university of groningen

faculty of science and engineering

Master Thesis

Fall detection with a wearable camera using traditional and deep learning algorithms

Author:
Mark Helmus

Supervisors:
dr. George Azzopardi
Xueyi Wang
dr. Estefanía Talavera Martínez

January 5, 2021

ACKNOWLEDGEMENTS

First, I would like to thank dr. George Azzopardi and Xueyi Wang for providing me the opportunity to develop my scientific skills in this exciting field. Their expertise helped me a lot to get on track and guided me where needed. Further, their quick responses and insightful tips really helped me to make progress in the thesis.

Second, I would like to thank Hichem Bouakaz for the collaboration of the first stage of the thesis. Working and discussing concepts together was very useful in the starting phase of the thesis.

Further, I would like to thank the volunteers for recording the videos that were used for this research. I would also like to thank the high performance experts at the Peregrine HPC cluster for the information and tools provided to work on this cluster.

Finally, I would like to thank my parents, sister and friends for always being there for me.

Abstract

A fall can have severe consequences on one's life, even leading to death if no adequate help is provided. This help can only be provided if the fall is detected. Detecting the fall automatically is preferred over detecting the fall manually. This is, among others, due to the scalability of the system. In this research, two traditional algorithms and one deep learning algorithm are proposed to automatically detect falls from videos filmed with a wearable camera. The traditional algorithms used are dense trajectories (DT) and improved dense trajectories (IDT). The compressed video action recognition algorithm (CoViAR) is the deep learning algorithm used for this research. The algorithms are trained and tested on the data set showing different actions, filmed with the camera mounted to the neck or waist. In total, the data set consists of 1459 videos. The data set is used to train the classifiers on four different problems: fall detection with videos filmed by both cameras; fall detection with videos filmed by the camera mounted to the neck; fall detection with videos filmed by the camera mounted to the waist; egocentric action recognition. After optimizing the hyperparameters of the classifiers, the deep learning algorithm CoViAR outperforms the traditional algorithms on each problem. When detecting falls using videos filmed with cameras mounted to either the neck or waist, CoViAR has a ROC-AUC of 99.82% and an accuracy of 97.95%. For action recognition with a wearable camera, CoViAR has an AUC of 95.09% and an accuracy of 70.9%. The traditional algorithms show that they can be used to detect falls as well. A ROC-AUC of 97.58% and 98.47% for respectively DT and IDT is obtained. Further, no evidence is found that a specific camera mounting point is preferred over another. Evaluating the results, the three selected algorithms can be used to detect falls from videos filmed with a wearable camera. The deep learning algorithm has a slight edge over the traditional algorithms, since it has better performance and does not need to compute optical flow explicitly.

CONTENTS

ABBREVIATIONS

AUC Area under the curve.

CNN Convolutional neural network.
CoViAR Compressed video action recognition.

DT Dense trajectories algorithm.

FN False negative.
FP False positive.
FPR False positive rate.

GMM Gaussian mixture method.

HOF Histogram of Optical Flow.
HOG Histogram of Oriented Gradients.

IDT Improved dense trajectories algorithm.

MBH Motion Boundary Histogram.

PCA Prinipal component analysis.
PR curve Precision-Recall curve.

RANSAC Random Consensus Algorithm.
RBF Radial basis function.
ReLU Rectified linear unit.
RGB Additive color space, using the colors red, blue and green as primaries.
ROC curve Receiver operating characteristic curve.

SURF Speeded Up Robust Features.
SVM Support vector machine.

TN True negative.
TP True positive.
TPR True positive rate.
TSN Temporal Segment Networks.

1 INTRODUCTION

In this chapter, we introduce fall detection to the reader. We further explain current challenges in fall detection. Next, the aim of this research is given, along with research questions. We finish the chapter with an overview of the thesis.

1.1 PROBLEM DEFINITION

A fall can have severe consequences on someone's life. Research conducted by the World Health Organization (WHO) [77] indicates that each year, 28%-35% of people aged 65 years or older fall. The research further shows that the fall rate increases to 32%-42% for people aged 70 years or older.

Older adults are more at risk, as they are more likely to suffer from neurological diseases, possibly resulting in falls [130]. For instance, they could fall due to the consequences of epilepsy. Moreover, older adults are more likely to live alone. Elliott et al. [28] investigated older adults living alone and found that they likewise had an increased risk of falling.

A fall can cause serious injuries, leading to disabling fractures [103]. Further, older adults suffer psychological consequences from falling and fear of falling. Hadjistavropoulos et al. identify that fear of falling is an independent risk factor for reduced quality of life, activity restriction, loss of independence and fall-risk [41]. It is estimated that by 2030, the number of fall induced injuries is doubled, if no protective measures will be taken in the future [48].

Fall detection is essential, as it enables the help of the person who fell. Older people tend to be injured easier due to weaker bones [11], and a quick response to a fall helps to reduce injuries. For instance, when one starts bleeding from the fall, the injury becomes more severe if help arrives later. Thus, reducing injuries lead to shorter recovery periods for the person who fell, which helps the injured person to return to his daily routine at a faster pace. In case no one is able to detect someone who fell, while the fallen person is unable to call for help, one has to find the fallen person by sheer luck. This increases the time between the fall and adequate care, thus increasing the risk of injury.

Further, monitoring a person 24 hours per day, seven days per week to detect possible falls is infeasible. This is infeasible for a couple of reasons. Firstly, monitoring of a person can be seen as an invasive operation regarding privacy. Secondly, the monitoring process is a tedious task for the observer. Thirdly, a fall detection system needing human observers does not scale well. To monitor more people, more observers are needed. Hiring extra observers leads to increased costs. This in turn increases the costs for the monitoring service. Consequently, this might make manual fall detection too expensive. Therefore, a manual fall detection system is impracticable.

As fall detector systems assist in responding fast to falls, therefore reducing

injuries, we notice that recovery periods are shorter due to reduced injuries. These shorter recovery periods lead to shorter treatment periods. The medical staff carries out these treatments. Given the shortened treatment of injury, the medical personnel can shift their focus on patients with other medical problems in a faster pace, reducing their working pressure.

Automating the process of detecting falls is a worthwhile project, as it reduces the risk of severe injuries, reduces expected recovery period, reduces the costs of detecting falls and reduces treatment time given by the medical staff to the patient that fell.

There are multiple concepts to detect falls. Many studies have been conducted using different approaches to detect falls. There are external sensor-based methods, wearable sensor-based methods and a mixture of these two types of methods. Within external sensor-based methods and wearable sensor-based methods there are many different options to choose from. For instance, one can choose a specific sensor such as a fixed camera or accelerometer, or choose to use an egocentric camera.

In this research, we use two different cameras mounted to a person. One camera is mounted to the neck, and the other camera is mounted to the waist. We opt for this method instead of a fixed camera, as we identify that using cameras mounted to the body has advantages over mounting the camera at a fixed place in a room.

The first advantage we identify is that with a fixed camera, we are limited to the place where the camera is mounted. For instance, when placing a camera in the bathroom, a fall is only detected when the user falls in the bathroom. A solution for this issue would be to place cameras everywhere in the house. However, this increases costs for the user, as multiple cameras have to be purchased with current technology.

Further, having cameras setup in different sections of the house could be experienced as intrusive, as the cameras are always observing the user at its home. This in turn could lead to an unpleasant experience for the end-user using the fall detection system with fixed cameras.

1.2 CHALLENGES

There are multiple challenges to detect falls. These challenges have been identified by Igual et al. [48]. A first challenge described in this paper is the performance under real-life conditions. Fall detectors have to be very accurate and reliable. The fall detector should identify falls robustly. Thus, the detectors should exhibit a low number of false positives and false negatives simultaneously.

Another challenge is the positioning of the detectors. For instance, when using smartphones as fall detectors, it should be possible to place the smartphones at places where a person normally places his smartphone, such as his pockets typically. For other devices, the devices should be positioned such that they do not interfere too much with the daily activities of the user.

Acceptance of using a fall detector on a person is another challenge. Older

adults may not be familiar with electronic devices. For instance, when the end-user has to adjust settings of the device by making use of speech recognition, it makes it harder to use if the end-user is not familiar with adjusting the settings via speech recognition. Adoption of a new device often involves a readjustment of well-established routines [64]. Further, one should take into account limitations of mobility, visual and hearing impairments [36].

Ozcan et al. [79] identify that there is no public database available for fall detection in an egocentric based vision. To our knowledge, no data set has been published after the publication of their paper. The lack of a data set is a problem, as researchers have to collect data when they want to conduct experiments, which is a time-consuming task.

Further, Habib et al. [40] identify that energy consumption is another issue. They identified this as a challenge for smartphones, although this challenge is not limited to smartphones only. It applies to other devices as well. The device should be capable of filming when the user assigns the device to do so, since the device is only capable of detecting falls using video data. Therefore battery life should be high, and the battery of the device should be able to recharge fast.

Privacy is another concern. Using an egocentric camera, the device inevitably films other people while it is powered on. Other people may have an issue with being filmed, as they are filmed without giving permission, threatening bystanders' privacy [88]. For instance, if the fall detector is powered on during a confidential business meeting, important comments could be made and unknowingly be recorded. Hence, the data obtained in the device should be confident.

1.3 AIMS AND OBJECTIVES

In this research, we aim to use an algorithm that can be used for fall detection. Having an algorithm that can be used for fall detection is necessary for a fall detector. This algorithm must meet certain requirements. If the algorithm has too many false negatives , i.e. if the action of a person is classified as a non-fall while a fall occurred, no corresponding help is coming for the user while it is needed. When the detector has too many false positives, i.e. the action of a person is classified as a fall while no fall occurred, we have many unnecessary calls for help from the detector, alarming emergency centres falsely. Therefore, we aim to reduce both types of error, while simultaneously be able to detect falls and non-falls. If these requirements are met, the algorithm can be used for fall detection. With this aim in mind, our first research question is:

- Can the proposed algorithms be used for fall detection, taking false negatives and false positives into account?

For this research, we use two types of algorithms. Algorithms using handcrafted features and a deep learning algorithm. The algorithms using handcrafted features are referred to as traditional algorithms in this research. These algorithms were traditionally used in computer vision. In current research, deep learning algorithms often outperform traditional algorithms. In Chapter 2.2, we expand on this topic. Using this finding, we construct our second research question:

- Does the deep learning algorithm outperform the traditional algorithms when detecting falls with a wearable camera?

Further, we aim to compare results between cases when the camera is mounted to the neck to cases when the camera is mounted to the waist. Knowing which mounting position corresponds to the best results has important consequences. Different mounting points lead to different device blueprints. For instance, when mounting the device on the neck yields better performance, we would opt to attach the device to the neck. Consequently, we would use a strap specifically designed for the neck. Therefore, it is interesting to know which camera mounting point is optimal and we construct our third research question as follows:

- Is there a difference in performance of the classification algorithms for fall detection when mounting the camera to the neck or waist?

Besides the fall events, we have non-falling events, such as running, sitting, etc. Another point of interest is to discover how well the selected algorithms perform when classifying these different events, using videos filmed from an egocentric view. Knowing the performance of the selected algorithms provides us valuable information for the future when classifying other events using egocentric videos. Therefore, our fourth research question is:

- How do the algorithms compare in terms of performance when taking into account the different classes of the data set?

1.4 PROPOSED SOLUTION

To achieve the objectives of this research, we obtain videos, filmed by volunteers. We preprocess these videos manually. Some of the videos have extreme lighting in the videos. Further, all videos finish with a hand placed on the camera to indicate an ending of the action. We remove frames from the videos having these properties. Furthermore, we aim to have a similar number of frames for each type of event so that the classifier is unable to distinguish different events merely by frame length.

Having preprocessed the videos, we use three different algorithms. Two traditional algorithms and one deep learning algorithm. We choose to use both types of algorithms, as both have their advantages and disadvantages. Rodriguez-Moreno et al. [99] identify that advantages of traditional algorithms are that these algorithms do not need large amounts of data for training. Further, these algorithms are simple to understand and visualize. Moreover, the features used are explicitly known. The disadvantages of these algorithms are that the features are usually not robust and computation of the features can be computationally intensive. The advantages of deep learning algorithms are that no need of expert knowledge of the features is needed, and features are automatically learned. Moreover, the networks can extract high-level representation in deep layers, making it more suitable for complex tasks. Disadvantages of deep learning algorithms are that often a lot of data is needed, learning is time-consuming, and the models can have problems with generalization of the data.

The traditional algorithms are executed by computing Fisher vectors, obtained from both dense trajectories, as described by Wang et al. [119] and improved dense trajectories, as described by Wang and Schmid [118]. Both

algorithms extract trajectories and compute static and temporal features, each in its own fashion. These features are subsequently Fisher encoded to create Fisher vectors. Next, a SVM classifier is trained using this data to detect falls. We denote the algorithm employing dense trajectory features as DT and denote the algorithm employing improved dense trajectories as IDT.

For deep learning, we make use of CoViAR, described by Wu et al. [125]. This algorithm makes use of the compressed videos and feeds them into three separate CNNs, training each CNN on the data it is being fed. Having trained each CNN, late fusion is applied to combine the CNNs. The scores obtained from the combined CNNs are used to classify the videos.

All three algorithms make use of hyperparameters. We optimize the hyperparameters of the algorithms to increase performance of the classifiers. This is done with k-fold cross validation. Having selected the optimal hyperparameters, we train and predict on the data sets to obtain the final results.

1.5 SCOPE OF THE THESIS

The scope of this research is to determine how the chosen algorithms perform under different class settings and how it performs when using different camera mounting points. For instance, one of the class settings is binary, where we classify a fall against a non-fall. In the binary setting, we compare the results of using different mounting points. Another setting is that we take into account multiple classes. To determine the performance of the algorithm, we work as described in Section 1.3.

1.6 THESIS STRUCTURE

The thesis is structured as follows. In Chapter 2 we explain the background of fall detection using wearable cameras. Further, the corresponding literature review is described. In Chapter 3, we give a detailed overview of the algorithms used for classification. In Chapter 4, the data set is described, the experimental design is explained and the results are shown. Results are discussed in Chapter 5 and compared with other studies. The conclusion of this research is discussed in Chapter 6.

2

BACKGROUND AND LITERATURE REVIEW

In this chapter we describe the background of research conducted on fall detection. We describe the journey from fall detection using body movements towards fall detection using wearable cameras. We further give a detailed overview of research conducted on fall detection with wearable cameras in the literature review, along with algorithms used for action recognition. We finish this chapter with selecting the algorithms and describing what our research adds to the literature.

2.1 BACKGROUND

One of the first researches conducted on fall detection was performed by Kroonenberg et al. [58]. This research was conducted by measuring the body movements of subjects after instructing them to fall onto a gymnasium mattress. Motivated by this study, Wu et al. [126] distinguished fall activities from normal activities by velocity characteristics. These velocity characteristics were measured by placing three markers on the posterior side of the trunk. The marker movements were measured by three cameras. Using the marker movements, velocity in the vertical direction and horizontal direction were calculated. It was found that velocity characteristics are higher for falls than normal movements. Furthermore, it was found that velocity characteristics at various locations of the body are different for different types of falls. Using these findings, many papers emerged for fall detection using either cameras or other sensors to predict falls. Other sensors often used are accelerometers and gyroscopes [4, 14, 54]. Using cameras, among others, Nait-Charif and McKenna [73] and Roughier et al. [100] tracked head movements to detect falls.

Different falls have different body movements. Further, different types of falls have been defined in various researches. El-Bendary et al. [8] defined three types of falls: forward, lateral and backward falls. Putra et al. [96] used the categories forward, backward, left-side, right-side, blinded-forward, and blinded backward fall. Chen et al. [19] grouped the types of falls in the categories: fall lateral left lie on the floor, fall lateral left and sit up from floor, fall lateral right and lie on the floor, fall lateral left and left sit up from the floor, fall forward and lie on the floor, and fall backward and lie on the floor. In this research, different types of falls are used as well.

Using wearable sensors, one has to place the sensors somewhere on the body. Kangas et al. [54] attempted to find the best mounting point for accelerometers by comparing multiple fall detection algorithms. In this research, the accelerometers were placed on the waist, head or wrist. It was found that fall detection using waist or head worn accelerometers are both efficient. The authors further noted that it is preferred to attach the accelerometer to the waist, as it has less limitations concerning usability and acceptance, compared to attaching the accelerometer to the head.

Numerous papers have been written on fall detection using fixed cameras

and other sensors. For an extensive overview, we refer to survey papers [24, 45, 72, 90, 123, 124, 128, 135].

Our research conducts experiments on fall detection using videos filmed from wearable cameras. Therefore, we focus on research being conducted using similar experimental settings. To best of our knowledge, [12, 13, 17, 78, 79, 80, 81] have conducted research using similar experimental settings. In the literature review, Section 2.2.1, a detailed description of these papers is given.

We observe that the literature on fall detection using wearable cameras is scarce. Therefore, we zoom out and evaluate research focusing on the recognition of certain actions from an egocentric viewpoint. Within action recognition using egocentric vision, a distinction can be made between two types of recognition.

The first type of recognition is the recognition of actions performed by the user wearing the camera. The actions performed typically consist of an interaction with objects, with the view being focused on the activity. An example of this type of action is cooking, where one interacts with kitchen equipment and the view of the camera is focused on the kitchen. Performance of action recognition algorithms for this type of actions is often compared using the data sets GTEA GAZE, GTEA GAZE+ [32], Kitchen [111], ADL [92], and UTE [61] as benchmark. Research conducted on these data sets both make use of traditional algorithms, such as [63, 107, 129] and deep learning algorithms, such as [68, 69, 106, 112, 136].

The other type of action using an egocentric viewpoint detects motions of the user. This type of action merely focuses on scene changes as different body movements are being performed. An example is a person running. The scene changes while the person is running forward. Moreover, each frame is a little different as a person slightly moves up or down due to the hops made during a run. The data sets HUJI [93], DogCentric [50] are data sets resembling body movements, where DogCentric resembles body movements of dogs. Further, some researchers created their own data sets for inference. Executing their method on the HUJI data set, both traditional algorithms [93] and deep learning algorithms [1, 2, 94, 122] are assessed. More research has been performed on DogCentric. For instance, [71, 82, 132] use traditional algorithms and [30, 39, 52, 91, 95, 101, 113, 127] use deep learning algorithms. Examples where researchers create their own data set for inference are [110, 131, 133]. These researches all used traditional algorithms. For a more thorough overview of research being conducted using egocentric vision, we refer to survey papers [5, 9, 23, 42, 74].

