish between normal and abnormal data sequences. The results also showed that the proposed method outperformed similar work done on the same dataset. The fundamental findings of this approach are two-fold. Firstly, it was determined that pose estimated data from video frames compares well with sensor data and moreover can be informative enough to classify human actions. Secondly, the high F-score rates, which provide a combined metric, show that the models are capable of classifying both normal and abnormal instances with a very low rate of false negatives. Thus, this technique can be utilised by existing (or new) surveillance systems that are aiming to enable the detection of human abnormal activities.

A

Appendix

A.1 Pose Estimation Output

```
...
{
        "score":0.9942124485969543,
        "part":"leftShoulder",
        "position":{
                "x":104.41399571381079,
                "y":207.2286595487681
        }
},
{
        "score":0.989216685295105,
        "part":"rightShoulder",
        "position":{
                "x":160.60633697045972,
                "y":193.82655331713383
        }
},
{
        "score":0.9260856509208679,
        "part":"leftElbow",
        "position":{
                "x":94.56689527331876,
                "y":258.8360584666026
```

```
                    }
          },
...
```

Listing A.1: Sample of JSON object for 3 keypoints returned by PoseNet. First key “score” is the confidence score of the keypoint, second key “part” refers to the name of the body part and the third key “position” contains the x and y location points of the keypoint.

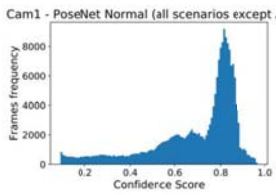
```
[
          ... (other keypoints of this frame)

          0.9942124485969543,
          104.41399571381079,
          207.2286595487681,
          "images_test/Cam1/ADL2/1//ADL2_Cam1.avi-0000000800.jpg",

          0.989216685295105,
          160.60633697045972,
          193.82655331713383,
          "images_test/Cam1/ADL2/1//ADL2_Cam1.avi-0000000800.jpg",

          0.9260856509208679,
          94.56689527331876,
          258.8360584666026,
          "images_test/Cam1/ADL2/1//ADL2_Cam1.avi-0000000800.jpg"

          ... (other keypoints of this frame)
],
[
          ... (other keypoints of the next frame)
]
```

Listing A.2: Sample of final JSON file for 3 keypoints saved for further processing. First element refers to the keypoint score, second and third elements refer to the x and y positions respectively and the fourth element represents the file name.

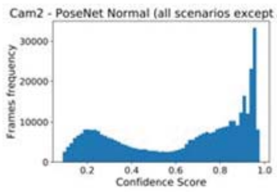
```
{
          "version":1.2,
        "people":[
                {
                        "pose_keypoints_2d":[
                        184.752,
                        53.1586,
                        0.801579,

                        186.092,
                        58.4105,
                        0.900784,

                        172.835,
                        59.7477,
                        0.872283,

                        ... (more keypoints)
                        ]
                }
        ]
}
```

Listing A.3: Sample ouput JSON file created by OpenPose. The file shows 3 keypoints (out of 25) together with the confidence score of each. The first and second elements refer to the x and y positions of the keypoint respectively and the third element refers to the confidence score of the point.

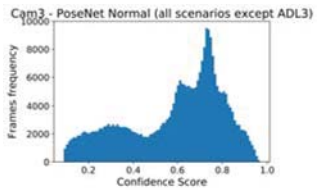
A.2 Histograms for all cameras for both models



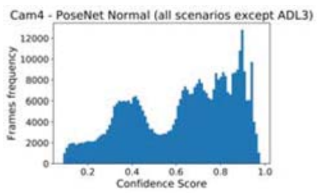
(a) PoseNet: Camera 1 - Normal



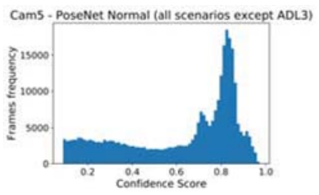
(b) PoseNet: Camera 2 - Normal



(c) PoseNet: Camera 3 - Normal

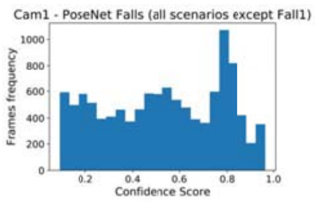


(d) PoseNet: Camera 4 - Normal

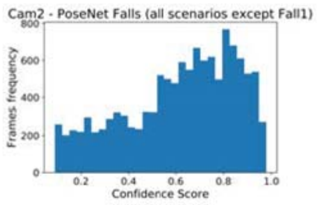


(e) PoseNet: Camera 5 - Normal

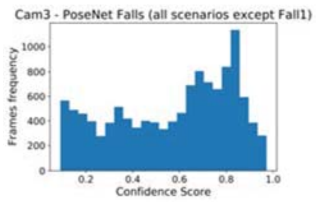
Figure A.1: ADL - PoseNet confidence score histograms for all cameras (normal)