For fall detection using a wearable camera, we are detecting motions using a wearable camera. From the literature on egocentric vision, we observe that research performed on motion detection using wearable cameras is scarce. The aforementioned data sets used for these types of detection are not largely used to test the performance of new algorithms. We further observe that research performed on egocentric data sets made use of algorithms that were successful at recognizing actions from a third-person view. Examples are [18, 63]. Therefore, we select algorithms having the ability to correctly predict actions at a high rate on benchmark data sets for action recognition from

a third-person viewpoint. These benchmark data sets have been explored more thoroughly than the egocentric data sets. The benchmark data sets are often used to compare new algorithms to state-of-the-art algorithms. One of the first benchmark data sets to evaluate algorithms were the KTH data set [104] and Weizmann [38]. Algorithms proposed and evaluated using these data sets often were traditional algorithms, such as [56, 60, 76]. Recognizing that these data sets are not representative of the richness and complexity of real-world action videos, Kuehne et al. [59] created the HMDB-51 data set and Soomro et al. [109] created the UCF-101 data set. Currently, these data sets are often used to assess proposed algorithms in terms of performance. In Section 2.2.2, we describe selected algorithms achieving high classification performance on these data sets.

2.2 LITERATURE REVIEW

In this section, we explore selected literature in more detail, explaining the current literature on fall detection with wearable cameras and give an overview of algorithms used to recognize actions with good performance on the UCF-101 and HMDB-51 data sets. This section is finished by explaining our choice of the selected algorithms.

2.2.1 *Fall detection with a wearable camera*

In the first research conducted on fall detection with wearable cameras, Casares et al. [17] proposed to modify HOG. This was done by employing both gradient orientation and strength histograms to detect abrupt changes. To compute the histograms, an image is divided into 16 cells. Next, the horizontal (dx) and vertical (dy) gradients are computed for every pixel within a cell. These values are used to calculate gradient orientation, $\arctan(\frac{dy}{dx})$ and gradient strength, $\sqrt{dx^2 + dy^2}$. For both histograms, the authors use nine bins to obtain the final feature descriptor. After normalizing these feature descriptors, the dissimilarity distance between the current frame at time t (having measurement vector $\boldsymbol{s}$) and previous frame at time $t-1$ (reference vector $\boldsymbol{r}$) for both the edge strength (ES) and orientation (EO) is computed. This is computed as follows

$$DRS = 1 - \frac{\sum_{i=1}^{N-1}(r_i - \bar{r})(s_i - \bar{s})}{\sqrt{\sum_{i=0}^{N-1}(r_i - \bar{r})^2 \sum_{i=0}^{N-1}(s_i - \bar{s})^2}},$$

where $\bar{r} = \frac{1}{N}\sum_{i=0}^{N-1} r_i$ and $\bar{s} = \frac{1}{N}\sum_{i=0}^{N-1} s_i$. The dissimilarity distance for edge strength and edge orientation is given by D_{ES} and D_{EO} respectively. The authors further cross-correlate the signal using $(D_{ES}D_{EO})^2$. Using these metrics, the authors detect falls and non-falls by comparing the values to a threshold specified by them. This threshold was set to 0.5. When it exceeds the threshold, the detector signals that a fall has occurred. Furthermore, the authors proposed a mechanism that adaptively controls the number of cells to be used for the feature descriptor. Cells are removed by computing the maximum amplitude among the bins within a cell. Next, the mean value and standard deviation of the vector of maximums from the cells in a frame are computed. Finally, the cells being α standard deviations away from the computed mean are removed. The authors further proposed to remove a maximum of eight cells.

To test their algorithm, the authors used $\alpha = 0.5$. 330 videos were collected, filmed from a camera worn on the belt. The data set consists of 110 fall videos, 110 sitting videos and 110 videos where the persons lies down. For fall detection, the true positives and false negatives are obtained. Next, the fall detection rate using $\frac{TP}{TP+FN}$, i.e. recall is computed. The authors obtain 100 true positives and ten false negatives, leading to a fall detection rate of 91%.

Motivated by this research, Ozcan et al. [78, 81] modified the algorithm in [17]. This was done by computing the dissimilarity distances D_{ES} and D_{EO} between frame t and frame $t - \Delta$. Δ is chosen such that the time elapsed between the past and current image is around one second. Furthermore, using the dissimilarity distance, the algorithm detects whether an event occurs by comparing this distance between multiple frames with parameter ρ. If this distance is larger than ρ, an event is detected. Next, the algorithm detects falls by comparing dissimilarity distances between the current frame t and frame $t - \Delta$ to a threshold τ to determine if a fall event occurs. If the algorithm detects an event, but not a fall event, the algorithm determines which event occurs. It does so by computing average optical flow for horizontal and vertical directions over γ consecutive frames. A sitting event is detected when the vertical mean of optical flow vectors is greater than the horizontal mean. If the horizontal mean of optical flow vectors is greater than the vertical mean, the signal is classified as a lying down event.

Their method was tested by attaching a camera to the belt around the waist of subjects. The following class distribution was obtained: 51 falls from standing, 52 falls from sitting, 52 lying down events and 53 sitting down events are recorded. With their proposed methods, the authors obtain a TPR of 82.69% for lying down and 86.79% for sitting down. For falls from standing and falls from sitting, the authors obtain a TPR of 92.15% and 78.84% respectively. They further compare their new method to the method in [17] in terms of specificity and sensitivity, showing that the proposed method outperforms [17] by comparing these metrics.

In a follow-up paper, Ozcan et al. [79] use gradient local binary patterns (GLBP) instead of edge strength to compute the dissimilarity distance. Here, D_{GLBP} was computed instead of D_{EO}. GLBP is computed by checking eight neighboring pixels for each center pixel. A value of 1 or 0 is assigned to a neighboring pixel if the intensity value is greater or less than the center pixel, respectively. Sequences that have a maximum of two transitions (from 0 to 1 or 1 to 0) are kept. The 8-bit sequence is analyzed to find the length of longest consecutive sequence of 1s and the angle of the edge. These values provide the index of the entry of the 7×8 matrix, incremented by the edge strength value. This matrix is filled by visiting each pixel in a cell, and then being normalized, resulting in a 56-dimensional GLBP feature. A block is divided into 16 cells. The concatenated GLBP vector for one frame is therefore of length 16×56. After calculating the GLBP feature for each cell, L2 normalization was applied before concatenation.

Further, data was obtained from the accelerometer. It was determined that in case the magnitude of the 3-axis vector is greater than γ, the algorithm declares the signal as a fall on the accelerometer-based part. To combine

both the camera data and the accelerometer data, output of both data was compared to a threshold τ_f. If both are greater than τ_f, a fall event detection is triggered.

Using the same data as in [78], the authors obtained a sensitivity of 96.36% and a specificity of 92.45%. The authors further found that combining the video data with the accelerometer further improved performance, where performance was based on the sensitivity and false positives.

Continuing from [79], Ozcan and Velipasalar [80] proposed to select the threshold using a relative-entropy-based threshold selection procedure, instead of selecting the threshold empirically. The selected procedure was performed using different Ali-Silvey distance measures. Having tried multiple measures, it was found that the relative-entropy-based approach provided the best overall results in selecting the optimal threshold for fall detection. For a detailed description on the computation of this threshold, we refer to [80]. For this research, data from the accelerometer was not used.

To test their method, the authors obtained a total of 400 videos. 100 videos showed falls from a sitting position, 100 videos showed falls from a standing position, 100 videos showed a person sitting and 100 videos showed a person lying down. The videos were filmed both indoors and outdoors. These videos were captured by mounting the camera to a belt around the waist. The authors did not specify the distribution between videos filmed indoors and outdoors. Experiments were performed for indoor and outdoor fall detection. Results were shown using a ROC curve and a sensitivity-specificity graph. Further, sensitivities were plotted over various thresholds. The plots indicate that the proposed method works better than previously proposed methods. The authors further obtained a fall detection rate of 93.78%. For outdoor experiments, the ROC curve and sensitivity-specificity curve indicate that their proposed method outperforms earlier proposed methods. The authors further obtained a fall detection rate of 89.8% for outdoor falls. Finally, performing five-fold cross validation, the authors obtained a mean sensitivity of 93.77% and mean specificity of 92.44% for all fall events.

Boudouane et al. [12, 13] proposed a fall detection method based on HOG, combined with optical flow. Having computed the HOG features for two consecutive frames, the algorithm computes the dissimilarity distance D between the HOG features in two consecutive frames. Furthermore, having computed optical flow, the horizontal (dx) and vertical (dy) direction is computed. Next, the mean of these two direction ($\bar{dx}$ and $\bar{dy}$) are computed. These means are used to compute the ratio $\frac{\bar{dy}}{\bar{dx}}$. If D is larger than a threshold α, the algorithm calculates the mean of the dissimilarity distance D and $\frac{\bar{dy}}{\bar{dx}}$ for n consecutive frames. If these means are larger than thresholds β and δ simultaneously, the detector signals that a fall is detected. The thresholds α, β and δ are chosen by the authors.

To test their method, the authors placed a camera around the hips using a belt to obtain videos. They obtained videos for the following actions: falling from a 'standing' position; falling from a sitting position; falling from an 'elongated' position; to sit; to lie down; rotating; to pick up an object from the ground. 20 tests, using 14 subjects, have been performed to obtain

videos with these actions. Using the information obtained from HOG for fall detection, the authors obtained a sensitivity of 95% and a specificity of 46.66%. By introducing optical flow to the algorithm, the authors obtained a sensitivity of 81% and a specificity of 68.33%.

2.2.2 *Algorithms used to recognize action in videos*

We explore algorithms that have a good performance on data sets UCF-101 and HMDB-51. UCF-101 and HMDB-51 contain short trimmed videos, each annotated with one action label. UCF-101 contains 13,320 videos, with 101 action categories. HMDB-51 contains 6,766 videos, with 51 action categories. To find algorithms we can use for our research, we use performance of algorithms on these data sets as a first filter for choosing the algorithms for this paper. We describe both traditional algorithms and deep learning algorithms. Next, we describe which algorithms are chosen for this research and outline the reason for this choice.

The research conducted by Wang et al. [119] obtained good performance in 2011 on the data sets used. A traditional approach is applied to construct the algorithm. By densely sampling points from each frame and tracking them using a median filter on a dense optical flow field, dense trajectories are obtained. Using these trajectories, local motion patterns are encoded, which are used as the trajectory feature. Adding the features HOG, HOF and MBH, the feature set is obtained from each video. These are the features used for the DT algorithm. Next, the performance of their approach is evaluated using a standard bag-of-features approach to convert the set of features into a fixed-dimensional vector. This feature vector is subsequently fed to a SVM classifier with a ξ^2 kernel. The authors found that combining all features yield the highest performance in terms of average accuracy, by using, among others, the KTH data set, yielding an accuracy of 90.2%.

Creating Action Bank Feature Vectors from action detectors, Sadanand and Corso [102] attempted to recognize actions. By feeding the resulting feature set to a standard SVM classifier, results are obtained for the, among others, KTH and HMDB-51 data sets. For these data sets, the following accuracies were obtained: 98.2% for KTH and 26.9 percent for HMDB-51. These results were better than previous results with different methods.

Modelling motion relationships between isolated local patches and temporal patch trajectories, Jiang et al. [51] aimed at modelling human actions, using motion reference points. Using local patch trajectories, which are computed as in [119], the background is separated from the foreground by clustering the trajectories. To classify, a standard bag-of-features approach is used to convert the set of descriptors into a fixed-dimensional vector. This feature vector is subsequently fed to a SVM classifier having a ξ^2 kernel. Using this approach, the authors obtained an accuracy of 40.7% on the HMDB-51 data set.

By estimating the camera motion, Wang and Schmid [118] attempted to improve the dense trajectories [119]. This was done by matching feature points between frames using SURF descriptors and dense optical flow. These matches are consequently used to robustly estimate a homography with RANSAC. Given the estimated camera motion, trajectories are removed that

are consistent with the camera motion. This estimation is used to cancel out camera motion from optical flow, improving motion-based descriptors, such as HOF and MBH. The resulting descriptors are classified using two methods. The first method makes use of the bag-of-features approach and feeds the resulting feature vector into a SVM with a RBF-ξ^2 kernel. The second method encodes the descriptors into a Fisher vector and feeds the Fisher vector to a linear SVM. The second method is the improved dense trajectories algorithm, denoted as IDT in this paper. Using both approaches, it was found that Fisher encoding combined with the linear SVM outperforms the bag-of-features approach combined with a SVM with a RBF-ξ^2 kernel. Thus, IDT outperformed the first method. With IDT, the authors obtained an average accuracy of 57.2% on the HMDB-51 data set. They further compared the performance of IDT with DT and found that DT has an accuracy of 52.2% for the HMDB-51 data set. The dense trajectories algorithm, abbreviated as DT, combines Fisher encoding with the linear SVM as well. However, DT encodes dense trajectories instead of improved dense trajectories.

Motivated by IDT, Peng et al. [86] proposed to create stacked Fisher vectors as a feature set to be used as input for a classifier. Having computed the descriptors HOF, HOG, MBH and trajectories using improved dense trajectories, these descriptors are first encoded using a Fisher encoder. The resulting Fisher vectors are aggregated within multi-scale subvolumes. The Fisher vectors are only aggregated over M subvolumes, where the number of trajectories is larger than a certain threshold. Next, a max-margin dimensionality reduction algorithm is used and the resulting vector is Fisher encoded, resulting in the final feature vector. To obtain the final results, the authors combined this feature vector with the Fisher vector obtained from regular improved dense trajectories, as done in [118]. Feeding the combined features to a linear SVM classifier, the authors obtained a mean average accuracy of 66.79% for the HMDB-51 data set.

Donahue et al. [27] created a Long-term Recurrent CNN to recognize actions. This method passes visual input through a CNN and subsequently passes it through a long short-term model, which is a recurrence sequence model, to predict actions performed in videos. The authors trained two CNNs, one for the RGB images, and one for optical flow. The CNNs are fused by taking the weighted average of the scores to obtain the predicted label. The authors tested their method on the UCF-101 data set and obtained an accuracy of 87.6% on this data set.

Introducing a 3D CNN, Tran et al. [116] attempted to learn spatiotemporal CNNs for action recognition. This method takes full videos as input instead of frames and does not rely on any preprocessing. Furthermore, this method makes use of 3D convolutions and 3D pooling and is finally used as a feature extractor for data sets. Next, the extracted features are combined with improved dense trajectories and fed into a SVM classifier for action classification. Executing this method on the UCF-101 data set, the authors obtained an accuracy of 90.4%.

Feichtenhofer et al. [34] used a two-stream CNN for video action recognition. The authors proposed to fuse the CNN spatially at the last convolutional layer. Fusing at this layer boosts accuracy, while effectively reducing the number of parameters needed. With the proposed technique, the authors fed

RGB images to the first stream and the second stream received the optical flow as input. Testing their method on the UCF-101 data set, the authors obtained a mean classification accuracy of 92.5%. For the HMDB-51 data set, the authors obtained a mean classification accuracy of 65.4%. Adding IDT features further increased the mean classification accuracy to 93.5% and 69.2% for UCF-101 and HMDB-51 respectively.

Modelling long-range temporal structure using CNNs, Wang et al. [121] attempted to recognize actions in the data sets UCF-101 and HMDB-51. The proposed algorithm combines a sparse temporal sampling strategy and video-level supervision to enable effective learning using the whole action video. This is done using the volume of the videos with the CNNs. With this method, the authors obtained a mean classification accuracy of 94.2% for UCF-101 and a mean classification accuracy of 69.4% for HMDB-51. RGB, optical flow and warped optical flow fields are used as input for the CNNs and the CNNs are combined by applying late fusion.

Carreira and Zisserman [16] introduced a two-stream inflated 3D CNN, based on 2D CNN inflation. Filters and pooling kernels of very deep image classification CNNs are expanded into 3D. With this method, the classifier is able to learn spatio-temporal feature extractors from videos while leveraging ImageNet [26] architecture designs. The input used for the streams are RGB frames and optical flows and the average scores between the two streams are used to obtain the final prediction. With this method, the authors obtained an accuracy of 97.9% on UCF-101 and an accuracy of 80.2% on HMDB-51.

Exploiting the compressed representation of videos, Wu et al. [125] recognized actions by leveraging the fact that successive frames are often very similar. The compression technique retains only a few frames completely and reconstructs other frames based on offsets. These other frames are called motion vectors and residuals, obtained from the complete images. Three CNNs are trained on the motion vectors, residuals, and a small number of complete images. This is the CoViAR algorithm. Performing this algorithm on UCF-101 and HMDB-51, accuracies of 90.4% and 59.1% are obtained. Note that these results are not better than aforementioned methods. However, as stated by Wu et al. [125], methods using 3D CNNs result in an explosion of parameters. Furthermore, optical flow does not need to be computed, making it a faster method than previously proposed deep learning methods.

Recognizing that computing optical flow is a time-consuming operation, Zhu et al. [137] proposed to train a CNN, MotionNet. This CNN is capable of recognizing optical flow by treating the optical flow estimation as an image reconstruction problem. Having trained this CNN, optical flow is computed using the MotionNet. Next, the output from the MotionNet is used as input for the temporal stream CNN. Thus, to obtain output from the temporal stream CNN, RGB frames are sufficient, speeding up the process for classification. For classification, two streams are used. The first one is the spatial stream CNN, modelling the frames directly. The second stream is the MotionNet, combined with the temporal stream CNN. Both streams take as input the RGB frames and can be trained independently. Having computed the scores from both streams, the streams are combined by applying late fusion to predict the class of an action. Applying this method on the UCF-101 data set, the authors obtained an accuracy of 89.82%. Using a different

architecture than initially proposed and using the trained MotionNet instead of optical flow, the authors obtain better performances. Using the TSN architecture [121], accuracies of 93.2% and 66.8% for the data sets UCF-101 and HMDB-51 were obtained. Using the architecture of I3D, as proposed in [16], the authors obtained accuracies of 97.1% and 78.7% for the data sets UCF-101 and HMDB-51 respectively.

Another research avoiding the computation of optical flow was conducted by Crasto et al. [20]. In this paper, the authors trained a standard 3D CNN, operating on RGB frames, mimicking the motion stream. To do so, the feature-based loss compared to the flow stream is minimized, producing the motion stream with high fidelity. Further, by leveraging both appearance and motion information, a linear combination of the feature-based loss and the standard cross-entropy loss for action recognition is used. This algorithm is called the Motion Augmented RGB Stream (MARS) algorithm. With this algorithm, the authors obtained an accuracy of 98.1% on the UCF-101 data set and accuracy of 80.9% on the HMDB-51 data set.

The current state-of-the-art algorithm performing best on the data sets UCF-101 and HMDB-51 is obtained in the research conducted by Kalfaoglu et al. [53]. These results were obtained by combining 3D convolution with late temporal modeling. This combination is created by replacing the conventional layer at the end of the 3D convolutional architecture with the Bidirectional Encoder Representations from Transformers (BERT) layer. Using the R(2+1)D architecture [115] with this strategy, the authors obtained an accuracy of 85.10% for the HMDB-51 data set and accuracy of 98.69% for the UCF-101 data set.

For this research, we use traditional algorithms and a deep learning algorithm. The selected traditional algorithms are DT [119] and IDT [118]. These algorithms are selected since they have good performance on the benchmark data sets. Further, these algorithms create trajectories, which resemble the motions in the video. The trajectories and their descriptors are useful for our data set as well, since our data set is based on motion detection. The selected traditional algorithms do not have the best performance among the traditional algorithms shown in the literature review. Nonetheless, we decide to use the selected algorithms, as we recognize that traditional algorithms with better performance are often build upon the improved dense trajectories. This leads to a more computationally intensive task, taking more time to obtain results and classify unseen data. We therefore decide to make a trade-off between time to compute the results and a possible performance improvement. Further, note that we both select DT and IDT. Since IDT attempts to remove camera motion by rectifying images, and removing camera motion can be ineffective to estimate global motion for first-person view [1], we use DT as well. DT does not remove camera motion. Furthermore, in researches where DT and IDT are compared, along with other proposed algorithms, we observe that IDT often outperforms DT. This is shown in [118]. However, in [63, 68] DT outperforms IDT in some occasions. This shows us that IDT does not always outperform DT. Thus, there is no clear consensus whether using IDT is always better than using DT.

For the selection of the deep learning algorithm, we choose to train a CNN that is capable of using only RGB frames as input, and is not computationally

demanding to train, such as 3D CNNs, due to the fact that 3D CNNs yield an explosion of parameters [125]. Therefore, given the current deep learning algorithms, we could choose between the hidden two-stream CNN, using a self-trained MotionNet for optical flow approximation [137] or CoViAR [125]. As observed, the hidden two-stream CNN has better performance when combining the algorithm with TSN [121], which is also a 2D CNN. However, we recognize that training a MotionNet and subsequently train the hidden two-stream CNN is more computationally intensive than training the CNNs of CoViAR. We thus make a trade-off between computational demand of the algorithm and a possible improvement in performance for deep learning by selecting CoViAR. Further, CoViAR can be used for our data set, since it models both the temporal information and spatial information, using multiple CNNs. The temporal information can be used to model the motions being made during an action. The spatial information is useful, as it gives information at which direction a person looks when an action is finished. For instance, the CNN modelling the spatial structure could learn that if the person falls, the person looks at the sky after the fall occurred. This in turn strengthens the overall performance of the classifier as this information is not ignored.

Finally, we note that the results of the deep learning algorithms are often better than the traditional algorithms on the selected data sets. With this in mind, we constructed our research question, investigating whether CoViAR outperforms DT and IDT for fall detection.

2.3 ADDITION TO THE LITERATURE

Investigating the current literature on fall detection with wearable cameras, we observe that performance of deep learning algorithms is missing for this exact type of video data. As the use of deep learning algorithms led to significant results on other computer vision projects, we use a deep learning algorithm to detect falls.

Further, given that there are no deep learning algorithms employed for fall detection with wearable cameras, we directly compare the performance of the deep learning algorithm with the traditional algorithms. Moreover, we make the same comparison for motion detection, as our data set allows to distinguish between multiple motions instead of only distinguishing falls from non-falls.

Finally, research conducted for the optimal placement of the camera for fall detection is absent. We compare the results of the fall detector with the camera mounted to the neck to results where the camera is mounted to the waist, using the proposed algorithms. If there is a difference in results, valuable insight is obtained by knowing where to place a wearable vision-based fall detector. This is important for the design of the fall detector.

3 METHODOLOGY

In this chapter, we give a detailed description of the algorithms DT, IDT and CoViAR.

3.1 TRADITIONAL ALGORITHMS

The traditional algorithms are DT and IDT. We explain how the features for these algorithms are obtained. Further, we explain how the features are encoded and explain the working of SVM, used for classification of the traditional algorithms.

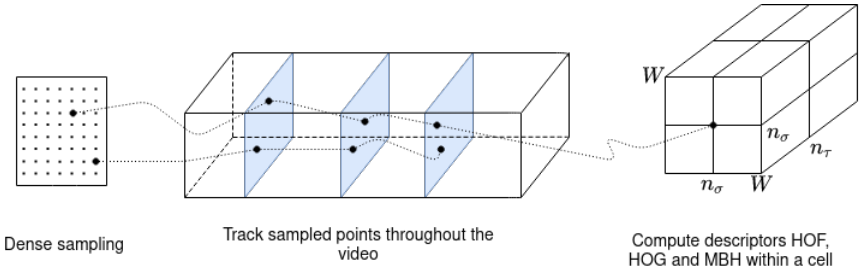

Figure 1: Extraction and characterization of (improved) dense trajectories. Feature points are densely sampled on the grid. Next, tracking is carried out for a specified number of frames by median filtering in a dense optical flow field. Finally, relative point coordinates represent the trajectory shape. Descriptors HOG, HOF and MBH are computed along the trajectory in a $W \times W$ pixels neighborhood, divided into $n_\sigma \times n_\sigma \times n_\tau$ cells [119].

Figure 1 summarizes the process to obtain trajectories and compute corresponding features.

3.1.1 *Dense trajectories*

The first algorithm we describe is DT. This algorithm consists of multiple processing steps. These processing steps are explained in the following sections. To implement this algorithm, OpenCV [15] is used for most of the functionality.

Figure 2: Trajectories of DT during a Fall

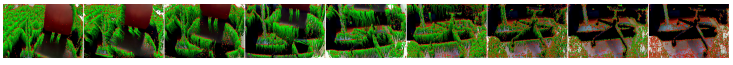

Figure 3: Trajectories of DT during a non-fall

Figures 2 and 3 show trajectories of DT computed during both a fall and a non-fall. Green dots depict the trajectories, red dots depict the densely sampled points.

3.1.1.1 *Dense sampling*

DT starts by reading videos frame by frame. Using the frames, keypoints are obtained by densely sampling points. These keypoints are densely sampled on a grid spaced by W pixels. Following Wang et al. [119], we set $W = 5$. This parameter setting yielded good results in their experiments. The purpose of dense sampling is to track the sampled points through the video. In homogeneous image areas without any structure, it is impossible to track points. This is impossible as it is unknown whether the points in two consecutive frames are exactly the same.

Having sampled the keypoints on the grid, we remove redundant keypoints. Keypoints on the grid are removed using the corner detector introduced by Shi and Tomasi [105]. The corner detector moves a window over each pixel. Suppose we move the window over the area (x, y) in image I and shift it by (u, v). The algorithm then computes the sum of squared differences between these areas. The sum of squared differences is given by

$$S(x,y) = \sum_x \sum_y w(x,y)(I(x+u, y+v) - I(x,y))^2,$$

with $w(x, y)$ being the window function. Using Taylor expansion and letting I_x and I_y be the partial derivatives of I, we obtain

$$I(x+u, y+v) \approx I(x,y) + I_x(x,y)u + I_y(x,y)v.$$

We can then approximate $S(x, y)$ with

$$S(x,y) \approx \sum_x \sum_y w(x,y)(I_x(x,y)u + I_y(x,y)v)^2.$$

This can be written in matrix form

$$S(x,y) \approx \begin{bmatrix} u & v \end{bmatrix} \boldsymbol{A} \begin{bmatrix} u & v \end{bmatrix}^T,$$

where $\boldsymbol{A}$ is a 2×2 matrix. Now, the eigenvalues λ_1 and λ_2 of $\boldsymbol{A}$ are computed. For each frame I, a threshold is computed

$$T = 0.001 \max_{i \in I} \min\{\lambda_1^{(i)}, \lambda_2^{(i)}\},$$

where $(\lambda_1^{(i)}, \lambda_2^{(i)})$ are the eigenvalues of point i in frame I. Then, each sampled point i in frame I is removed if $\min\{\lambda_1^{(i)}, \lambda_2^{(i)}\} < T$. After removing, we obtain the densely sampled points.

3.1.1.2 *Trajectories*

Having obtained the densely sampled points, trajectories are created. For frame I_t, its dense optical flow field $\boldsymbol{\omega}_t = (u_t, v_t)$ is computed with respect to the next frame I_{t+1}. u_t and v_t are the horizontal and vertical components of the optical flow respectively. Given a point $\boldsymbol{p}_t = (x_t, y_t)$, we have

$$\boldsymbol{p}_{t+1} = (x_{t+1}, y_{t+1}) = (x_t, y_t) + (\boldsymbol{M} * \boldsymbol{\omega}_t)|_{(x_t, y_t)},$$

with $\boldsymbol{M}$ being the median filtering kernel. This filter takes all neighborhood pixels together with the selected pixel and returns the median of this set of pixel values. The size of the median filter kernel $\boldsymbol{M}$ is 3×3 pixels. Following Wang et al. [119], a median filter is used, since it is more robust to outliers than bilinear interpolation. This improves trajectories for points at motion boundaries that would otherwise be smoothed out. The motion boundaries represent changes in the flow field.

Since the dense optical flow field is computed, points can be tracked densely without additional cost. Moreover, dense optical flow has smoothness constraints. These constraints allow for robust tracking of fast and irregular motion patterns. Following Wang et al., we use the algorithm of Färneback to extract dense optical flow fields. This algorithm is explained in Section 3.1.1.4.

To form trajectories $(\boldsymbol{p}_t, \boldsymbol{p}_{t+1}, \boldsymbol{p}_{t+2}, ...)$, points of subsequent frames are concatenated. As trajectories tend to drift from their initial locations during the tracking process, the length is limited to L frames. Following Wang et al. [119], we set $L = 15$. Taking into account the computational cost, we do not apply hyperparameter optimization on this parameter. It is possible that the algorithm does not find a tracked point in $W \times W$ neighborhood for a frame. If this is the case, a new point is sampled. This point is added to the tracking process so that a dense coverage of the video is ensured.

Having obtained the trajectories, some trajectories are pruned. Static trajectories do not contain motion information and are therefore pruned. A trajectory is defined to be static if it has a very small variation. Further, we remove trajectories with sudden large displacements. Such trajectories are detected if the displacement vector between two consecutive frames is larger than 70% of the overall displacement of the trajectory.

The shape of these trajectories encode local motion patterns. Given a trajectory of length L, the shape by a sequence $(\Delta\boldsymbol{p}_t, ..., \Delta\boldsymbol{p}_{t+L-1})$ of displacement vectors $\Delta\boldsymbol{p}_t = \boldsymbol{p}_{t+1} - \boldsymbol{p}_t = (x_{t+1} - x_t, y_{t+1} - y_t)$ is described. The resulting vector is normalized by the sum of displacement vector magnitudes. This results in the trajectory $\boldsymbol{\tau}$, with

$$\boldsymbol{\tau} = \frac{(\Delta\boldsymbol{p}_t, ..., \Delta\boldsymbol{p}_{t+L-1})}{\sum_{j=t}^{t+L-1} ||\Delta\boldsymbol{p}_j||}.$$

As the trajectory has a fixed length of 15 frames, we obtain a 30-dimensional descriptor, 15 for the horizontal direction and 15 for the vertical direction.

3.1.1.3 *Oriented gradients*

To capture static appearance information, we make use of the histogram of gradients (HOG). This concept became widely known after the research conducted by Dalal and Triggs [21]. HOG makes use of the oriented gradients. For each point obtained from dense sampling, we compute its gradient. The result of this computation is the oriented gradient. This is computed using the Sobel operator, first described by Sobel and Feldman [108]. This is an operator computing the approximation of the gradient of the image intensity

function. The gradient is approximated by computing the x-gradient by filtering the current point with its neighbors using the filter

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix},$$

and the y-gradient by filtering the current point with its neighbors using the filter

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}.$$

Having obtained the gradients for each sampled point, we compute HOG. Computation of the histogram is described in Section 3.1.1.6.

3.1.1.4 *Optical flow*

To extract the dense optical flow fields and the optical flow, used for the histogram of optical flow (HOF), the algorithm by Farnebäck [31] is used. This algorithm is a two-frame motion estimation algorithm. It approximates each neighborhood of both frames using quadratic polynomials. In this research, we use a neighborhood size of ten. The approximation is made by using the polynomial expansion transform.

The polynomial expansion is performed by approximating each pixel with a polynomial. Since every pixel is being taken into account, this is a dense optical flow algorithm. The algorithm starts by considering the quadratic polynomial for the signal f_1 of the first frame

$$f_1(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{A}_1 \boldsymbol{x} + \boldsymbol{b}_1^T \boldsymbol{x} + c_1,$$

where we let $\boldsymbol{x}$ be the coordinates (x, y) of the pixels. Further, a new signal for the second frame, f_2 is constructed by a global displacement $\boldsymbol{d}$. The global displacement is given by $\boldsymbol{d} = (u, v)$. u is the displacement in the horizontal direction and v is the displacement in the vertical direction. Next,

$$\begin{aligned} f_2(\boldsymbol{x}) =& f_1(\boldsymbol{x} - \boldsymbol{d}) = (\boldsymbol{x} - \boldsymbol{d})^T \boldsymbol{A}_1 (\boldsymbol{x} - \boldsymbol{d}) + \boldsymbol{b}_1^T (\boldsymbol{x} - \boldsymbol{d}) + c_1 \\ &= \boldsymbol{x}^T \boldsymbol{A}_1 \boldsymbol{x} + (\boldsymbol{b}_1 - 2\boldsymbol{A}_1 \boldsymbol{d})^T \boldsymbol{x} + \boldsymbol{d}^T \boldsymbol{A}_1 \boldsymbol{d} - \boldsymbol{b}_1^T \boldsymbol{d} + c_1 \\ &= \boldsymbol{x}^T \boldsymbol{A}_2 \boldsymbol{x} + \boldsymbol{b}_2^T \boldsymbol{x} + c_2. \end{aligned}$$

Letting $f_1(\boldsymbol{x}) = f_2(\boldsymbol{x})$, we get

$$\begin{aligned} \boldsymbol{A}_2 &= \boldsymbol{A}_1 \\ \boldsymbol{b}_2 &= \boldsymbol{b}_1 - 2\boldsymbol{A}_1 \boldsymbol{d} \\ c_2 &= \boldsymbol{d}^T \boldsymbol{A}_1 \boldsymbol{d} - \boldsymbol{b}_1^T \boldsymbol{d} + c_1. \end{aligned}$$

Now solving for the translation $\boldsymbol{d}$, which is the variable of interest, we obtain

$$2\boldsymbol{A}_1 \boldsymbol{d} = -(\boldsymbol{b}_2 - \boldsymbol{b}_1) \Leftrightarrow \boldsymbol{d} = -\frac{1}{2} \boldsymbol{A}_1^{-1} (\boldsymbol{b}_2 - \boldsymbol{b}_1).$$

This holds if $\boldsymbol{A}_1$ is nonsingular.

Is is unrealistic to assume that an entire signal is a single polynomial and

that the global translation is related to two signals. Therefore, Farnebäck suggested to replace the global polynomial with local polynomial approximations. Polynomial expansion of both images is performed. This yields expansion coefficients $\boldsymbol{A}_1(\boldsymbol{x})$, $\boldsymbol{b}_1(\boldsymbol{x})$ and $c_1(\boldsymbol{x})$ for frame 1 and $\boldsymbol{A}_2(\boldsymbol{x})$, $\boldsymbol{b}_2(\boldsymbol{x})$ and $c_2(\boldsymbol{x})$ for frame 2. Next, the approximation

$$\boldsymbol{A}(\boldsymbol{x}) = \frac{\boldsymbol{A}_1(\boldsymbol{x}) + \boldsymbol{A}_2(\boldsymbol{x})}{2} \tag{1}$$

is obtained.

$$\Delta\boldsymbol{b}(\boldsymbol{x}) = -\frac{1}{2}(\boldsymbol{b}_2(\boldsymbol{x}) - \boldsymbol{b}_1(\boldsymbol{x})) \tag{2}$$

is introduced to obtain the constraint

$$\boldsymbol{A}(\boldsymbol{x})\boldsymbol{d}(\boldsymbol{x}) = \Delta\boldsymbol{b}(\boldsymbol{x}).$$

Here, $\boldsymbol{d}(\boldsymbol{x})$ indicates that the global replacement $\boldsymbol{d}$ is replaced with a spatially varying displacement field.

$\boldsymbol{d}(\boldsymbol{x})$ is found using weighted least squares, i.e. minimizing:

$$\sum_{\Delta\boldsymbol{x}\in I} w(\Delta\boldsymbol{x})||\boldsymbol{A}(\boldsymbol{x}+\Delta\boldsymbol{x})\boldsymbol{d}(\boldsymbol{x}) - \Delta\boldsymbol{b}(\boldsymbol{x}+\Delta\boldsymbol{x})||^2.$$

$w(\Delta\boldsymbol{x})$ is a weight function for the points in the neighborhood. In our case, the weight function is the Gaussian weight function. We use a standard deviation of 1.5, as used by Farnebäck.

Solving this optimization problem, we find that the minimum is obtained for:

$$\boldsymbol{d}(\boldsymbol{x}) = \left(\sum w\boldsymbol{A}^T\boldsymbol{A}\right)^{-1}\sum w\boldsymbol{A}^T\Delta\boldsymbol{b},$$

where the indices for w, $\boldsymbol{A}$ and $\Delta\boldsymbol{b}$ are dropped for readability. The minimum value of the problem is given by

$$e(\boldsymbol{x}) = \left(\sum w\Delta\boldsymbol{b}^T\Delta\boldsymbol{b}\right) - \boldsymbol{d}(\boldsymbol{x})^T\sum w\boldsymbol{A}^T\Delta\boldsymbol{b}.$$

This means that $\boldsymbol{A}^T\boldsymbol{A}$, $\boldsymbol{A}^T\Delta\boldsymbol{b}$ and $\Delta\boldsymbol{b}^T\Delta\boldsymbol{b}$ are computed pointwise and averaged with w before being solved for displacement.