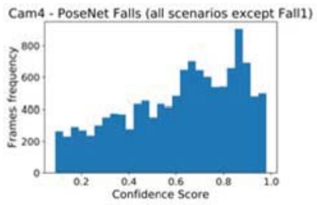
(a) PoseNet: Camera 1 - Abnormal



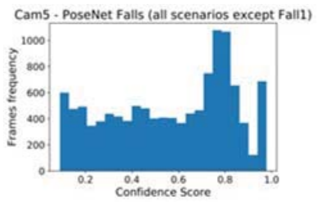
(b) PoseNet: Camera 2 - Abnormal



(c) PoseNet: Camera 3 - Abnormal

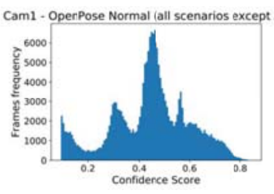


(d) PoseNet: Camera 4 - Abnormal

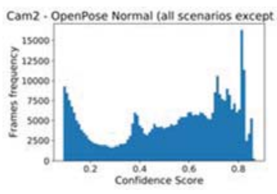


(e) PoseNet: Camera 5 - Abnormal

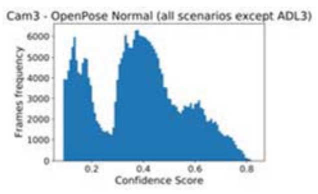
Figure A.2: ADL - PoseNet confidence score histograms for all cameras (abnormal)



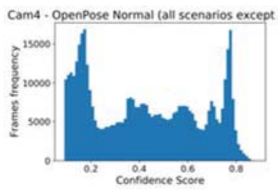
(a) OpenPose: Camera 1 - Normal



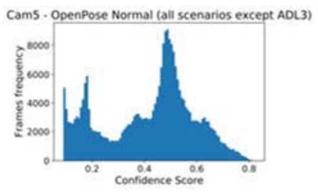
(b) OpenPose: Camera 2 - Normal



(c) OpenPose: Camera 3 - Normal

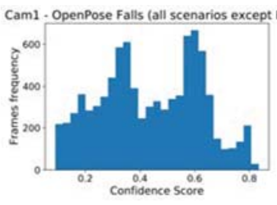


(d) OpenPose: Camera 4 - Normal

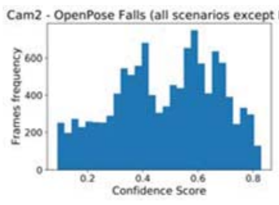


(e) OpenPose: Camera 5 - Normal

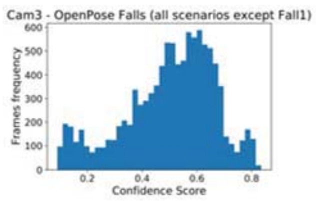
Figure A.3: ADL - OpenPose confidence score histograms for all cameras (normal)



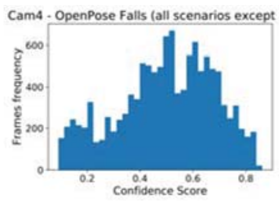
(a) OpenPose: Camera 1 - Abnormal



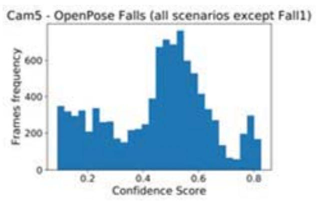
(b) OpenPose: Camera 2 - Abnormal



(c) OpenPose: Camera 3 - Abnormal



(d) OpenPose: Camera 4 - Abnormal



(e) OpenPose: Camera 5 - Abnormal

Figure A.4: ADL - OpenPose confidence score histograms for all cameras (abnormal)

A.3 Results for different hyper-parameter values

Model	LSTM Units or CNN Filters	MCS-WL	Mean PR AUC (all cams)
LSTM-AE	64	0.5-3	0.86
LSTM-AE	64	0.4-3	0.76
LSTM-AE	128 *	0.5-1 *	0.86
LSTM-AE	128 *	0.4-2 *	0.77
LSTM-AE	256	0.5-1	0.88
LSTM-AE	256	0.4-1	0.78
1DConv-AE	64 *	0.5-3 *	0.9
1DConv-AE	64 *	0.4-3 *	0.83
1DConv-AE	128	0.5-3	0.92
1DConv-AE	128	0.4-3	0.86
1DConv-AE	256	0.5-3	0.93
1DConv-AE	256	0.4-3	0.88

Table A.1: PoseNet - Top performing models with different hyper-parameter values. Rows marked with * denote the results for the hyper-parameters originally set. The results demonstrate marginal differences for the LSTM-AE model and a slight improvement for the 1DConv-AE model.