Furthermore, Farnebäck points out that robustness of the algorithm can be improved. The algorithm can be improved by parameterizing the displacement field according to some motion model. Farnebäck proposes to use a motion model which is linear in the parameters. He uses the eight parameter model. The eight parameter model in 2D is given by

$$\begin{aligned} d_x(x,y) &= a_1 + a_2x + a_3y + a_7x^2 + a_8xy, \\ d_y(x,y) &= a_4 + a_5x + a_6y + a_7xy + a_8y^2 \end{aligned}$$

This can be rewritten to the system $\boldsymbol{d} = \boldsymbol{S}\boldsymbol{p}$, with

$$\begin{aligned} \boldsymbol{S} &= \begin{bmatrix} 1 & x & y & 0 & 0 & 0 & x^2 & xy \\ 0 & 0 & 0 & 1 & x & y & xy & y^2 \end{bmatrix} \\ \boldsymbol{p} &= \begin{bmatrix} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 \end{bmatrix}^T, \end{aligned}$$

which yields the following weighted least squares problem

$$\sum_i w_i ||\boldsymbol{A}_i \boldsymbol{S}_i \boldsymbol{p} - \Delta \boldsymbol{b}_i||^2.$$

Its solution is given by

$$\boldsymbol{p} = \left(\sum_i w_i \boldsymbol{S}_i^T A_i^T \boldsymbol{A}_i \boldsymbol{S}_i \right)^{-1} \sum_i w_i \boldsymbol{S}_i^T \boldsymbol{A}_i^T \Delta \boldsymbol{b}_i$$

which is the model that we use as well.

Farnebäck further notes that since polynomial expansions are local models, the displacements will vary spatially. This introduces errors in the constraint $\boldsymbol{A}(\boldsymbol{x})\boldsymbol{d}(\boldsymbol{x}) = \Delta \boldsymbol{b}(\boldsymbol{x})$. With large displacements, errors are large. A solution is to use a priori knowledge about the displacement field. Using the a priori knowledge, the polynomial at $\boldsymbol{x}$ in the first signal is compared to the polynomial at $\boldsymbol{x} + \tilde{\boldsymbol{d}}(\boldsymbol{x})$ of the second signal. Here, we have that $\tilde{\boldsymbol{d}}(\boldsymbol{x})$ is the a priori displacement field. Replacing Equations (1) and (2) with

$$\boldsymbol{A}(\boldsymbol{x}) = \frac{\boldsymbol{A}_1(\boldsymbol{x}) + \boldsymbol{A}_2(\tilde{\boldsymbol{x}})}{2}$$

and

$$\Delta \boldsymbol{b}(\boldsymbol{x}) = -\frac{1}{2}(\boldsymbol{b}_2(\tilde{\boldsymbol{x}}) - \boldsymbol{b}_1(\boldsymbol{x})) + \boldsymbol{A}(\boldsymbol{x})\tilde{\boldsymbol{d}}(\boldsymbol{x}),$$

where

$$\tilde{\boldsymbol{x}} = \boldsymbol{x} + \tilde{\boldsymbol{d}}(\boldsymbol{x}).$$

To find a solution using a priori knowledge, estimated displacements are used as a priori displacement in the next iteration to estimate the updated displacements. For this research, we use two iterations.

3.1.1.5 *Motion boundaries*

Having obtained the optical flow, we obtain the horizontal and vertical components of the optical flow. The derivatives of these components are the motion boundaries, proposed by Dalal et al. [22]. The motion boundaries represent relative motion between the pixels. Using motion boundaries, locally constant camera motion is removed and information about changes in the flow field is kept. Motion boundaries are more robust to camera motion than optical flow[120]. We let u denote the image containing the x (horizontal) component of optical flow and let v denote the image containing the y (vertical) component of optical flow. Further, let u_x, u_y, v_x and v_y denote the corresponding x- and y-gradient differential flow images. For instance,

$$u_y = \frac{\partial}{\partial y} u$$

is the y-gradient of the x component of optical flow. We compute these gradients using the Sobel operator. These flow images are the motion boundaries and quantize the derivative of the optical flow in horizontal and vertical direction. Having obtained the motion boundaries, we compute MBH. This is computed in the same way as HOG and HOF are computed.

3.1.1.6 *Obtaining HOG, HOF and MBH*

After obtaining the descriptors (oriented gradients, optical flow and motion boundaries), we compute corresponding histograms to reduce the dimensionality of the descriptors.

Following Wang et al. [119], we use eight bins for HOG and MBH. We further use nine bins for HOF. From each descriptor, we obtain the horizontal direction g_x and vertical direction g_y. Using these directions, we compute the magnitude and angle with the formulas

$$g = \sqrt{g_x^2 + g_y^2}$$

and

$$\theta = \arctan \frac{g_y}{g_x}.$$

After the computation of the magnitude and angle, we perform an extra operation for HOF. In this operation, we check whether there are optical flow magnitudes that are lower than a specified threshold. This threshold is set to 0.4. If this is the case, we add the magnitude to the zero bin of HOF. This zero bin accounts for optical flows having a magnitude which is lower than a threshold. Having performed this operation for HOF, we continue as we do with HOG and MBH.

Next, the algorithm checks which bin corresponds to the computed magnitude. Each bin is represented by a corresponding angle. We use eight bins. Each bin corresponds to an angle interval. With eight bins, bin i has the angle interval $[22.5 \cdot i, 22.5 \cdot (i+1)$ for $i = 0, ..., 7$. Each bin thus covers $\frac{180}{8} = 22.5$ degrees. The center of bin i is given by $22.5 \cdot (i + 0.5)$. Note that for the bin coverage, we use 180 in the numerator instead of 360 because we use unsigned gradients. To describe the process of creating the histogram, suppose that we obtain a magnitude of 2 and an angle of 12.25 degrees. Then, 2 is added to the first bin. Now, suppose that we obtain an angle of 22.5 degrees and a magnitude of 2. Then, the magnitude is shared proportionally between the first and second bin. $\frac{37.5-22.5}{22.5} \cdot 2$ is added to the first bin and $\frac{22.5-12.25}{22.5} \cdot 2$ is added to the second bin.

After obtaining the histograms, we normalize each histogram with the L_2 norm. Letting vector $\boldsymbol{x}$ describe the descriptor, we thus update element x_i using the transformation

$$\tilde{x}_i = \frac{x_i}{\sqrt{\sum_{j=1}^{n} x_j^2}}.$$

Histograms corresponding to the trajectories are computed over multiple cells. If the trajectory reaches a length of 15, one histogram is computed. This is done by summing up all histograms into a cell into one histogram. The summation is performed by treating each histogram as a vector, where each element consists of a value for the bin. Then, we sum all vectors within a cell into an output vector, having the same dimensions. This is the histogram corresponding to a cell, computed for a trajectory. Each trajectory has $2 \times 2 \times 3$ cells. Having computed the histograms for each cell, we concatenate all histograms by iterating over the cells. Since a histogram

is a vector of bins, we obtain different feature vectors for each trajectory. For HOG, we obtain a $2 \times 2 \times 3 \times 8 = 96$-dimensional vector. We obtain the same vector size for MBH in the horizontal direction and MBH in the vertical direction. Concatenating the horizontal and vertical vector, we obtain a 192-dimensional vector for MBH. As we have nine bins for the HOF, we obtain a $2 \times 2 \times 3 \times 9 = 108$-dimensional vector for HOF.

3.1.2 *Improved dense trajectories*

An adjustment to the dense trajectories is the improved dense trajectories algorithm, proposed by Wang et al. [118]. To compute these trajectories, the algorithm takes into account camera calibration. By taking the camera calibration into account, estimation of optical flow improves. To account for camera calibration, the Speeded up robust features (SURF) method is used to extract keypoints. The keypoints obtained from SURF are matched based on the nearest neighbor rule, using the features from SURF. Further, the densely sampled points are obtained. The method of densely samplings point is similar to that of dense trajectories and is explained in Section 3.1.1.1. The densely sampled points and the keypoints obtained from SURF complement each other. SURF focuses on the blob-type structures and the densely sampled points focus on corners and edges. Combining these approaches results in a more balanced distribution of matched points. This is important for the estimation of the homography, which rectifies images. To compute the descriptors HOG, HOF, MBH and the trajectories, first the homography is estimated using RANSAC. The homography is estimated using feature matches extracted between two consecutive frames. Using the estimated homography, the second frame is warped to the first frame. Then, optical flow is re-computed between the first and warped second frame, yielding the warped optical flow. Finally, HOF, MBH and the improved trajectories are obtained in the same manner as described in Sections 3.1.1.2, 3.1.1.4 and 3.1.1.5, but now using the warped optical flow. HOG is obtained similarly as for DT, as HOG is a static feature. In the following sections, we describe the adjustments that are made to create the descriptors of IDT.

Figure 4: Trajectories of IDT during a fall

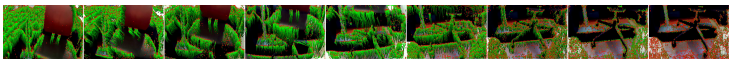

Figure 5: Trajectories of IDT during a non-fall

Figures 4 and 5 show trajectories of IDT computed during both a fall and a non-fall. Green dots depict the trajectories, red dots depict the densely sampled points.

3.1.2.1 *Speeded Up Robust Features*

The SURF algorithm is used for camera calibration. SURF, proposed by Bay et al. [6, 7], is a scale- and rotation-invariant keypoint detector and keypoint descriptor. The detector is based on approximating the Hessian matrix, relying on integral images to reduce the time needed for computation.

The algorithm starts with computing the integral image (or summed-area table). An element of the integral image $I_\Sigma(x, y)$ is given by the sum of all pixels in the input image I of a rectangular region formed by the point (x, y) and the origin. It is given by

$$I_\Sigma(x, y) = \sum_{i=0}^{i \leq x} \sum_{j=0}^{j \leq y} I(i, j).$$

Given point (x, y) in image I, the Hessian matrix $\boldsymbol{H}(x, y, \sigma)$ at point (x, y) at scale σ is defined as

$$\boldsymbol{H}(x, y, \sigma) = \begin{bmatrix} L_{xx}(x, y, \sigma) & L_{xy}(x, y, \sigma) \\ L_{xy}(x, y, \sigma) & L_{yy}(x, y, \sigma) \end{bmatrix},$$

where $L_{xx}(x, y, \sigma)$ is the convolution of the Gaussian second order derivative $\frac{\partial^2}{\partial x^2} g(\sigma)$. Approximations for the Gaussian second order derivatives are computed. These approximations are given by D_{xx}, D_{xy}, D_{yx} and D_{yy} and are computed using box filters. The determinant of the approximated Hessian is given by

$$det(H) = D_{xx} D_{yy} - (w D_{xy})^2.$$

Using OpenCV, we have that $w = 0.81$. w is used as a measure of local change around the point. The points with maximum determinants are chosen as possible keypoints.

Keypoints are found at different scales. These scale spaces are implemented as an image pyramid. The images in these scales are repeatedly smoothed with a Gaussian and then sub-sampled to achieve a higher level of the pyramid. Since box filters and integral images are used, applying the same filter to the output of a previously filtered layer is not needed. Instead, box filters of any size are applied to the image.

Scale space is analysed by up-scaling the filter size. The output of the 9×9 box filter is considered as the initial scale layer, referred to as scale $s = 1.2$. Next, larger masks are used to obtain following layers. Doing so, we obtain layers with filter size 9×9, 15×15, 21×21 and 27×27 to filter the image. The ratios of the filters remain constant after scaling. Therefore, the approximated Gaussian derivatives scale accordingly. For example, the filter of size 27×27 corresponds to $\sigma = 1.2 \times 3 = 3.6$.

After scaling, non-maximum suppression in a $3 \times 3 \times 3$ neighborhood is applied. The first two dimensions are the neighboring pixels in its own scale and the third dimension is the dimension in the lower and upper scale. We check if a pixel in the neighborhood is the maximum for the current scale, lower scale and upper scale. The possible keypoint is selected as final keypoint if it exceeds a certain threshold τ. We set $\tau = 200$.

To ensure that SURF is invariant to image rotation, the orientation of keypoints is identified. The Haar wavelet responses in horizontal and vertical direction are computed within a circular neighborhood of radius $6s$ around the keypoint. The sampling step is s. The size of the wavelets are chosen to be $4s$. Using the integral image, only six operations are needed to compute

the response in horizontal or vertical direction at any scale. After computing the wavelet responses, the responses are weighted using a Gaussian, centred at the keypoint. The weighted responses are represented as points, with horizontal responses corresponding to the horizontal axis and vertical responses corresponding to the vertical axis of a graph. Next, dominant orientation is estimated by calculating the sum of all responses within a sliding window of size $\frac{\pi}{3}$. These two summed responses yield a local orientation vector. After sliding all windows and computing orientation vectors for each window, the orientation corresponding to the longest orientation vector is chosen. After finding the orientation, the descriptor of a keypoint can be extracted.

To extract the descriptor corresponding to a keypoint, a square region around the keypoint is constructed. This is oriented along the selected orientation. This region is split into smaller 4×4 regions. In each sub-region, Haar wavelet responses are computed at 5×5 regularly spaced sample points. Let d_x and d_y be the Haar wavelet responses in horizontal and vertical direction, respectively. These responses are weighted with a Gaussian, centred at the keypoint. The responses d_x and d_y are summed over each sub-region, forming two 16-dimensional vectors. Further, the sum of absolute values of the responses, $|d_x|$ and $|d_y|$ are computed. Concatenating d_x, d_y, $|d_x|$ and $|d_y|$ yields the 64-dimensional vector, the descriptor describing the keypoint.

Having obtained the keypoints and corresponding descriptors of two consecutive frames, we start matching the keypoints in both frames. Matches are computed based on the distance between two keypoints, where the distance D between two descriptor vectors is computed using the L2-norm, i.e.

$$D = ||\boldsymbol{x}_1^{(i)} - \boldsymbol{x}_2^{(j)}||^2,$$

with $\boldsymbol{x}_1^{(i)}$ being the descriptor of a keypoint i in frame 1 and $\boldsymbol{x}_2^{(j)}$ being the descriptor of a keypoint j in frame 2. A keypoint i in the first frame is matched with a keypoint j in second frame by comparing distance. Let N_1 (N_2) be the number of keypoints in the first (second) frame. Then, keypoint i of the first frame and keypoint j of the second frame are a match if the distance between $\boldsymbol{x}_1^{(i)}$ and $\boldsymbol{x}_2^{(j)}$ is the minimum for all N_2 keypoints. The algorithm compares distances of all combinations of the N_1 keypoints in frame 1 with the N_2 keypoints in frame 2 to find all matches between the two frames.

3.1.2.2 *Homography and RANSAC*

Having obtained the extracted points from dense sampling and SURF, the frames are rectified using homography. Homography describes the projective geometry of two cameras and a world plane. The homography maps images of points lying on a world plane from one camera view to another. The relationship is projective as it depends on the intersection of planes with lines only. Estimating the homography rectifies the image.

Given all matched points (x_i, y_i) in frame 1 and points (x_i', y_i') in frame

2, we solve the homography $\boldsymbol{p}^{'} = \boldsymbol{H}\boldsymbol{p}$. For one point being matched in two frames, we obtain the system:

$$\begin{bmatrix} wx_i^{'} \\ wy_i^{'} \\ w \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}. \tag{3}$$

We have that $\boldsymbol{p}_i = \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$ and $\boldsymbol{p}_i^{'} = \begin{bmatrix} wx_i^{'} \\ wy_i^{'} \\ w \end{bmatrix}$. We can rewrite Equation (3) to

$$\begin{aligned} x_i^{'} &= \frac{h_{00}x_i + h_{01}y_i + h_{02}}{h_{20}x_i + h_{21}y_i + h_{22}} \\ &\Leftrightarrow x_i^{'}h_{20}x_i + h_{21}y_i + h_{22} = h_{00}x_i + h_{01}y_i + h_{02} \\ y_i^{'} &= \frac{h_{10}x_i + h_{11}y_i + h_{12}}{h_{20}x_i + h_{21}y_i + h_{22}} \\ &\Leftrightarrow y_i^{'}h_{20}x_i + h_{21}y_i + h_{22} = h_{10}x_i + h_{11}y_i + h_{12}, \end{aligned}$$

which can be rewritten to the system

$$\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x_i^{'}x_i & -x_i^{'}y_i & -x_i^{'} \\ 0 & 0 & 0 & x_i & y_i & 1 & -y_i^{'}x_i & -y_i^{'}y_i & -y_i \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Repeating this procedure for all points, we obtain the system

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1^{'}x_1 & -x_1^{'}y_1 & -x_1^{'} \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y_1^{'} & -y_1^{'}y_1 & -y_1^{'} \\ & & & & & \dots & & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -x_n^{'}x_n & -x_n^{'}y_n & -x_n^{'} \\ 0 & 0 & 0 & x_n & y_n & 1 & -y_n^{'}x_n & -y_n^{'}y_n & -y_n^{'} \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

$\Leftrightarrow \boldsymbol{A}\boldsymbol{h} = \boldsymbol{0}$.

Next, we estimate the parameter $\boldsymbol{h}$. To estimate $\boldsymbol{h}$, we can rewrite the matrix form to the the least squares problem

$$\min ||\boldsymbol{A}\boldsymbol{h} - \boldsymbol{0}||^2,$$

which is solved by computing the estimate $\hat{\boldsymbol{h}}$ for $\boldsymbol{h}$. The solution for $\hat{\boldsymbol{h}}$ is given by the eigenvector of $\boldsymbol{A}^T\boldsymbol{A}$ corresponding to the smallest eigenvalue of $\boldsymbol{A}^T\boldsymbol{A}$. We now rebuild the estimation for $\boldsymbol{H}$ by indexation from $\hat{\boldsymbol{h}}$.