Model	**LSTM Units or CNN Filters**	**MCS-WL**	**Mean PR AUC (all cams)**
LSTM-AE	64	0.1-1	0.83
LSTM-AE	64	0.4-1	0.77
LSTM-AE	128 *	0.1-1 *	0.87
LSTM-AE	128 *	0.4-1 *	0.79
LSTM-AE	256	0.1-1	0.89
LSTM-AE	256	0.4-1	0.81
1DConv-AE	64 *	0.1-1 *	0.76
1DConv-AE	64 *	0.4-1 *	0.71
1DConv-AE	128	0.1-1	0.8
1DConv-AE	128	0.4-1	0.79
1DConv-AE	256	0.1-1	0.83
1DConv-AE	256	0.4-1	0.82

Table A.2: OpenPose - Top performing models with different hyper-parameter values. Rows marked with * denote the results for the hyper-parameters originally set. The results demonstrate marginal differences for the LSTM-AE model and a slight improvement for the 1DConv-AE model.

B Installation Instructions

This section contains the installation instructions for executing the code presented with this dissertation as well as replicate the results. The code was divided into four main components:

1. Frame extraction and concatenation of body keypoint files
2. Pose Estimation
3. Detecting abnormal behaviour: Pre-processing data, training and evaluating the models
4. Comparing Pose estimated data with Sensor data: Pre-processing data, training and evaluating the models

B.1 Prerequisites

The following is a list of software that is required to execute the code:

- Python 3
- Keras
- Tensorflow
- Jupyter Notebook
- Numpy
- Pandas

- OpenCV
- Sklearn
- OpenPose (Windows module)
- Apache/NGINX HTTP Server
- PHP
- FFmpeg

All components were fully tested on Windows 10 and partially tested on Ubuntu 16.04.

B.2 Frame extraction and Concatenation

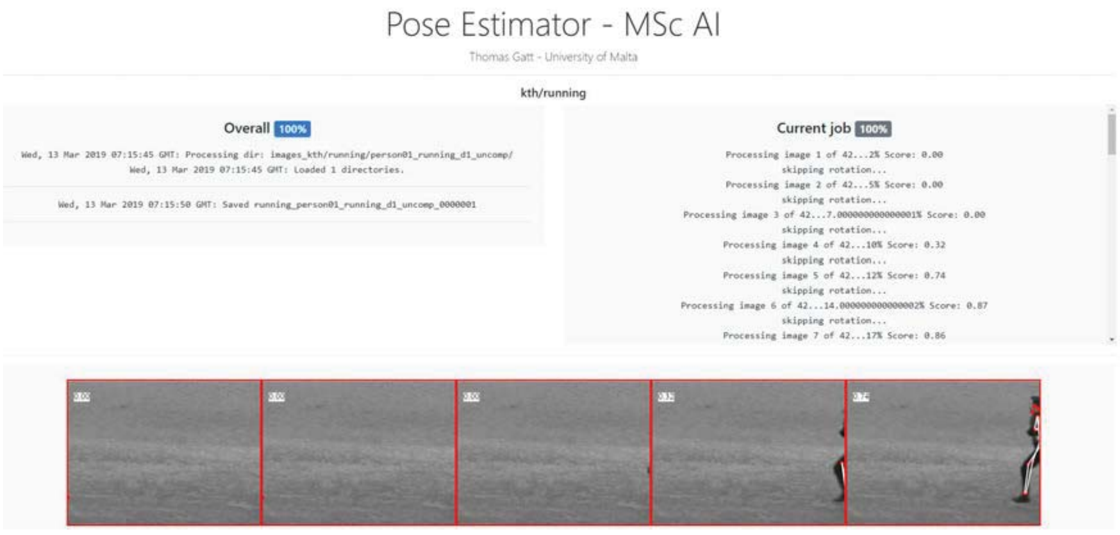
The first component is responsible for processing and extracting frames from videos as well as concatenating the data after it is processed by the pose estimation algorithm. This covers the code for modules 1 and 2 outlined in Section 4.2. The file is called **ExtractFrames_ConcatenateFiles.ipynb** and is in a form of a Jupyter notebook. The file consists of a number of cells and is responsible for the following tasks:

- extracting frames from video (using FFmpeg)
- extract specific annotated frames from video
- splitting and sorting files into folders
- processes pose estimation for OpenPose
- handles all dataset processing for modules 1 and 2 (KTH, UTH-MHAD and ADL)

B.3 Pose Estimation

The second component is responsible for extracting the body keypoints from a set of images. This component was developed with Tensorflow.JS and uses a pre-trained model called PoseNet. A screenshot of this component is presented in Figure B.1, which shows a sample output on the KTH dataset. Since this component is in a form of a web application, an HTTP server such as Apache is required. PHP also needs to be installed with the server as the tool uses PHP to save the keypoints to file. This component contains more than one file and therefore is packaged in a folder called **PoseEstimator**. The main files in this package include:

- index.html - Loads TensorFlow.JS and the required packages, then it loads and processes all images stored in set folder and estimates the body keypoints for each image.
- functions.js - Functions used by index.html to visualise the body keypoints and perform image rotations, amongst others.
- save.php - PHP file responsible for saving the output for every folder in JSON format.
- images folder - Folder with a number of images ready to be processed
- output folder - Contains list of estimated keypoints



```
Wed, 13 Mar 2019 07:15:45 GMT: Processing dir: images_kth/running/person01_running_d1_uncomp/
Wed, 13 Mar 2019 07:15:45 GMT: Loaded 1 directories.
Wed, 13 Mar 2019 07:15:50 GMT: Saved running_person01_running_d1_uncomp_0000001
```

```
Processing image 1 of 42...2% Score: 0.00
skipping rotation...
Processing image 2 of 42...5% Score: 0.00
skipping rotation...
Processing image 3 of 42...7.000000000000001% Score: 0.00
skipping rotation...
Processing image 4 of 42...10% Score: 0.32
skipping rotation...
Processing image 5 of 42...12% Score: 0.74
skipping rotation...
Processing image 6 of 42...14.000000000000002% Score: 0.87
skipping rotation...
Processing image 7 of 42...17% Score: 0.86
```

Figure B.1: Pose Estimator demo on KTH dataset

B.4 Detecting abnormal behaviour

The third component is responsible loading and pre-processing the data extracted from both pose estimation algorithms. This covers the code for modules 3 and 4 outlined in Section 4.2. The file is called **Preprocess_Train_LSTM_Conv1D.ipynb** and is in a form of a Jupyter notebook. The file consists of a number of cells, each of which are well documented, and contains the following:

- library imports

- functions for data pre-processing (including plotting functions)
- functions for training both AutoEncoder models (LSTM-AE and 1DConv-AE)
- functions for evaluating both models
- execution of the above functions (training, testing and evaluation)

B.5 Comparing Pose data with Sensor data

The final and fourth component is responsible for training separate models in order to compare pose estimated data with sensor data. As outlined in Section 4.6, different data representations are loaded, trained separately and compared. The code for such component is also in a form a Jupyter notebook and is named **PoseVsSensor_PreProcess_Train.ipynb**. This file contains the code for:

- importing required libraries
- loading all datasets (RGB - PoseNet and OpenPose representations, Inertial data, Skeleton 2D and 3D)
- model architecture for training the supervised model
- padding sequences and preparing the dataset for training
- splitting datasets to training, testing and validation sets (CV)
- training the models
- evaluating the models

B.6 Pre-processed data and trained models

In addition to the code components presented with this study, folders containing the pre-processed data and pre-trained models are included. Such pre-processed data was used to train both models in various experiments that were documented in Chapter 5. The trained models were saved for evaluation purposes and are being provided as an additional resource. Due to size limitation, only a sample of the data is being provided.

References

[1] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2014.

[2] Jinwon An and Sungzoon Cho. Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE*, 2:1–18, 2015.

[3] Panagiota Antonakaki, Dimitrios I. Kosmopoulos, and Stavros J. Perantonis. Detecting abnormal human behaviour using multiple cameras. *Signal Processing*, 89(9):1723–1738, 2009. doi: 10.1016/j.sigpro.2009.03.016.

[4] Moez Baccouche, Franck Mamalet, Christian Wolf, Christophe Garcia, and Atilla Baskurt. Sequential deep learning for human action recognition. In *Human Behavior Unterstanding - Second International Workshop, HBU 2011, Amsterdam, The Netherlands, November 16, 2011. Proceedings*, pages 29–39, 2011. doi: 10.1007/978-3-642-25446-8_4. URL `https://doi.org/10.1007/978-3-642-25446-8_4`.

[5] Greet Baldewijns, Glen Debard, Gert Mertes, Bart Vanrumste, and Tom Croonenborghs. Bridging the gap between real-life data and simulated data by providing a highly realistic fall dataset for evaluating camera-based fall detection algorithms. *Healthcare technology letters*, 3(1):6–11, 2016.

[6] Oresti Baños, Juan Manuel Galvez, Miguel Damas, Héctor Pomares, and Ignacio Rojas. Window size impact in human activity recognition. *Sensors*, 14(4):6474–6499, 2014. doi: 10.3390/s140406474.

[7] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*, pages 153–160, 2006. URL `http://papers.nips.cc/paper/3048-greedy-layer-wise-training-of-deep-networks`.

[8] Andrew P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, 1997. doi: 10.1016/S0031-3203(96)00142-2.

[9] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 1302–1310, 2017. doi: 10.1109/CVPR.2017.143. URL `https://doi.org/10.1109/CVPR.2017.143`.

[10] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, 2009. doi: 10.1145/1541880.1541882.

[11] Chen Chen, Roozbeh Jafari, and Nasser Kehtarnavaz. UTD-MHAD: A multimodal dataset for human action recognition utilizing a depth camera and a wearable inertial sensor. In *2015 IEEE International Conference on Image Processing, ICIP 2015, Quebec City, QC, Canada, September 27-30, 2015*, pages 168–172, 2015. doi: 10.1109/ICIP.2015.7350781. URL `https://doi.org/10.1109/ICIP.2015.7350781`.