Having found the keypoints, we might obtain outliers. These outliers could cause a disturbed homography. To mitigate this problem, Fischler and Bolles

[35] proposed the RANSAC algorithm. RANSAC is used to remove outliers. RANSAC runs for multiple iterations. In each iteration, the following operation is executed:

- Select some pairs of points randomly, with a pair being a match of a point in the first and second frame.
- Compute the homography.
- Find the inliers by computing the distance between the points in the first frame and the points in the warped frame. The distance $||\boldsymbol{p}_i^{'}, \boldsymbol{H}\boldsymbol{p}_i|| < \epsilon$ is computed.
  - ϵ is the error tolerance between deciding whether the point is an inlier or outlier.
  - If the distance is smaller than ϵ, the match is added to the set of inliers.
- Keep the largest set of inliers.

After finding a set with a sufficient number of inliers using the corresponding model, the loop is terminated. $\boldsymbol{h}$ is estimated again, but now using all of the inliers to obtain the final model for the homography.

3.1.3 *Encoding the descriptors and performing classification*

After obtaining the descriptors of either DT or IDT, we encode these descriptors to reduce the number of dimensions. After encoding the video, the video is no longer described by a large matrix consisting of multiple features, obtained from the dense and improved dense trajectories. Instead, each video is described by one vector, the Fisher vector. Encoding is done using PCA, GMM and creating a Fisher vector. After encoding, the resulting features are fed into a linear SVM classifier for training and prediction.

3.1.3.1 *Principal component analysis*

Principal component analysis (PCA) was first described by Pearson [84]. This method is used to extract features of a data set using a statistical procedure. PCA enables us to reduce the number of dimensions of the data set. Given a data set $\boldsymbol{X} \in \mathbb{R}^{n \times p}$, we want to reduce the data from p columns to L columns. To do so, we start by computing the empirical mean for each column $j = 1, ..., p$. These empirical means are added into the vector $\boldsymbol{u}$. These elements of $\boldsymbol{u}$, the empirical means u_j, are computed with the formula

$$u_j = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{X}_{i,j}.$$

Next, we center the data. To center the data, deviations from the mean are computed. We let $\boldsymbol{B}$ be the matrix after centering the data. We then have

$$\boldsymbol{B} = \boldsymbol{X} - \mathbf{1}\boldsymbol{u}^T,$$

with $\mathbf{1} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \end{bmatrix}^T$ being a $n \times 1$ vector. In the following step, the covariance matrix $\boldsymbol{C}$ is computed. We have

$$\boldsymbol{C} = \frac{1}{n-1} \boldsymbol{B}^T \cdot \boldsymbol{B},$$

where we use the centered matrix $\boldsymbol{B}$. Using $\boldsymbol{C}$, we compute the eigenvectors and eigenvalues of $\boldsymbol{C}$, using the equation

$$\boldsymbol{V}^{-1}\boldsymbol{C}\boldsymbol{V} = \boldsymbol{D}.$$

$\boldsymbol{D}$ corresponds to the diagonal matrix of eigenvalues of $\boldsymbol{C}$. $\boldsymbol{V}$ is the matrix of eigenvectors, $\boldsymbol{V} \in \mathbb{R}^{p\times p}$. The matrix of eigenvectors is sorted by the eigenvalues in descending order.

To obtain the final matrix from PCA dimensionality reduction, we use the formula

$$\tilde{\boldsymbol{X}} = \boldsymbol{X} \cdot \boldsymbol{V}_L.$$

$\boldsymbol{V}_L$ is the block matrix of $\boldsymbol{V}$ where the block is taken on all p rows and the first L columns, i.e. $\boldsymbol{V}_L \in \mathbb{R}^{p\times L}$. Further,

$$\boldsymbol{V} = \begin{bmatrix} v_{1,1} & \cdots & v_{1,L} & v_{1,L+1} & \cdots & v_{1,p} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ v_{p,1} & \cdots & v_{p,L} & v_{p,L+1} & \cdots & v_{p,p} \end{bmatrix} = \begin{bmatrix} & v_{1,L+1} & \cdots & v_{1,p} \\ \boldsymbol{V}_L & \vdots & \vdots & \vdots \\ & v_{p,L+1} & \cdots & v_{p,p} \end{bmatrix}.$$

Having obtained the data set after PCA, we note that the transformed matrix $\tilde{\boldsymbol{X}}$ obtained after applying PCA has a diagonal covariance matrix. This is a useful property that can be used for the GMM.

3.1.3.2 *Gaussian mixture model*

To compute the Fisher vectors, we need certain parameters. These parameters are obtained from the Gaussian mixture model (GMM). The parameters are estimated by taking into account a number of data points obtained from the data matrix resulting after PCA.

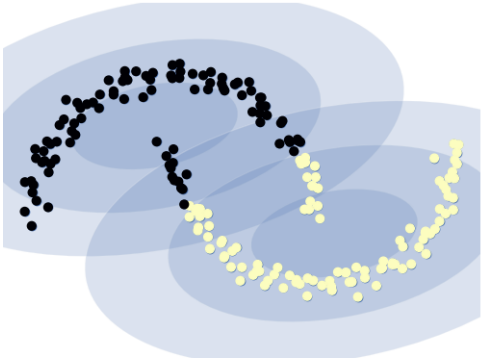

Figure 6: Example of a GMM with 2 clusters, where we observe that clusters can overlap. Data points can be a member of a cluster with a certain probability.

A Gaussian Mixture is a function consisting of several Gaussians. The Gaussian density function is given by

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{\frac{D}{2}}|\boldsymbol{\Sigma}|^{\frac{1}{2}}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}.$$

$\boldsymbol{x}$ is a feature and D is the number of dimensions of each data point. Each Gaussian is identified by cluster k, where $k = 1, .., K$. Each Gaussian cluster has the following parameters:

- $\boldsymbol{\mu}_k$, defining the centre of cluster k.
- $\boldsymbol{\Sigma}_k$, defining the variance of cluster k.
- π_k, being the mixing coefficient π_k, which gives us the proportion of the Gaussian function k.

The mixing coefficients are probabilities, with restriction

$$\sum_{k=1}^{K} \pi_k = 1.$$

Using the GMM, we can find a relation between a data point $\boldsymbol{x}$ and the probability that $\boldsymbol{x}$ belongs to a cluster k. This probability is expressed as $p(z_{nk} = 1|\boldsymbol{x}_n)$, with z being a latent variable, $z \in \{0, 1\}$.

We let $\boldsymbol{z} = \left[z_1, ..., z_K\right]$. Further, it is assumed that each z_i is independent from each other and $z_i = 1$ if and only if k is equal to the cluster point. We thus have:

$$p(\boldsymbol{z}) = p(z_1 = 1)^{z_1} p(z_2 = 1)^{z_2} \cdots p(z_K = 1)^{z_K} = \prod_{k=1}^{K} \pi_k^{z_k}.$$

Now, we have that

$$p(\boldsymbol{x}_n|\boldsymbol{z}) = \prod_{k=1}^{K} \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_k},$$

as $\boldsymbol{x}_n$ is normally distributed. Using the product rules of probabilities, we get:

$$p(\boldsymbol{x}_n, \boldsymbol{z}) = p(\boldsymbol{x}_n|\boldsymbol{z})p(\boldsymbol{z}).$$

Now, we need to find $p(\boldsymbol{x}_n)$, which can be found by applying marginalization. We obtain

$$p(\boldsymbol{x}_n) = \sum_{k=1}^{K} p(\boldsymbol{x}_n|\boldsymbol{z})p(\boldsymbol{z}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

and its joint probability of all observations $\boldsymbol{x}_n$ is given by

$$p(\boldsymbol{X}) = \prod_{n=1}^{N} p(\boldsymbol{x}_n) = \prod_{n=1}^{N} \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k).$$

Taking the log of this function, we obtain:

$$\ln(p(\boldsymbol{X})) = \sum_{n=1}^{N} \ln \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k).$$

Since the derivative of this equation has no analytical root solution, we cannot find the optimal parameters analytically.

To find the optimal parameters, the Expectation-Maximization algorithm is used, introduced by Dempster et al. [25]. To do so, Bayes rule is applied:

$$\begin{aligned} p(z_k = 1|\boldsymbol{x}_n) &= \frac{p(\boldsymbol{x}_n|z_k = 1)p(z_k = 1)}{\sum_{j=1}^{K} p(\boldsymbol{x}_n|z_j = 1)p(z_j = 1)} \\ &= \frac{\pi_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} = \gamma(z_{nk}). \end{aligned} \tag{4}$$

Then, for the EM algorithm, we define

$$\boldsymbol{\theta} = \{\pi_1, ..., \pi_K, \boldsymbol{\mu}_1, ..., \boldsymbol{\mu}_K, \boldsymbol{\Sigma}_1, ..., \boldsymbol{\Sigma}_K\}.$$

The first step is to initialise $\boldsymbol{\theta}$. This is done by performing the K-means algorithm [66]. This is another clustering algorithm with the hard constraint that each data point can only belong to one cluster. It does not specify the probabilities that a data point belongs to a certain cluster. From the K-means algorithm, we obtain the mean of each cluster. This mean is used to initialize $\boldsymbol{\mu}_k$, the mean of cluster k. Further, computing the within-cluster covariance of cluster k, we obtain the initial value for $\boldsymbol{\Sigma}_k$. The proportion of samples in cluster k is used to initialize π_k.

Next, the expectation step is performed. This is done by evaluating:

$$\mathrm{Q}(\boldsymbol{\theta}^*, \boldsymbol{\theta}) = E(\ln p(\boldsymbol{X}, \boldsymbol{Z}|\theta^*)) = \sum_{\boldsymbol{Z}} p(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{\theta}) \ln p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta}^*),$$

which can be rewritten as

$$\mathrm{Q}(\boldsymbol{\theta}^*, \boldsymbol{\theta}) = \sum_{\boldsymbol{Z}} \gamma(z_{nk}) \ln p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta}^*).$$

We find

$$p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta}^*) = \prod_{n=1}^{N} \prod_{k=1}^{K} \pi_k^{z_{nk}} \mathcal{N}(\boldsymbol{X}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_{nk}}$$

Taking logs, we obtain

$$\ln p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta}^*) = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk}(\ln \pi_k + \ln \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)). \tag{5}$$

As we are dealing with the restriction that $\sum_k \pi_k = 1$ and we are optimizing (5), the Lagrange multiplier is introduced:

$$\mathcal{L}(\boldsymbol{\theta}^*, \boldsymbol{\theta}) = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk}(\ln \pi_k + \ln \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)) - \lambda(\sum_{k=1}^{K} \pi_k - 1).$$

Differentiating $\mathcal{L}$ by π_k, μ_k and Σ_k and finding its root, we obtain the following optimal parameters

$$\begin{aligned} \pi_k^* &= \frac{\sum_{n=1}^{N} \gamma(z_{nk})}{N} \\ \boldsymbol{\mu}_k^* &= \frac{\sum_{n=1}^{N} \gamma(z_{nk}) \boldsymbol{x}_n}{\sum_{n=1}^{N} \gamma(z_{nk})} \\ \boldsymbol{\Sigma}_k^* &= \frac{\sum_{n=1}^{N} \gamma(z_{nk})(\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^T}{\sum_{n=1}^{N} \gamma(z_{nk})}, \end{aligned}$$

with $\gamma(z_{nk})$ being given in Equation (4). Next, $\gamma(z_{nk})$ is computed in the next iteration using the optimal parameters found. The log-likelihood in (5) is evaluated to observe if the log-likelihood changes. If there is convergence, i.e. the log-likelihood does not change or changes by a very small amount, the process converges and the algorithm terminates. This yields the final parameters $\boldsymbol{\theta}^*$. We use the VLFeat library [117] to obtain the parameters from GMM.

3.1.3.3 *Fisher encoding*

Having obtained the optimized parameter value $\boldsymbol{\theta}$ from GMM, we compute the Fisher vectors, following Perronnin et al. [89]. A Fisher vector encodes both the first and second order statistics between the video descriptors and a GMM. Given the parameter $\boldsymbol{\theta}$ and the descriptor set $\boldsymbol{X} \in \mathbb{R}^{T \times D}$, we assume that $\boldsymbol{X}$ can be modeled by the normal distribution $u_{\boldsymbol{\theta}}$. We describe $\boldsymbol{X}$ by the gradient vector:

$$G_{\boldsymbol{\theta}}^{\boldsymbol{X}} = \frac{1}{T} \partial_{\boldsymbol{\theta}} \log u_{\boldsymbol{\theta}}(\boldsymbol{X}),$$

where T is the number of descriptors. The gradient describes the contribution of the parameters to the generation process. A natural kernel on these gradients is subsequently given by

$$K(X, Y) = G_{\boldsymbol{\theta}}^{\boldsymbol{X}'} F_{\boldsymbol{\theta}}^{-1} G_{\boldsymbol{\theta}}^{\boldsymbol{X}},$$

with $F_{\boldsymbol{\theta}}$ being the Fisher information matrix of $u_{\boldsymbol{\theta}}$:

$$F_{\boldsymbol{\theta}} = E_{x \sim u_{\boldsymbol{\theta}}} \left(\partial_{\boldsymbol{\theta}} \log u_{\boldsymbol{\theta}}(\boldsymbol{X}) \partial_{\boldsymbol{\theta}} \log u_{\boldsymbol{\theta}}(\boldsymbol{X})' \right).$$

Since $F_{\boldsymbol{\theta}}$ is symmetric and positive definite, it has the Cholesky decomposition $F_{\boldsymbol{\theta}} = L_{\boldsymbol{\theta}}' L_{\boldsymbol{\theta}}$ and $K(X, Y)$ can be rewritten as a dot-product between normalized vectors $\mathcal{G}_{\boldsymbol{\theta}}$ with

$$\mathcal{G}_{\boldsymbol{\theta}}^{\boldsymbol{X}} = L_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}^{\boldsymbol{X}}.$$

Note that it is assumed that each $\boldsymbol{x}_n$ is generated independently by the normal distribution $u_{\boldsymbol{\theta}}$. Further, note that the covariance matrices are diagonal for each cluster. Using these observations, we can compute the d-th element of the Fisher vector with respect to the mean μ_k with the formula

$$\mathcal{G}_{\mu_k, d}^{\boldsymbol{X}} = \frac{1}{T\sqrt{\pi_k}} \sum_{n=1}^{T} \gamma_n(k) \left(\frac{x_{n,d} - \mu_k^{(d)}}{\sigma_k^{(d)}} \right)$$

and the d-th element of the Fisher vector with respect to the standard deviation σ_k is given by

$$\mathcal{G}_{\sigma_k, d}^{\boldsymbol{X}} = \frac{1}{T\sqrt{2\pi_k}} \sum_{n=1}^{T} \gamma_n(k) \left(\frac{x_{n,d} - \mu_k^{(d)}}{\sigma_k^{(d)} \sigma_k^{(d)}} - 1 \right),$$

with $x_{n,d}$ being the element of $\boldsymbol{X}$ with row n and column d. $\mu_k^{(d)}$ is the d-th element of the mean vector of cluster k and $\sigma_k^{(d)}$ is the d-th diagonal element of $\boldsymbol{\Sigma}_k$, which is the covariance matrix corresponding to cluster k. Further,

$$\gamma_n(k) = \frac{w_k e^{-\frac{1}{2}(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)}}{\sum_{t=1}^{K} w_t e^{-\frac{1}{2}(\boldsymbol{x}_n - \boldsymbol{\mu}_t)^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}_n - \boldsymbol{\mu}_t)}},$$

with w_k being the weight for cluster k, $k = 1, ..., K$. Computing each element, we obtain

$$\mathcal{G}_{\mu_k}^{\boldsymbol{X}} = \begin{bmatrix} \mathcal{G}_{\mu_k,0}^{\boldsymbol{X}} \\ \vdots \\ \mathcal{G}_{\mu_k,D-1}^{\boldsymbol{X}} \end{bmatrix}, \quad \mathcal{G}_{\sigma_k}^{\boldsymbol{X}} = \begin{bmatrix} \mathcal{G}_{\sigma_k,0}^{\boldsymbol{X}} \\ \vdots \\ \mathcal{G}_{\sigma_k,D-1}^{\boldsymbol{X}} \end{bmatrix},$$

yielding the D-dimensional Fisher vectors with respect to the mean and standard deviation. To obtain the final Fisher vector $\boldsymbol{F}$, for all clusters, we concatenate $\mathcal{G}_{\mu,k}^{\boldsymbol{X}}$ and $\mathcal{G}_{\sigma,k}^{\boldsymbol{X}}$ vectors for $k = 1, ..., K$, i.e.

$$\boldsymbol{f} = \begin{bmatrix} \mathcal{G}_{\mu,1}^{\boldsymbol{X}} \\ \vdots \\ \mathcal{G}_{\mu,K}^{\boldsymbol{X}} \\ \mathcal{G}_{\sigma,1}^{\boldsymbol{X}} \\ \vdots \\ \mathcal{G}_{\sigma,K}^{\boldsymbol{X}} \end{bmatrix},$$

yielding a $2Kd$ dimensional vector, describing each video by one large vector.

Having obtained the Fisher vector, we apply power and L2 normalization to the vector. It is shown by Perronnin et al. [89] that in this case, classification performance improves when transforming these vectors. We start by applying power normalization to each element, yielding vector $\bar{f}$, having element $\bar{f}(z)$:

$$\bar{f}(z) = \text{sign}(z)|z|^{\alpha}.$$

We use $\alpha = 0.5$, following Perronnin et al. Next, we apply L2 normalization to the obtained vector. The vector $\bar{f}$ is transformed as follows:

$$\tilde{f} = \frac{\bar{f}}{||\bar{f}||},$$

with $\tilde{f}$ being the final Fisher vector. We use the VLFeat library [117] to compute the Fisher vectors.

3.1.3.4 *Support Vector Machine*

Having computed the Fisher vector for each video, we use the support vector machine (SVM). Wang et al. [118] obtained good results using the linear SVM in combination with Fisher vectors. Further, the SVM can handle a large feature dimensionality, as indicated by Hua and Sun [47]. Therefore, we choose the linear SVM as our classifier for DT and IDT.