[12] Nancy Chinchor. MUC-4 evaluation metrics. In *Proceedings of the 4th Conference on Message Understanding, MUC 1992, McLean, Virginia, USA, June 16-18, 1992*, pages 22–29. Association for Computational Linguistics, 1992. doi: 10.3115/1072064.1072067. URL `https://doi.org/10.3115/1072064.1072067`.

[13] Heeryon Cho and Sang Min Yoon. Divide and conquer-based 1d CNN human activity recognition using test data sharpening. *Sensors*, 18(4):1055, 2018. doi: 10.3390/s18041055.

[14] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1724–1734, 2014. URL `http://aclweb.org/anthology/D/D14/D14-1179.pdf`.

[15] Francois Chollet. Building autoencoders in keras. `https://blog.keras.io/building-autoencoders-in-keras.html`, 2016. Accessed: 2018-09-28.

[16] Yong Shean Chong and Yong Haur Tay. Abnormal event detection in videos using spatiotemporal autoencoder. In *Advances in Neural Networks - ISNN 2017 - 14th International Symposium, ISNN 2017, Sapporo, Hakodate, and Muroran, Hokkaido, Japan, June 21-26, 2017, Proceedings, Part II*, pages 189–196, 2017. doi: 10.1007/978-3-319-59081-3_23. URL `https://doi.org/10.1007/978-3-319-59081-3_23`.

[17] Jesse Davis and Mark Goadrich. The relationship between precision-recall and ROC curves. In *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, pages 233–240, 2006. doi: 10.1145/1143844.1143874. URL `https://doi.org/10.1145/1143844.1143874`.

[18] Jesse Davis, Elizabeth S. Burnside, Inês de Castro Dutra, David Page, Raghu Ramakrishnan, Vítor Santos Costa, and Jude W. Shavlik. View learning for statistical relational learning: With an application to mammography. In *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, pages 677–683, 2005. URL `http://ijcai.org/Proceedings/05/Papers/0664.pdf`.

[19] Glen Debard, Greet Baldewijns, Toon Goedemé, Tinne Tuytelaars, and Bart Vanrumste. Camera-based fall detection using a particle filter. In *37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBC 2015, Milan, Italy, August 25-29, 2015*, pages 6947–6950, 2015. doi: 10.1109/EMBC.2015.7319990. URL `https://doi.org/10.1109/EMBC.2015.7319990`.

[20] Thomas G Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural computation*, 10(7):1895–1923, 1998.

[21] K SRIVASTAVA Durgesh and B Lekha. Data classification using support vector machine. *Journal of Theoretical and Applied Information Technology*, 12(1):1–7, 2010.

[22] Mark J. F. Gales and Steve J. Young. The application of hidden markov models in speech recognition. *Foundations and Trends in Signal Processing*, 1(3):195–304, 2007. doi: 10.1561/2000000004.

[23] Samuele Gasparrini, Enea Cippitelli, Susanna Spinsante, and Ennio Gambi. A depth-based fall detection system using a kinect® sensor. *Sensors*, 14(2):2756–2775, 2014. doi: 10.3390/s140202756.

[24] Felix A. Gers, Jürgen Schmidhuber, and Fred A. Cummins. Learning to forget: Continual prediction with LSTM. *Neural Computation*, 12(10):2451–2471, 2000. doi: 10.1162/089976600300015015.

[25] Zoubin Ghahramani. An introduction to hidden markov models and bayesian networks. *International journal of pattern recognition and artificial intelligence*, 15(1):9–42, 2001. doi: 10.1142/S0218001401000836.

[26] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016. ISBN 978-0-262-03561-3. URL `http://www.deeplearningbook.org/`.

[27] Nico Görnitz, Mikio L. Braun, and Marius Kloft. Hidden markov anomaly detection. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 1833–1842, 2015. URL `http://jmlr.org/proceedings/papers/v37/goernitz15.html`.

[28] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *IEEE Trans. Neural Netw. Learning Syst.*, 28(10):2222–2232, 2017. doi: 10.1109/TNNLS.2016.2582924.

[29] Riza Alp Güler, Natalia Neverova, and Iasonas Kokkinos. Densepose: Dense human pose estimation in the wild. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 7297–7306, 2018. doi: 10.1109/CVPR.2018.00762. URL `http://openaccess.thecvf.com/content_cvpr_2018/html/Guler_DensePose_Dense_Human_CVPR_2018_paper.html`.

[30] Matheus Gutoski, Nelson Marcelo Romero Aquino, Manassés Ribeiro, André Engênio Lazzaretti, and Heitor Silvério Lopes. Detection of video anomalies using convolutional autoencoders and one-class support vector machines. In *Proc. XIII Brazilian Congress on Computational Intelligence*, 2017.

[31] Fei Han, Brian Reily, William Hoff, and Hao Zhang. Space-time representation of people based on 3d skeletal data: A review. *Computer Vision and Image Understanding*, 158:85–105, 2017. doi: 10.1016/j.cviu.2017.01.011.

[32] Mahmudul Hasan, Jonghyun Choi, Jan Neumann, Amit K. Roy-Chowdhury, and Larry S. Davis. Learning temporal regularity in video sequences. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 733–742, 2016. doi: 10.1109/CVPR.2016.86. URL `https://doi.org/10.1109/CVPR.2016.86`.

[33] Nima Hatami, Yann Gavet, and Johan Debayle. Classification of time-series images using deep convolutional neural networks. In *Tenth International Conference on Machine Vision (ICMV 2017)*, volume 10696, page 106960Y. International Society for Optics and Photonics, 2018.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90. URL `https://doi.org/10.1109/CVPR.2016.90`.

[35] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313:504–507, 2006. doi: 10.1126/science.1127647.

[36] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.

[37] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[38] Ya-Xuan Hung, Chih-Yen Chiang, Steen J. Hsu, and Chia-Tai Chan. Abnormality detection for improving elder's daily life independent. In *Aging Friendly Technology for Health and Independence, 8th International Conference on Smart Homes and Health Telematics, ICOST 2010, Seoul, Korea, June 22-24, 2010. Proceedings*, pages 186–194, 2010. doi: 10.1007/978-3-642-13778-5_23. URL `https://doi.org/10.1007/978-3-642-13778-5_23`.

[39] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[40] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(1):221–231, 2013. doi: 10.1109/TPAMI.2012.59.

[41] Wenchao Jiang and Zhaozheng Yin. Human activity recognition using wearable sensors by deep convolutional neural networks. In *Proceedings of the 23rd Annual ACM Conference on Multimedia Conference, MM '15, Brisbane, Australia, October 26 - 30, 2015*, pages 1307–1310, 2015. doi: 10.1145/2733373.2806333. URL `https://doi.org/10.1145/2733373.2806333`.

[42] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[43] B. Ravi Kiran, Dilip Mathew Thomas, and Ranjith Parakkal. An overview of deep learning based methods for unsupervised and semi-supervised anomaly detection in videos. *J. Imaging*, 4(2):36, 2018. doi: 10.3390/jimaging4020036.

[44] Serkan Kiranyaz, Turker Ince, and Moncef Gabbouj. Real-time patient-specific ECG classification by 1-d convolutional neural networks. *IEEE Trans. Biomed. Engineering*, 63(3):664–675, 2016. doi: 10.1109/TBME.2015.2468589.

[45] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25: 26th Annual*

Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States., pages 1106–1114, 2012. URL `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks`.

[46] Javier Ortiz Laguna, Angel García Olaya, and Daniel Borrajo. A dynamic sliding window approach for activity recognition. In *User Modeling, Adaption and Personalization - 19th International Conference, UMAP 2011, Girona, Spain, July 11-15, 2011. Proceedings*, pages 219–230. Springer, 2011. doi: 10.1007/978-3-642-22362-4_19. URL `https://doi.org/10.1007/978-3-642-22362-4_19`.

[47] César Laurent, Gabriel Pereyra, Philemon Brakel, Ying Zhang, and Yoshua Bengio. Batch normalized recurrent neural networks. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2016, Shanghai, China, March 20-25, 2016*, pages 2657–2661, 2016. doi: 10.1109/ICASSP.2016.7472159. URL `https://doi.org/10.1109/ICASSP.2016.7472159`.

[48] Quoc V Le et al. A tutorial on deep learning part 2: autoencoders, convolutional neural networks and recurrent neural networks. *Google Brain*, pages 1–20, 2015.

[49] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[50] Fei-Fei Li and Andrej Karpathy. Convolutional neural networks for visual recognition, 2015.

[51] Hyungui Lim, Jeongsoo Park, and Y Han. Rare sound event detection using 1d convolutional recurrent neural networks. In *Proc. of DCASE*, pages 80–84, 2017.

[52] Zachary Chase Lipton. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, abs/1506.00019, 2015.

[53] Jian Liu, Naveed Akhtar, and Ajmal Mian. Skepxels: Spatio-temporal image representation of human skeleton joints for action recognition. *CoRR*, abs/1711.05941, 2017.

[54] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. Long short term memory networks for anomaly detection in time series. In *23rd European Symposium on Artificial Neural Networks, ESANN 2015, Bruges, Belgium, April 22-24, 2015*, 2015. URL `http://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2015-56.pdf`.

[55] Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Lstm-based encoder-decoder for multi-sensor anomaly detection. *arXiv preprint arXiv:1607.00148*, abs/1607.00148, 2016.

[56] Julieta Martinez, Rayat Hossain, Javier Romero, and James J. Little. A simple yet effective baseline for 3d human pose estimation. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2659–2668, 2017. doi: 10.1109/ICCV.2017.288. URL `https://doi.org/10.1109/ICCV.2017.288`.

[57] Fulufhelo Vincent Nelwamondo, Dan Golding, and Tshilidzi Marwala. A dynamic programming approach to missing data estimation using neural networks. *Inf. Sci.*, 237:49–58, 2013. doi: 10.1016/j.ins.2009.10.008.