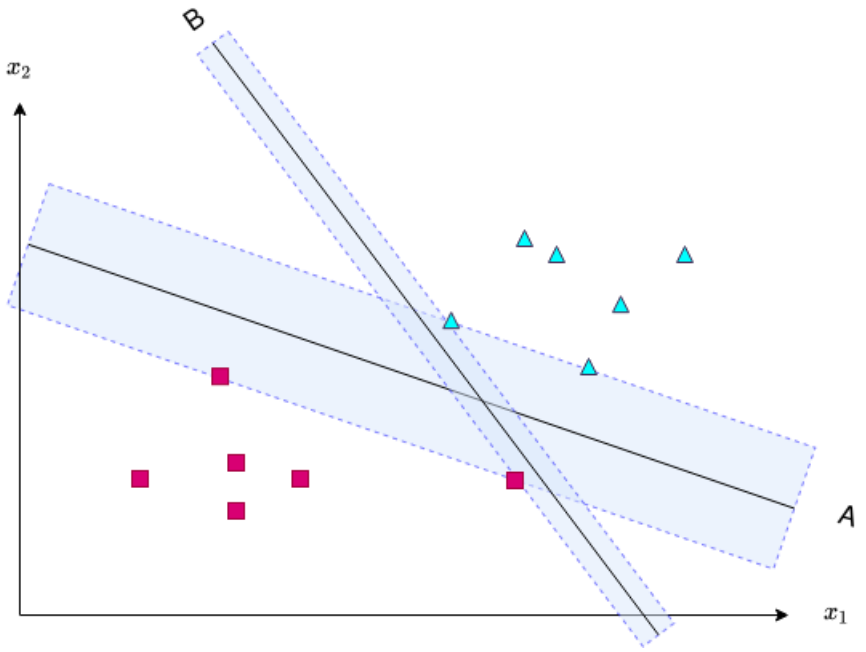

Figure 7: Example of two SVMs creating hyperplanes A and B, along with the corresponding support vectors. These hyperplanes separate two classes. The SVM creating hyperplane A has a larger margin than the SVM creating hyperplane B, since the distance between the two support vectors corresponding to A is larger. This figure is based on [29].

The SVM is a classifier proposed by Boser et al. [10]. The SVM is used to distinguish two classes from each other, given labeled data. The classifier distinguishes these classes using separating hyperplanes. An SVM specifies that the classes are best distinguished when the margin between the hyperplane separating the classes is maximized. This hyperplane is defined by

$$\boldsymbol{w}^T \boldsymbol{x}_i + b = 0,$$

where $\boldsymbol{w}$ is the weight vector and b is the bias, which are among the parameters to be optimized. In case $\boldsymbol{w}^T \boldsymbol{x}_i + b < 0$, the classifier predicts that the data belongs to the first class. In case $\boldsymbol{w}^T \boldsymbol{x}_i + b > 0$, the classifier predicts that the data belongs to the second class.

In order to find the optimal parameters, the soft margin classifier is used. This corresponds to the following optimization problem:

$$\begin{aligned} &\min_{\boldsymbol{w},b,\zeta} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{n}\zeta_i \\ &\text{s.t.} \\ &y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1 - \zeta_i \\ &\zeta_i \geq 0,\ i = 1, ..., n. \end{aligned} \tag{6}$$

This shows that the margin is maximized (by minimizing $\boldsymbol{w}^T\boldsymbol{w}$), while incurring a penalty when a sample is misclassified or within the margin boundary. In the ideal case, $y_i(\boldsymbol{w}^T\boldsymbol{x}_i) + b) \geq 1$ for all samples. However, the data is not always perfectly separable with a hyperplane. Therefore, some samples are

allowed to be at a distance ζ_i from the correct margin boundary. The penalty term C controls the strength of the penalty, acting as an inverse regularization parameter. The major role of C is to determine the trade-off between increasing the margin size and misclassification of training examples.

In the case of multi-class classification, we use the one-against-rest approach. We train one classifier against all the other classes, leading to a binary problem. We then train multiple classifiers, with the total number of classifiers being equal to the number of classes. To classify unseen data $\boldsymbol{x}_{new}$ with the obtained model for each classifier, each classifier obtains $\boldsymbol{x}_{new}$ as input. Each classifier computes a score and we select the classifier having the highest score. The scikit-learn package [85] is used to model the classifiers for the binary problem and the multi-class problem.

3.2 COMPRESSED VIDEO ACTION RECOGNITION

The deep learning algorithm we use in this research is CoViAR. Wu et al. [125] proposed this algorithm, making use of deep neural networks in combination with information provided by compressed videos to recognize different actions. In this section, we describe how the algorithm works. To implement the code, we use PyTorch [83], a Python library suitable for deep learning.

3.2.1 *Compression*

CoViAR takes compressed videos as input. The compressed videos are obtained from MPEG-4 compression. The compression algorithms use the fact that successive frames are usually very similar. One frame can be stored efficiently by reusing contents from another frame, storing the difference. Modern codecs split a video into I-frames (the intracoded frames), P-frames (predictive frames) and zero or more B-frames (bi-directional frames). I-frames are regular images. P-frames reference the previous frames and encode the change between frames. Both I-frames and P-frames are used by CoViAR. The CoViAR algorithm does not use the B-frames to model actions. For an detailed description of the MPEG-4 compression algorithm, we refer to [98].

With CoViAR, we divide the P-frames into two parts. The first part consists of the motion vectors. These are represented as the movements of block of pixels from the source frame to the target frame at time t, denoted by $\mathcal{T}^{(t)}$.

After compensating for block movement, a difference between the original image and the predicted image at time t can occur. We denote this residual difference by $\Delta^{(t)}$. This is the second part of the P-frames. Consequently, a P-frame consists of the motion vectors $\mathcal{T}^{(t)}$ and the residual $\Delta^{(t)}$:

$$I_i^{(t)} = I_{i-\mathcal{T}_i^{(t)}}^{(t-1)} + \Delta_i^{(t)} \tag{7}$$

for all pixels i, with $I^{(t)}$ denoting the RGB frame at time t.

Having obtained I-frames and P-frames, we feed the I-frames into a CNN. P-frames are not fed directly into the CNN, since they depend on the reference

frame. The reference frame is a frame of the compressed video which defines future frames. The reference frame may be a P-frame itself. This chain of dependency continues all the way back to a preceding I-frame. Therefore, treating each P-frame as an independent observation violates the dependency property.

A strategy to address this issue is to reuse features from the reference frame. Now, features are only updated given the new information. The CoViAR method makes use of a back-tracing technique decoupling individual P-frames. This is done by tracing all motion vectors back to the reference I-frame, accumulating the residuals. Now, each P-frame depends only on the I-frame and not on other P-frames. We can write this process in mathematical form. Given a pixel at location i in frame t, let $\mu_{\mathcal{T}^{(i)}}(i) = i - \mathcal{T}_i^{(t)}$ be the referenced location in the previous frame. The location traced back to frame $k < t$ is consequently given by

$$\mathcal{J}_i^{(t,k)} = \mu_{\mathcal{T}^{(k+1)}} \circ \ldots \circ \mu_{\mathcal{T}^{(t)}}(i)$$

Then, the accumulated motion vectors $\mathcal{D}_i^{(t)} \in \mathcal{R}^{H \times W \times 2}$ and the accumulated residuals $\mathcal{R}_i^{(t)} \in \mathcal{R}^{H \times W \times 3}$ at frame t are given by:

$$\begin{aligned}\mathcal{D}_i^{(t)} &= i - \mathcal{J}_i^{(t,k)} \\ \mathcal{R}_i^{(t)} &= \Delta_{\mathcal{J}_i^{(t,k+1)}}^{(k+1)} + \ldots + \Delta_{\mathcal{J}_i^{(t,t-1)}}^{(t-1)} + \Delta_i^{(t)}.\end{aligned}$$

Now, each P-frame has the dependency

$$I_i^{(t)} = I_{i - \mathcal{D}_i^{(t)}} + \mathcal{R}_i^{(t)},$$

which ensures that P-frames only depend on the I-frame. The recurrence relation shown in Equation (7) has been eliminated. Thus, P-frames can be processed in parallel. Now, the three input sources I-frames, motion vectors and residuals are modeled individually by a CNN: the I-frame CNN, the motion CNN and the residual CNN. Having obtained the scores for each CNN, we apply late fusion to combine the three networks to obtain final results. Furthermore, Temporal Segment Networks (TSN) [121] are used to model the long term dependency. The method of TSN is explained in Section 3.2.2.9. Late fusion is explained in Section 3.2.2.10.

Figure 8: Example of inputs for the three CNNs of CoViAR for a fall.

Figure 9: Example of inputs for the three CNNs of CoViAR for a non-fall.

Figures 8 and 9 show inputs for the three CNNs in CoViAR. The first three images depict I-Frame inputs, the middle three images depict the motion vectors and the last three images depict the residual frames.

3.2.2 *Convolutional Neural Network*

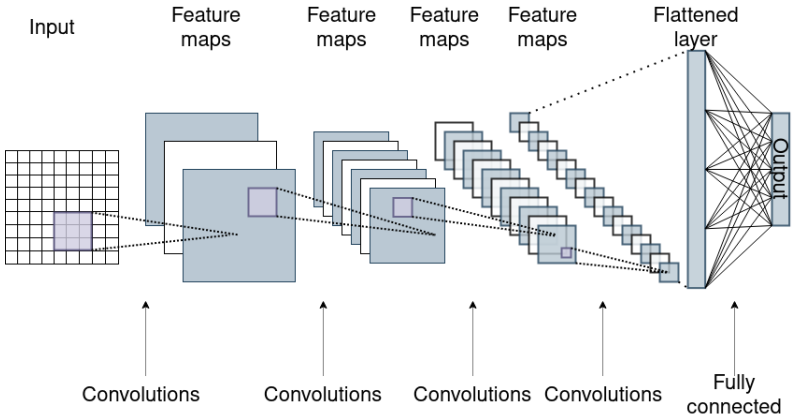

Figure 10: Example of a CNN depicting multiple convolutional layers and a fully connected layer. Note: The CNN in this image does not have the same architecture as the CNN used in this paper. This figure is based on [3].

A CNN is a neural network with multiple layers performing different operations in each layer. The network employs, among others, convolutions in the forward pass to compute a loss. The loss is computed using a loss function. Forwarding the data and backpropagation are repeated for multiple epochs in the network during training to obtain the optimal weights. We describe the different operations used in our CNN architecture in this section.

3.2.2.1 *Convolutional layer*

The convolutional layer is used as a filter to learn structures found in the frames. The filter itself is learned by iterating and updating the weights in the filter. This filter is applied to the data it is being fed. Making use of a convolution filter, we do not have to fully connect each pixel to a hidden layer. This is different to a regular neural network, where full connection occurs at each layer. Using convolution filters to train the model, the number of weights are manageable for modern hardware. Further, the convolution filter is shared among all elements of the input matrix in the current layer. Therefore, CNNs make use of weight sharing. In each layer, multiple filters are used to identify features.

The formula to compute the resulting element of convolution between $\boldsymbol{X}$ and $\boldsymbol{F}$ is given by

$$(\boldsymbol{X} * \boldsymbol{F})(m,n) = \sum_j \sum_i \boldsymbol{X}(m,n) \cdot \boldsymbol{F}(m-i, n-j),$$

where i and j correspond to the filter height and width. Zero padding is applied accordingly and the indices of m and n are determined by the stride that is used.

In our first layer of the CNNs, the RGB-frames are used as input for the I-frame CNN and residual CNN and two-dimensional motion vectors are used as input for the motion CNN. Consequently, we have three different channels, one for each primary color. This gives us three matrices for the

I-frame CNN and residual CNN. For the motion CNN, we have two different channels, as the motion vector is two-dimensional. This gives us two matrices. Each matrix has its own filter matrix $\boldsymbol{F}$, with learnable weights. In each CNN, after performing the convolution on each matrix, we take the sum of the resulting matrices. This yields a single matrix, i.e.

$$\boldsymbol{X}_{\text{sum\_after\_conv}} = \sum_{i=1}^{n_{\text{channels}}} \boldsymbol{X}_i * \boldsymbol{F}_i.$$

$\boldsymbol{X}_i$ and $\boldsymbol{F}_i$ are the matrix and filter corresponding to channel i respectively. Having performed this operation, we add bias b to each element of $\boldsymbol{X}_{\text{sum\_after\_conv}}$.

3.2.2.2 *Batch Normalization*

Having applied convolutions on the input matrix of each batch, we apply batch normalization on the output matrix of the batch. Batch normalization is performed as described by Ioffe and Szegedy [49]. This paper describes how each element in the input matrix is normalized. Each input element x_i is updated to output element y_i using the formula

$$y_i = \frac{x_i - E(\boldsymbol{x})}{\sqrt{\text{var}(\boldsymbol{x}) + \epsilon}}.$$

$E(\boldsymbol{x}) = \frac{1}{n}\sum_i x_i$ is the mean taken over all the elements of a batch corresponding to a certain output. $\text{var}(\boldsymbol{x}) = \frac{1}{n}\sum_i (x_i - E(\boldsymbol{x}))^2$ is the biased variance estimator of the elements in an output batch. Further, $\epsilon = 10^{-5}$ is used for numerical stability.

To illustrate, suppose a batch consists of two samples and we use two filter matrices. After convolution, suppose that we obtain matrices $\tilde{\boldsymbol{X}}_{i,j}$, $i, j \in \{0, 1\}$. Index i corresponds to the sample index in the batch and index j corresponds to the filter used to compute the output. Then, applying batch normalization, elements in $\tilde{\boldsymbol{X}}_{0,0}$ and $\tilde{\boldsymbol{X}}_{1,0}$ are normalized using the mean and variance of all elements in $\tilde{\boldsymbol{X}}_{0,0}$ and $\tilde{\boldsymbol{X}}_{1,0}$. A similar procedure is followed with $\tilde{\boldsymbol{X}}_{0,1}$ and $\tilde{\boldsymbol{X}}_{1,1}$. Thus, a full batch is used to apply normalization.

Furthermore, a layer keeps running estimates of its computed mean and variance, used for the normalization step. The running estimates use a momentum of 0.1, a parameter to indicate importance of the running estimates. For instance, having obtained the means $E(\boldsymbol{x})$ in the previous computation and obtaining $E(\boldsymbol{x})_t$ in the current computation, we use

$$\widehat{E(\boldsymbol{x})} = (1 - 0.1) \cdot E(\boldsymbol{x}) + 0.1 \cdot E(\boldsymbol{x})_t$$

and $\widehat{E(\boldsymbol{x})}$ is now used for batch normalization. The same procedure applies to $\text{var}(\boldsymbol{x})$.

3.2.2.3 *Rectified linear unit*

Another function used in a layer is the activation function. Activation functions are used to introduce nonlinearity to the network, allowing nodes in the network to learn more complex structures. The activation function

used by CoViAR is ReLU, an activation function introduced by Glorot et al. [37]. Its function is given by

$$f(x) = \max\{0, x\},$$

thus returning zero or the positive part of the input.

3.2.2.4 *Max pooling and average pooling*

In some layers, we want to downsample the data for the next layer by applying pooling methods. The pooling methods perform an operation on a block within a matrix, downsampling this block to one element. For max pooling, the maximum is taken of all elements in a block. For average pooling, the average is computed over all elements in a block. For instance, with a 4×4 matrix and a stride of two, applying one of the pooling functions, we have:

$$\mathrm{Pool_f}\left(\begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \\ x_{41} & x_{42} & x_{43} & x_{44} \end{bmatrix}\right) = \begin{bmatrix} \mathrm{f}\left(\begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix}\right) & \mathrm{f}\left(\begin{bmatrix} x_{13} & x_{14} \\ x_{23} & x_{24} \end{bmatrix}\right) \\ \mathrm{f}\left(\begin{bmatrix} x_{31} & x_{32} \\ x_{41} & x_{42} \end{bmatrix}\right) & \mathrm{f}\left(\begin{bmatrix} x_{33} & x_{34} \\ x_{43} & x_{44} \end{bmatrix}\right) \end{bmatrix},$$

with $f(\cdot)$ being the function returning the maximum or average of the elements of its input block.

3.2.2.5 *Fully connected layer*

Having created the convolution layers, the final layer is the fully connected layer. This layer is obtained by flattening the previous layer, i.e. reshaping the tensor or matrix to a vector. Subsequently, each element in the vector is assigned to a node. Each node in this layer is connected to all output nodes. This is the fully connected layer. The number of output nodes is equivalent to the total number of classes used for the classification problem.

We let the weight of the edges from node x_i to node y_j be denoted by w_{ij}. We further use the bias b_j for class j. Now, we can compute the score for each node y_j with the formula

$$y_j = \sum_i (w_{ij} x_i) + b_j.$$

3.2.2.6 *Loss function*

Having obtained the scores for each class from the fully connected layer, probabilities for each class are computed. These probabilities are computed with the SoftMax function, defined by the formula

$$p_i = \frac{e^{y_i}}{\sum_{j=1}^{C} e^{y_j}}.$$

y_i is the score for class i and C is the number of classes. To obtain the predicted class, we let the predicted class be the index corresponding to the maximum of the SoftMax scores.

Using the output from the SoftMax function, we compute the loss using

entropy. We use C as the number of classes and let the output obtained from SoftMax be given by:

$$\begin{bmatrix} p_0 & p_1 & \dots & p_{C-1} \end{bmatrix}.$$

Now, suppose that the correct class is 1, we compute the loss for a video with the formula

$$loss = -\log(p_1).$$

In each epoch, we compute the losses for each batch of input videos. Then, the average of the losses in the batch is computed. This average is used as information to update the weights and biases using backpropagation.

3.2.2.7 *Backpropagation*

CNNs update their weights and biases in a similar fashion as regular neural networks do. It is done by backpropagation. Using the loss function, the partial derivatives are computed. The partial derivatives are taken over the weights and biases in each layer. The weights and biases are updated using Adaptive moment estimation (Adam), introduced by Kingma and Ba [55]. The variant we use is the modified Adam using decoupled weight decay, proposed by Loshchilov and Hutter [67]. The optimizer makes use of learning rate α, exponential decay rates β_1 and β_2, λ for the weight decay parameter and $\epsilon = 10^{-8}$ for numerical stability. Following the paper, we set $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\lambda = 10^{-4}$. The algorithm runs as follows:

---

initialize:
  At time $t \leftarrow 0$, initialize parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \mathbf{0}$, second moment vector $\boldsymbol{v}_0 \leftarrow \mathbf{0}$, where $\mathbf{0}$ is a vector of zeros and schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
  $t \leftarrow t + 1$
  Compute gradient $\nabla f_t(\boldsymbol{\theta}_{t-1})$
  $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1})$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$
  $\hat{\boldsymbol{m}}_t \leftarrow \frac{\boldsymbol{m}_t}{1-\beta_1^t}$
  $\hat{\boldsymbol{v}}_t \leftarrow \frac{\boldsymbol{v}_t}{1-\beta_2^t}$
  $\eta_t \leftarrow$ SetScheduleMultiplier(t)
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t(\frac{\alpha \hat{\boldsymbol{m}}_t}{\sqrt{\hat{\boldsymbol{v}}_t + \epsilon}} + \lambda \boldsymbol{\eta}_{t-1})$
until All epochs are evaluated
return optimized parameters $\boldsymbol{\theta}_t$

Note that we perform element-wise operations. For instance, $\boldsymbol{g}_t^2 = \begin{bmatrix} g_1^2 & g_2^2 & \dots & g_n^2 \end{bmatrix}$. Further, our SetScheduleMultiplier(t) method takes into account the epochs and divides learning rate by ten when we reach a specified set of epochs for the I-frame CNN, motion CNN or residual CNN.

3.2.2.8 *Architecture*

Having described the components of the CNN, the architecture of the three CNNs is described next. Following the authors of CoViAR[125], we use the

same architectures. The architecture used for the I-frame CNN is ResNet-152 and the ResNet-18 architecture is used for the motion and residual CNN. Both architectures are proposed by He et al. [44]. Using these architectures, we can make use of transfer learning. This means that we use pre-trained models, such that we do not have to initialize weights randomly. Instead, we can use the weights obtained from these pre-trained models.

Layer	Output size	I-frame CNN	Motion and residual CNN
Conv1	112×112	7×7, 64, stride 2	
Conv2	56×56	3×3 max pool, stride 2	
		$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$
Conv3	28×28	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$
Conv4	14×14	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$
Conv5	7×7	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$
Final layer	1×1	average pool, fully connected layer, SoftMax	

Table 1: Architecture used for the three CNNs

In Table 1, we show the architectures used for the different CNNs. The CNN starts by performing a 7×7 convolution with stride 2. Next, batch normalization and ReLU are applied. Finally, in conv1, max pooling is applied to downsample the data. Moving on in the CNN, we immediately observe that in layers conv2 - conv5, the cells are different per architecture. For example, in conv2 of the I-frame CNN, we have the cell $\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$. This cell denotes three so-called bottleneck blocks, which are executed sequentially during the forward pass. In the bottleneck block, the following operations are executed, denoted as a sublayer:

- Sublayer 1:
  - Convolve input with 64 different 1×1 filters.
  - Apply batch normalization to the output. This output is used as input for sublayer 2.
- Sublayer 2:
  - Convolve input with 64 different 3×3 filters.
  - Apply batch normalization to the output. This output is used as input for sublayer 3.
- Sublayer 3:
  - Convolve input with 256 different 1×1 filters.
  - Apply batch normalization to the output and subsequently apply the ReLU activation function, yielding the output of this bottleneck block.

In conv2 - conv5, after having performed the forward pass through the first block, the data is downsampled by applying 1×1 convolution with stride 2 to the input obtained. For the motion and residual CNN, ReLU is applied in the first sublayer of each block, after performing batch normalization. For these two CNNs, ReLU is not applied in the last sublayer.

In the final layer, we apply average pooling to each matrix. We let the block size be equal to the matrix size, i.e. we take the average over all elements in the matrix. Next, these obtained averages are assigned to the nodes that are fully connected to the output nodes. For instance, the residual CNN has 512 different matrices after completing the operations in the conv5 layer. After average pooling, 512 averages are computed and assigned to 512 nodes. Each of these nodes is connected to each final output node. The final output node computes the final score for a corresponding class, as explained in Section 3.2.2.5.

3.2.2.9 *Keeping long term dependency with Temporal Segment Networks*

Temporal Segment Networks (TSN), proposed by Wang et al. [121], enable to model dynamics throughout the whole input tensor. The TSN algorithm operates on a sequence of snippets sampled from the entire video. Each snippet in this sequence produces its own prediction of the action classes. Next, a consensus among the snippets is derived to obtain the final video-level prediction. This is done by first dividing the input tensor into K segments, $\{S_1, S_2, ..., S_K\}$. Then, we randomly sample a snippet from each segment S_i, $i = 1, ..., K$. We next compute the scores in the CNN (before applying SoftMax) for each snippet and subsequently take the average of these scores to find the consensus score. Using this consensus score, we then compute the loss and obtain the prediction for a given input video.

Besides randomly selecting snippets, data augmentation is used to generate training samples and prevent overfitting[121]. This is done by following a few steps. First, the input image is resized to 256×340. Next, the cropped region is randomly selected from $\{256, 224, 192, 168\}$ for the I-frame CNN and randomly selected from $\{256, 224, 192\}$ for the motion and residual CNN. Having selected the region, the random crop is taken from the original frame. This selected crop is resized to 224×224. Next, random horizontal flipping is applied on the selected crop. Having executed this process, the input for the CNN is obtained.

3.2.2.10 *Late fusion*

Having obtained the outputs for the three different CNNs, we apply late fusion to combine the scores into one score. Given a video, we obtain the scores for each class in each CNN (before applying SoftMax). For instance, given video i and having C classes, we obtain the scores

$$\boldsymbol{s}_{\text{I-frame}} = \begin{bmatrix} s^{(0)}_{\text{I-frame}} \\ s^{(1)}_{\text{I-frame}} \\ \vdots \\ s^{(C-1)}_{\text{I-frame}} \end{bmatrix}, \boldsymbol{s}_{\text{Motion}} = \begin{bmatrix} s^{(0)}_{\text{Motion}} \\ s^{(1)}_{\text{Motion}} \\ \vdots \\ s^{(C-1)}_{\text{Motion}} \end{bmatrix}, \boldsymbol{s}_{\text{Residual}} = \begin{bmatrix} s^{(0)}_{\text{Residual}} \\ s^{(1)}_{\text{Residual}} \\ \vdots \\ s^{(C-1)}_{\text{Residual}} \end{bmatrix}.$$

We then obtain the combined scores

$$s_{\text{combined}} = s_{\text{I-frame}} + s_{\text{Motion}} + s_{\text{Residual}}.$$

Thus, late fusion adds the scores of the three different CNNs after obtaining the score for each CNN individually. The class having the highest score is selected as the predicted class.

4 EXPERIMENTS AND RESULTS

In this chapter, we describe the data used, our experimental setup and show the results obtained from our experiments.

4.1 DATA USED IN THIS RESEARCH

Multiple participants were asked to simulate falls and other actions, wearing a camera on the neck and on the waist. The participants were instructed to perform an action. If this action was performed, the participant placed its hand on the lens. Having obtained these videos, we preprocessed these videos. After the preprocessing stage, we obtained 1459 videos. Each video has one label. For example, a video is given the label 'walking'. In the binary setting (fall versus non-fall), this video is relabeled to a non-fall.

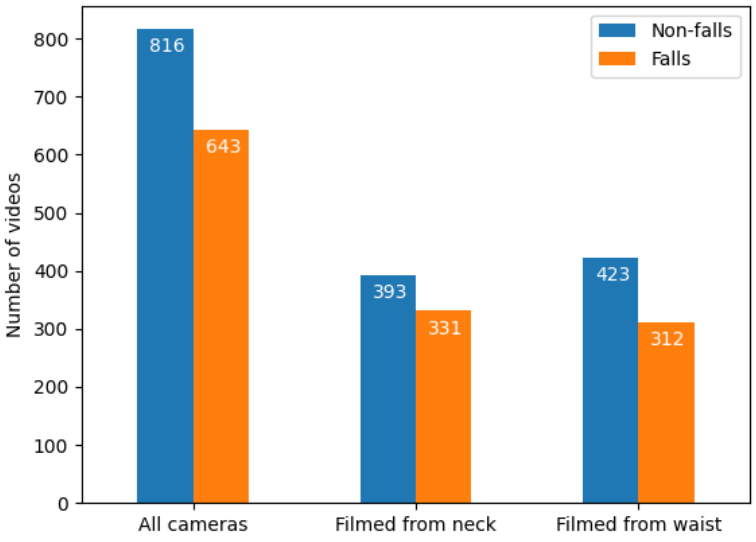

Figure 11: Class distributions in the binary settings for videos filmed from both camera mounting points, videos filmed from a camera mounted to the neck or videos filmed from a camera mounted to the waist.

After relabeling, we obtain the bar plots in Figure 11. The two bars on the left with index 'All cameras' show the distribution of the classes when videos filmed from both camera mounting points are used. The two bars on the middle with index 'Filmed from neck' show the distribution of classes when videos filmed from a camera mounted to the neck are used. The two bars on the right with index 'Filmed from waist' show the distribution of classes when videos filmed from a camera mounted to the waist are used.

Further, we investigate how the classifiers perform when using the original classes. The original classes are:

- **Sit:** The person goes from a standing position to a sitting position.
- **Walk:** The person is walking.
- **Lie in bed:** The person goes from a standing position to lying in bed.
- **Rise from bed:** The person rises from its bed.
- **Sit in chair:** The person goes from a standing position to sitting in a chair.
- **Fall front left:** The person falls to the front left from a standing position.
- **Fall back left:** The person falls to the back left from a standing position.
- **Fall back lying:** The person falls on the back from a standing position. The person ends op lying.
- **Fall syncope wall:** The person falls down, while sliding against the wall. The syncope is simulated.
- **Fall lateral right:** The person falls to the right from a standing position.
- **Fall down syncope:** The person falls from a standing position, simulating syncope.
- **Fall front:** The person falls to the front.
- **Fall lateral left:** The person falls to the left from a standing position.
- **Fall front knee:** The person falls to the front on its knees.
- **Fall front right:** The person falls to the front right from a standing position.
- **Fall back right:** The person falls to the back right from a standing position.
- **Stumble:** The person stumbles.
- **Limp:** The person walks with difficulty.
- **Bending:** The person bends and looks towards the floor.
- **Squatting down:** The person squats down.

This is the multi-class setting, used to further assess performance of the selected algorithms.

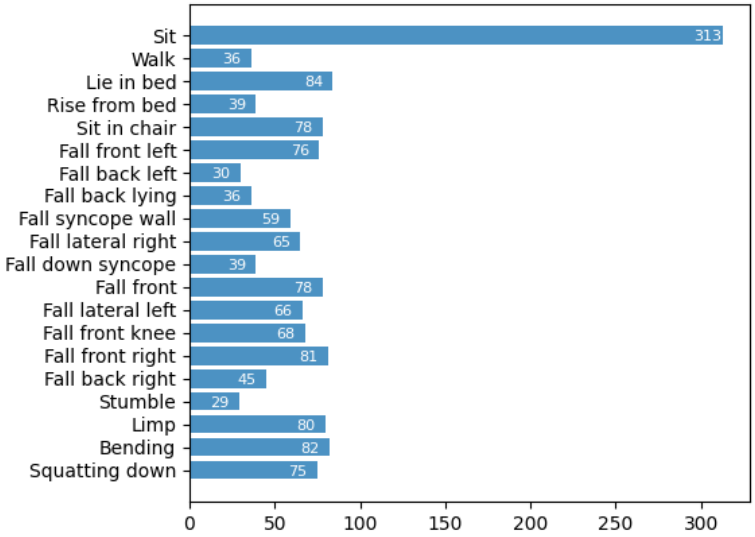

Figure 12: Class distribution of videos for multiple classes filmed with cameras mounted to the neck or waist.

The class distribution of this data set is shown in Figure 12. We observe that we are dealing with an imbalanced data set, as videos with the class **Sit** are overrepresented in the data set.

Having obtained the class distributions, we decided to create multiple data sets for some problems. The first data set of each problem was created to find the best performing classifier. These data sets are fall_non-fall_all-cameras, fall_non-fall_neck_1, fall_non-fall_waist_1 and all-classes_all-cameras_1. Each data set is annotated by the type of problem and the camera used. For instance the data set fall_non-fall_neck_1 refers to the binary problem with videos filmed from cameras mounted to the neck. Further, we create extra data sets: fall_non-fall_neck_2, fall_non-fall_waist_2 and all-classes_all-cameras_2. These are used to further assess the best performing classifier. The data sets annotated with a 2 ensure that these data sets have the same videos in the train and test set as fall_non-fall_all-cameras. The data sets annotated with a 2 are added to compare results with fall_non-fall_all-cameras. Doing so, we can observe whether training the classifier using both camera mounting points in the data set yields a different result compared to training the classifier using a data set having videos with a specific camera mounting point. Further, all-classes_all-cameras_2 is used to evaluate whether it is useful to train at the multi-class problem first and then relabel the predicted classes to the binary classes.

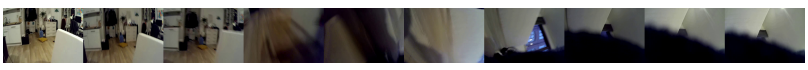

Figure 13: Falls example

Figure 14: Non-falls example (squatting down)

In Figures 13 and 14 we show an example of a fall and a non-fall. In the frames shown, it can be seen that the frames where the person is falling are more chaotic due to sudden movements compared to the frames where the person performs a controlled action.

4.2 EXPERIMENTAL SETUP

In this section we describe how we setup the experiments and reason why we chose this setup. To run the experiments, we use the Peregrine High Performance Computing cluster.

4.2.1 *Preprocessing of the videos*

After obtaining the videos from subjects performing different actions using an egocentric view, we preprocessed the videos manually. The persons simulating falls were instructed to put a hand on the camera once their action stopped. We filtered out the frames where the hand enters the frame, as these did not convey any information about the action.

Further, multiple actions were filmed within one video. Thus, multiple frames showed a hand being placed on the lens to signal the ending of an action. Starting a new action, the hand was removed from the lens. Removal of the hand from the lens added a lot of lighting to the first few frames of the video. We decided to filter out these frames as well.

Finally, we ensured that the distribution of falls and non-falls where distributed similarly in terms of frame length. Doing so, we ensured that the classifiers do not distinguish falls from non-falls simply by evaluating the number of frames.

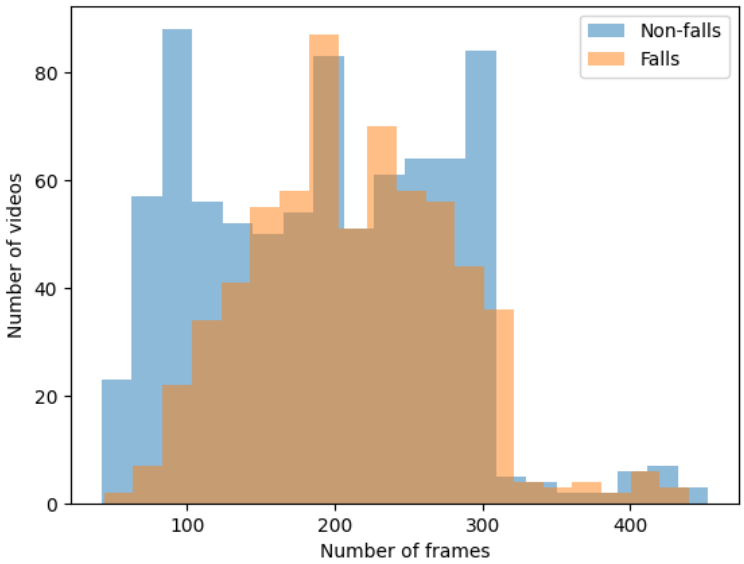

Figure 15: Histogram depicting the distribution of frames for videos classified as a fall and videos classified as a non-fall.

Figure 15 shows the distribution of the number of frames for videos classified as a fall or non-fall. The two distributions are similarly distributed in terms of frame length. Further, we resized all videos to 600×400. Frames are therefore smaller and thus can be faster processed by the algorithms.

4.2.2 *Hyperparameter optimization*

After preprocessing the videos, we trained the selected algorithms. The training process started by finding optimal hyperparameters for the algorithms DT, IDT and CoViAR. DT and IDT are similar algorithms in terms of parameters. The difference between these algorithms is the way the descriptors are retrieved. Therefore, we optimized the same hyperparameters for these algorithms. For CoViAR, we optimized a different hyperparameter. This hyperparameter is not used by DT and IDT.

To perform the hyperparameter optimization, we divided each data set into a train and test set, using a 80/20 split. Subsequently, hyperparameter optimization was performed within the train set, using ten-fold cross validation. Ten-fold cross validation is performed by splitting the train set into ten folds. Nine of these folds are concatenated into one set. The classifiers were trained on this set. Next, performance metrics are obtained by predicting the classes in the fold, the validation set, that was not part of the nine folds. This process was repeated ten times, where each fold is used as validation set once. From the validation sets, we obtain a performance metric for each validation set. The average of the performance metric over the validation sets is taken. The hyperparameter corresponding to the highest performance metric is selected for training on the full train set and prediction on the test set.

For DT and IDT, we optimized the hyperparameter L, the number of result-

ing dimensions after executing the PCA algorithm. For an explanation of this parameter, we refer to Section 3.1.3.1. L is varied by values

$$\{10, 25, 50, 75, 100, 150, 200\}.$$

We ensured that the proportional variance of the data after PCA is greater than 95%.

Further, we optimized hyperparameter K, the number of clusters used by GMM. This parameter was explained in Section 3.1.3.2. K is varied by values

$$\{16, 32, 64, 96, 128, 192, 256, 384\}.$$

Besides varying the hyperparameters L and K, we further varied the combinations of the descriptors. Given that we have descriptors HOF, HOG, MBH, and the trajectories, we have the combinations

$$\{HOF, HOF - HOG, HOF - HOG - MBH,$$
$$HOF - HOG - MBH - Trajectory, HOF - HOG - Trajectory,$$
$$HOF - MBH, HOF - MBH - Trajectory, HOF - Trajectory,$$
$$HOG, HOG - MBH, HOG - MBH - Trajectory,$$
$$HOG - Trajectory, MBH, MBH - Trajectory, Trajectory\}.$$

Note that some descriptor combinations have less than 50, 100, 150 or 200 dimensions. Therefore we could not reduce these sets to specific dimensions chosen for PCA. Hence for some descriptor combinations, we do not use all values of L. For instance, HOF has 108 dimensions. This descriptor can thus not be reduced to 150 or 200 dimensions.