[58] Christopher Olah. Understanding lstm networks. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`, 2015. Accessed: 2018-10-5.

[59] World Health Organization et al. Ageing; life course unit. who global report on falls prevention in older age. *World Health Organization*, 2008.

[60] Ayse Ozcan, Hulya Donat, Nihal Gelecek, Mehtap Ozdirenc, and Didem Karadibak. The relationship between risk factors for falling and the quality of life in older adults. *BMC Public Health*, 5(1):90, 2005.

[61] George Papandreou, Tyler Zhu, Nori Kanazawa, Alexander Toshev, Jonathan Tompson, Chris Bregler, and Kevin Murphy. Towards accurate multi-person pose estimation in the wild. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 3711–3719, 2017. doi: 10.1109/CVPR.2017.395. URL `https://doi.org/10.1109/CVPR.2017.395`.

[62] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pages 1310–1318, 2013. URL `http://jmlr.org/proceedings/papers/v28/pascanu13.html`.

[63] Xiaojiang Peng and Cordelia Schmid. Multi-region two-stream R-CNN for action detection. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, pages 744–759, 2016. doi: 10.1007/978-3-319-46493-0_45. URL `https://doi.org/10.1007/978-3-319-46493-0_45`.

[64] Lutz Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the Trade*, pages 55–69. 1996. doi: 10.1007/3-540-49430-8_3. URL `https://doi.org/10.1007/3-540-49430-8_3`.

[65] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 91–99, 2015. URL `http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks`.

[66] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 1278–1286, 2014. URL `http://jmlr.org/proceedings/papers/v32/rezende14.html`.

[67] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144, 2016. doi: 10.1145/2939672.2939778. URL `https://doi.org/10.1145/2939672.2939778`.

[68] C. Rougier, J. Meunier, A. St-Arnaud, and J. Rousseau. Monocular 3d head tracking to detect falls of elderly people. In *2006 International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 6384–6387. IEEE, Aug 2006. doi: 10.1109/IEMBS.2006.260829.

[69] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, abs/1609.04747, 2016.

[70] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[71] Tara N. Sainath, Oriol Vinyals, Andrew W. Senior, and Hasim Sak. Convolutional, long short-term memory, fully connected deep neural networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2015, South Brisbane, Queensland, Australia, April 19-24, 2015*, pages 4580–4584, 2015. doi: 10.1109/ICASSP.2015.7178838. URL `https://doi.org/10.1109/ICASSP.2015.7178838`.

[72] Takaya Saito and Marc Rehmsmeier. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PloS one*, 10(3):e0118432, 2015.

[73] Bernhard Schölkopf, Robert C. Williamson, Alexander J. Smola, John Shawe-Taylor, and John C. Platt. Support vector method for novelty detection. In *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, pages 582–588, 1999. URL `http://papers.nips.cc/paper/1723-support-vector-method-for-novelty-detection`.

[74] Christian Schüldt, Ivan Laptev, and Barbara Caputo. Recognizing human actions: A local SVM approach. In *17th International Conference on Pattern Recognition, ICPR 2004, Cambridge, UK, August 23-26, 2004.*, pages 32–36, 2004. doi: 10.1109/ICPR.2004.1334462. URL `https://doi.org/10.1109/ICPR.2004.1334462`.

[75] Amir Shahroudy, Jun Liu, Tian-Tsong Ng, and Gang Wang. NTU RGB+D: A large scale dataset for 3d human activity analysis. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 1010–1019, 2016. doi: 10.1109/CVPR.2016.115. URL `https://doi.org/10.1109/CVPR.2016.115`.

[76] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 802–810, 2015. URL `http://papers.nips.cc/paper/5955-convolutional-lstm-network-a-machine-learning-approach-for-precipitation-nowcasting`.

[77] Jae Hyuk Shin, Boreom Lee, and Kwang Suk Park. Detection of abnormal living patterns for elderly living alone using support vector data description. *IEEE Trans. Information Technology in Biomedicine*, 15(3):438–448, 2011. doi: 10.1109/TITB.2011.2113352.

[78] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 568–576, 2014. URL `http://papers.nips.cc/paper/5353-two-stream-convolutional-networks-for-action-recognition-in-videos`.

[79] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, abs/1409.1556, 2014.

[80] Leslie N. Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision, WACV 2017, Santa Rosa, CA, USA, March 24-31, 2017*, pages 464–472, 2017. doi: 10.1109/WACV.2017.58. URL `https://doi.org/10.1109/WACV.2017.58`.

[81] Sijie Song, Cuiling Lan, Junliang Xing, Wenjun Zeng, and Jiaying Liu. An end-to-end spatio-temporal attention model for human action recognition from skeleton data. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 4263–4270, 2017. URL `http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14437`.

[82] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[83] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. Unsupervised learning of video representations using lstms. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 843–852, 2015. URL `http://jmlr.org/proceedings/papers/v37/srivastava15.html`.

[84] Jiayu Sun, Jie Shao, and Chengkun He. Abnormal event detection for video surveillance using deep one-class learning. *Multimedia Tools Appl.*, 78(3):3633–3647, 2019. doi: 10.1007/s11042-017-5244-2.

[85] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3104–3112, 2014. URL `http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks`.

[86] Gaurav Tripathi, Kuldeep Singh, and Dinesh Kumar Vishwakarma. Convolutional neural networks for crowd behaviour analysis: a survey. *The Visual Computer*, pages 1–24, 2018.

[87] T. v. Kasteren and B. Krose. Bayesian activity recognition in residence for elders. In *2007 3rd IET International Conference on Intelligent Environments*, pages 209–212, Sep. 2007. doi: 10.1049/cp:20070370.

[88] Tim van Kasteren, Gwenn Englebienne, and Ben J. A. Kröse. An activity monitoring system for elderly care using generative and discriminative models. *Personal and Ubiquitous Computing*, 14(6): 489–498, 2010. doi: 10.1007/s00779-009-0277-9.

[89] Gaël Varoquaux. Cross-validation failure: Small sample sizes lead to large error bars. *NeuroImage*, 180(Part):68–77, 2018. doi: 10.1016/j.neuroimage.2017.06.061.

[90] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11:3371–3408, 2010.

[91] Jiang Wang, Xiaohan Nie, Yin Xia, Ying Wu, and Song-Chun Zhu. Cross-view action modeling, learning, and recognition. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, pages 2649–2656, 2014. doi: 10.1109/CVPR.2014.339. URL `https://doi.org/10.1109/CVPR.2014.339`.

[92] Sun-Chong Wang. Artificial neural network. In *Interdisciplinary computing in java programming*, pages 81–100. Springer US, 2003. doi: "10.1007/978-1-4615-0377-4_5". URL "`https://doi.org/10.1007/978-1-4615-0377-4_5`".

[93] Dan Xu, Elisa Ricci, Yan Yan, Jingkuan Song, and Nicu Sebe. Learning deep representations of appearance and motion for anomalous event detection. In *Proceedings of the British Machine Vision Conference 2015, BMVC 2015, Swansea, UK, September 7-10, 2015*, pages 8.1–8.12, 2015. doi: 10.5244/C.29.8. URL `https://doi.org/10.5244/C.29.8`.

[94] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, Jie Chen, Zhaogang Wang, and Honglin Qiao. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, pages 187–196, 2018. doi: 10.1145/3178876.3185996. URL `https://doi.org/10.1145/3178876.3185996`.

[95] Junji Yamato, Jun Ohya, and Kenichiro Ishii. Recognizing human action in time-sequential images using hidden markov model. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 1992, Proceedings, 15-18 June, 1992, Champaign, Illinois, USA*, pages 379–385, 1992. doi: 10.1109/CVPR.1992.223161. URL `https://doi.org/10.1109/CVPR.1992.223161`.

[96] Lei Yang, Yanyun Ren, Huosheng Hu, and Bo Tian. New fast fall detection method based on spatio-temporal context tracking of head by using depth images. *Sensors*, 15(9):23004–23019, 2015. doi: 10.3390/s150923004.

[97] Leiyue Yao, Weidong Min, and Keqiang Lu. A new approach to fall detection based on the human torso motion model. *Applied Sciences*, 7(10):993, 2017.

[98] Jie Yin, Qiang Yang, and Jeffrey Junfeng Pan. Sensor-based abnormal human-activity detection. *IEEE Trans. Knowl. Data Eng.*, 20(8):1082–1090, 2008. doi: 10.1109/TKDE.2007.1042.

[99] Ming Zeng, Le T. Nguyen, Bo Yu, Ole J. Mengshoel, Jiang Zhu, Pang Wu, and Joy Zhang. Convolutional neural networks for human activity recognition using mobile sensors. In *6th International Conference on Mobile Computing, Applications and Services, MobiCASE 2014, Austin, TX, USA, November 6-7, 2014*, pages 197–205, 2014. doi: 10.4108/icst.mobicase.2014.257786. URL `https://doi.org/10.4108/icst.mobicase.2014.257786`.

[100] Yiru Zhao, Bing Deng, Chen Shen, Yao Liu, Hongtao Lu, and Xian-Sheng Hua. Spatio-temporal autoencoder for video anomaly detection. In *Proceedings of the 2017 ACM on Multimedia Conference, MM 2017, Mountain View, CA, USA, October 23-27, 2017*, pages 1933–1941, 2017. doi: 10.1145/3123266.3123451. URL `https://doi.org/10.1145/3123266.3123451`.

[101] Hua Zhong, Jianbo Shi, and Mirkó Visontai. Detecting unusual activity in video. In *2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2004), with CD-ROM, 27 June - 2 July 2004, Washington, DC, USA*, pages 819–826, 2004. doi: 10.1109/CVPR.2004.78. URL `http://doi.ieeecomputersociety.org/10.1109/CVPR.2004.78`.