Furthermore, the hyperparameter C of SVM (Section 3.1.3.4) was optimized. This was done with the method GridSearchCV from the scikit-learn library[85]. This method performs a grid search using specified values for parameter C. C is varied by values $\{1, 10, 100, 1000\}$. Subsequently, GridSearchCV selects the optimal parameter by applying an internal cross validation on the train set received. The hyperparameter is selected by maximizing the mean accuracy obtained from the internal folds. Five internal folds are used for optimization.

For CoViAR, we varied the learning rate α, which is used in the Adam optimizer for updating the weights. We only varied this parameter due to a constraint in computational resources. This parameter is described in Section 3.2.2.7. We varied α by

$$\{0.001, 0.005, 0.0003\}.$$

for all three CNNs. We selected the learning rate for each CNN apart. Due to the same constraint in computational resources, we decided to only perform hyperparameter optimization on the data set fall_non-fall_all-cameras to find the optimal learning rate. Next, this selected learning rate for each CNN was used for all classification problems.

4.2.3 *Remaining parametric settings*

To estimate the GMM parameters, used in the DT and IDT algorithms, features are randomly sampled. For the training procedure for hyperparameter

optimization, we sampled 108,000 features from the train set to estimate the GMM parameters. When using our complete training set for final inference, we sampled 120,000 features from the videos in the train set to estimate the GMM parameters. Further, to prevent overfitting of the SVM classifier, we follow the procedure as described in [46]. After cross validation, we compute the average of the AUCs obtained after training in each fold. We further compute the average of the AUCs obtained in the validation sets. Furthermore, we compute the standard deviations over the folds. The obtained average AUCs in the train sets and validation sets are compared. If these values differ by a large margin, we do not select these classifiers with corresponding hyperparameters, as a large margin suggests that the classifier is overfitting.

Other parameteric settings for CoViAR are set following [125]. All videos are resized to 340×256 at the start of the algorithm, but are later on resized to 224×224, as described in Section 3.2.2.9. The CNN is further fine-tuned using the modified ADAM, as explained in Section 3.2.2.7. Further, to train the CNN, we divide the video in three segments. For testing, we divide the video in 25 segments. Besides optimizing the learning rate, we used the standard settings as specified in 3.2.2.7. We further divided the learning rate by ten after running the algorithm for a specific number of epochs. The specific number depends on the CNN that is being trained. We use the following settings for each CNN

- I-frame CNN: 220 epochs in total, divide the learning rate by ten after every 55 epochs. A batch-size of 10 is used.
- Motion CNN: 360 epochs in total, divide the learning rate by ten after every 80 epochs. A batch-size of 20 is used.
- Residual CNN: 300 epochs in total, divide the learning rate by ten after every 60 epochs. A batch-size of 20 is used.

We further note that CoViAR is a deep learning algorithm, and that deep learning algorithms are prone to overfitting. Therefore, we use a validation set for CoViAR for the final training. While training, CoViAR is assessed on the validation set. This validation set consists of 30% of the training samples. The training set consists of the remaining 70%. We investigate the validation loss during training. If we observe that the classifier starts overfitting, i.e. the validation does not improve, while the training loss keeps decreasing, we apply early stopping. If we observe that overfitting occurs, we save the weights corresponding to the lowest validation loss.

4.2.4 *Metrics used for evaluation*

To decide which hyperparameters we should use for our binary classification problems, a metric is needed for evaluation. The metric we use is the area under the curve (AUC) of the receiver operating characteristic (ROC) curve. The ROC curve is a plot illustrating the diagnostic ability to discriminate between two classes. The plot is created by varying the classification thresholds. These thresholds are compared to the scores obtained from the classifier. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. FPR is plotted at the x-axis and TPR is plotted at the y-axis. TPR and FPR are given by

$$TPR = \frac{TP}{TP + FN}, \quad FPR = \frac{FP}{FP + TN},$$

where TP is the number of true positives, FP is the number of false positives, TN is the number of true negatives, and FN is the number of false negatives. Thus, all types of correct predictions and errors are taken into account. We further identify that both maximizing the number of correct and minimizing the number of incorrect predictions is important. Therefore, the ROC curve is an appropriate metric for evaluation. In the case of a fall and a non-fall, we have the following definitions for TP, FP, TN, and FN:

- TP: Cases where the classifier predicts correctly that a fall occurs.
- FP: Cases where the classifier predicts that a fall occurs, while it does not.
- TN: Cases where the classifier predicts correctly that a fall does not occur.
- FN: Cases where the classifier predicts that a fall does not occur while it does occur.

Having obtained the ROC curve, we observed how well the classifier performs by obtaining a metric for the complete curve. This is done by computing the AUC. The area under the ROC curve is obtained by aggregating the performance across the classification thresholds. The AUC quantifies the overall ability of the classifier to discriminate between, in our case, falls and non-falls.

Other metrics reported for fall detection are metrics often used in the literature for fall detection [40]. These metrics are precision, recall, specificity, and accuracy. Note that recall is the same metric as TPR (or sensitivity). The formulas of the metrics precision, specificity, and accuracy are given, in abbreviated form, by

$$prec = \frac{TP}{TP + FP}, \quad spec = \frac{TN}{TN + FP}, \quad acc = \frac{TP + TN}{TP + FP + TN + FN}.$$

Another metric evaluated is the FPR. Further, the precision-recall (PR) curve is evaluated to further assess the algorithm. To obtain the PR curve, a similar procedure as for the ROC curve is followed. For the PR curve, however, we plot recall on the x-axis against precision on the y-axis under various threshold settings. This shows how the algorithms perform under various threshold settings, without taking into account the correctly identified non-falls. Finally, confusion matrices are computed, obtained from the different classifiers after testing.

Using the ROC curve for the multi-class problem, we could create K different ROC curves, one for each class, where we have K classes. However, this compromises the insensitivity to class skew of the ROC curve, as explained in [33]. For instance, one class c_k in the multi-class setting might be easy to identify. Consequentially, this could produce low scores for the classifier when the classifier trains for class $c_i, i \neq k$. Then, increasing the number of samples with class c_k might alter performance, which alters the ROC curve. However, it does not necessarily mean that classifier performance of distinguishing classes is altered. Therefore, producing K different ROC curves is sensitive to class skew, as it does not necessarily show that the classifier is better or worse at distinguishing classes in the multi-class setting.

To select the hyperparameters for the multi-class problem, the ROC curve and its AUC are computed as proposed by Hand and Till [43]. With this method, the AUC is calculated for every pair of classes. The derivation of the computation is based upon the principle that the AUC is equivalent to the probability that the classifier ranks a randomly chosen positive instance higher than a randomly chosen negative instance. Using this form, a metric is devised that measures the unweighted pairwise discriminability of classes. To compute this metric, we let

$$AUC(c_i, c_j) = \frac{p(i|j) + p(j|i)}{2},$$

with $p(i|j)$ being the probability that a randomly drawn member of class j has a lower estimated probability of belonging to class i than a randomly drawn member of class i. The formula to compute the total AUC_{total} is subsequently given by

$$AUC_{\text{total}} = \frac{2}{|C|(|C|-1)} \sum_{\{c_i, c_j\}} AUC(c_i, c_j),$$

with C being the set of classes and $|C|$ being the number of classes. For details on AUC_{total} and the computation of $p(i|j)$, we refer to [43]. AUC_{total} is insensitive to changes in the class distribution [33], which is useful for our case with the imbalanced data set in the multi-class problem.

Further, dealing with an imbalanced data set, we compute macro-averaged metrics for the multi-class problem. Macro-averaged metrics are computed by first computing each metric per class, using the one-vs-all principle. Next the average of the metric is computed, treating all classes equally. For example, suppose we compute the macro-averaged precision. Further, suppose we obtain $TP = 1, FP = 1$ for class A, $TP = 10, FP = 90$ for class B. and $TP = 1, FP = 1$ for class C. Then, the macro-averaged precision is given by $\frac{1}{3}(\frac{1}{1+1} + \frac{10}{90+10} + \frac{1}{1+1}) = \frac{11}{30} \approx 0.367$. For our final results in the multi-class problem, the confusion matrix is shown. Further, the accuracy, macro-averaged recall and macro-averaged precision are given, along with a bar plot of the recall and precision for each class.

4.3 RESULTS

Having performed the hyperparameter optimization, we find the following optimal learning rates for CoViAR: 0.001 for the I-frame CNN; 0.005 for the motion CNN; 0.005 for the residual CNN. Furthermore, for the traditional algorithms, the optimal hyperparameters varied for each classification problem. The optimal hyperparameters are given in the tables in the following subsections. These tables show the results obtained after predicting the classes in the test set after training the different classifiers. We further show the tables and graphs that correspond to the training phase of the classifiers. Moreover, the ROC curves and PR curves are shown. Finally, the confusion matrix corresponding to the best performing classifier is given. Remaining results are given in Appendix A.

4.3.1 *Results obtained from* fall_non-fall_all-cameras

Classifier	PCA dim	Clusters	C	ROC-AUC training	ROC-AUC validation
DT Trajectory	25	192	100	99.99 (0.01)	98.03 (1.06)
IDT HOF - HOG - MBH	150	256	1000	99.03 (0.14)	97.48 (1.54)

Table 2: Train and validation AUC for the traditional algorithms for the fall_non-fall_all-cameras. Standard deviations are given in brackets.

Comparing the training and validation AUCs in Table 2, we do not observe large differences between the values, suggesting that the classifiers do not overfit.

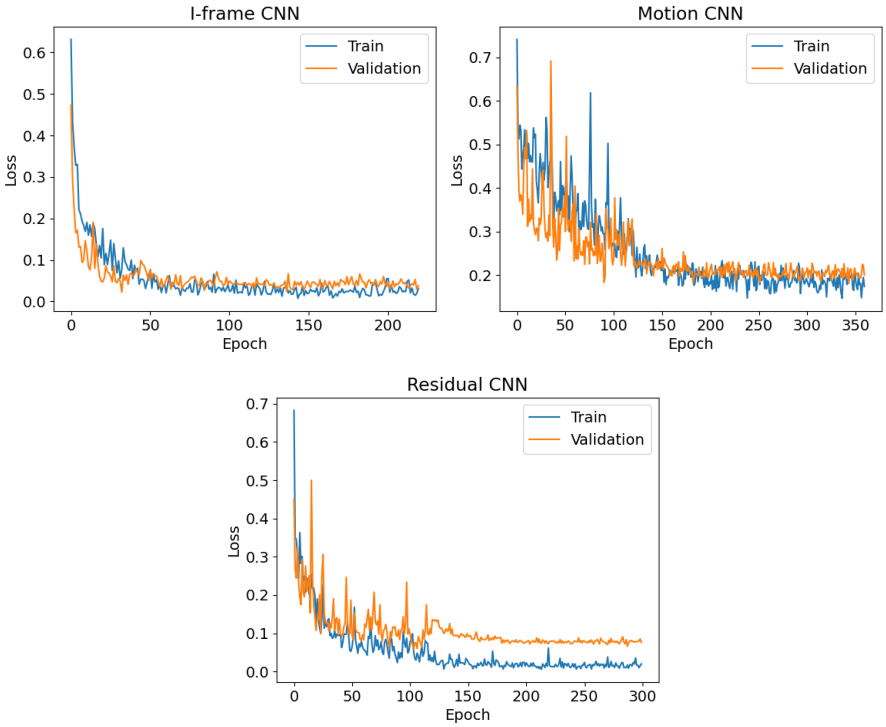

Figure 16: Training and validation losses for the CoViAR CNNs obtained after training on fall_non-fall_all-cameras.

Figure 16 shows both the training losses and validation losses for the three CNNs of CoViAR, obtained during training. We see that both losses follow the same path. The plot shows that the validation loss does not diverge from the training loss for the I-frame and motion CNN. For the residual CNN, there seems to be an increase after epoch 100, but this increase has decreased to its old level around epoch 175, following a similar path as the training loss. Thus, no early stopping is applied to this CNN.

Classifier	PCA dim	Clusters	ROC-AUC	Accuracy	Recall (TPR)	FPR	Precision	Specificity
DT Trajectory	25	192	97.58	90.78	88.37	7.32	90.48	92.68
IDT HOF - HOG - MBH	150	256	98.47	93.52	89.92	3.66	95.08	96.34
CoViAR I-frame CNN	x	x	99.54	**97.95**	**99.22**	**3.05**	**96.24**	**96.95**
CoViAR motion CNN	x	x	99.56	96.25	98.45	5.49	93.38	94.51
CoViAR residual CNN	x	x	99.61	96.59	96.12	**3.05**	96.12	**96.95**
CoViAR Late Fusion	x	x	**99.82**	**97.95**	**99.22**	**3.05**	**96.24**	**96.95**

Table 3: Performance metrics obtained after testing the different algorithms on fall_non-fall_all-cameras.

Table 3 shows the performance of the algorithms using the different performance metrics. The values are given in percentages. Bold values depict the best results. Note that a low FPR is better than a high FPR. This is due to the fact that FPR computes the rate of false positives out of all negatives. Further, this table shows the selected hyperparameter settings for DT and IDT.

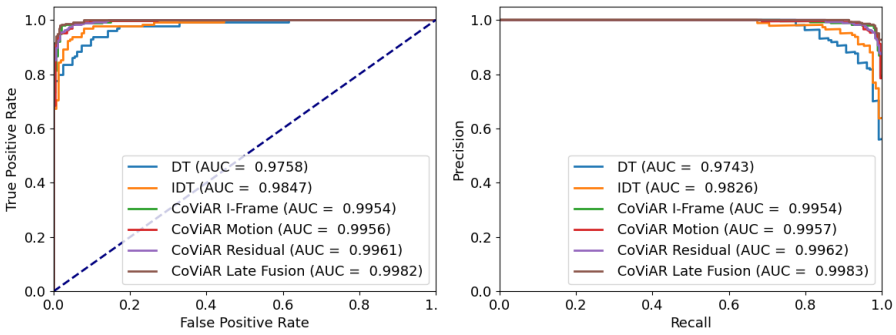

Figure 17: ROC curve (left) and PR curve (right) obtained after testing on fall_non-fall_all-cameras.

In Figure 17, we show the ROC curve and the PR curve, along with their corresponding AUC for the different algorithms.

From these results, we observe that CoViAR after late fusion performs best. We further observe that using IDT performs better than DT. The confusion matrix for CoViAR after late fusion is given in the table below, Table 4.

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	159	5	164
	Fall	1	128	129
	Total	160	133	293

Table 4: Confusion matrix after executing CoViAR after late fusion on fall_non-fall_all-cameras.

4.3.2 *Results obtained from* fall_non-fall_neck_1

Classifier	PCA dim	Clusters	C	ROC-AUC training	ROC-AUC validation
DT Trajectory	25	128	100	99.99 (0.01)	97.54 (1.86)
IDT HOF - MBH - Trajectory	150	128	1000	98.73 (0.84)	96.56 (2.98)

Table 5: Train and validation AUC for the traditional algorithms for the fall_non-fall_neck_1. Standard deviations are given in brackets.

Similar to the results for fall_non-fall_all-cameras, the training and validation AUCs in Table 5 do not show large differences, suggesting that the classifiers do not overfit.

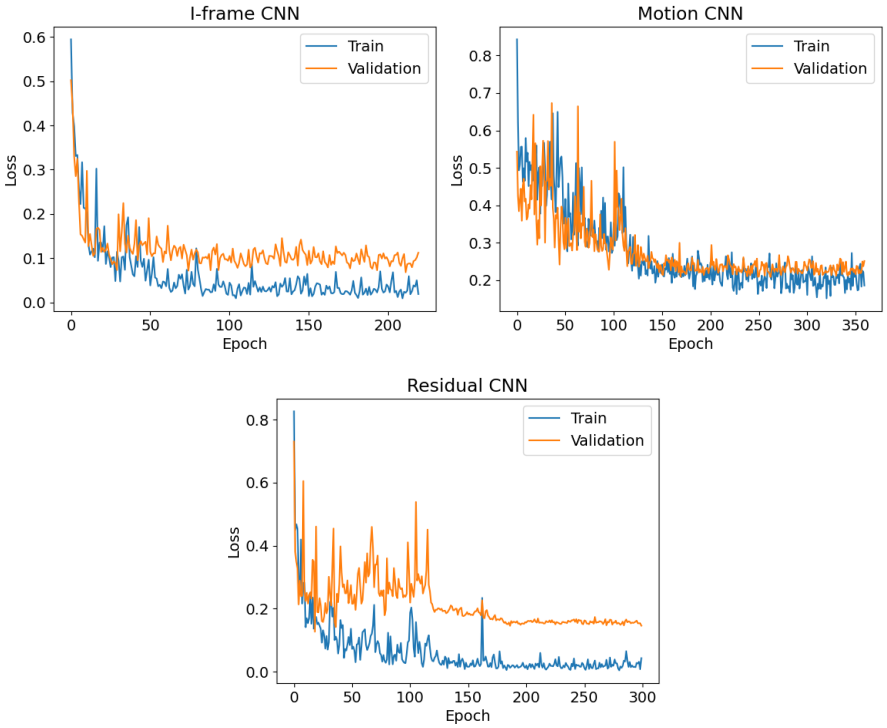

Figure 18: Training and validation losses for the CoViAR CNNs obtained after training on fall_non-fall_neck_1.

Figure 18 shows similar behaviour as observed in Figure 16. For the three CNNs, both training loss and validation loss follow the same path. Further, the validation loss does not diverge from the training loss.

Classifier	PCA dim	Clusters	ROC-AUC	Accuracy	Recall (TPR)	FPR	Precision	Specificity
DT Trajectory	25	128	99.26	93.84	97.01	8.86	90.28	91.14
IDT HOF - MBH - Trajectory	150	128	96.71	91.78	95.52	11.39	87.67	88.61
CoViAR I-frame CNN	x	x	99.26	95.89	95.52	3.8	95.52	96.20
CoViAR motion CNN	x	x	99.16	94.52	**98.51**	8.86	90.41	91.14
CoViAR residual CNN	x	x	**99.97**	**99.32**	**98.51**	**0.0**	**100.0**	**100.0**
CoViAR Late Fusion	x	x	99.96	97.95	97.01	1.27	98.48	98.73

Table 6: Performance metrics obtained after testing the different algorithms on fall_non-fall_neck_1.

In Table 6 the performance of the algorithms are shown using the different metrics. These values are given in percentages. Bold values depict the best results. Further, the selected descriptors for DT and IDT and the selected hyperparameters are given in this table.

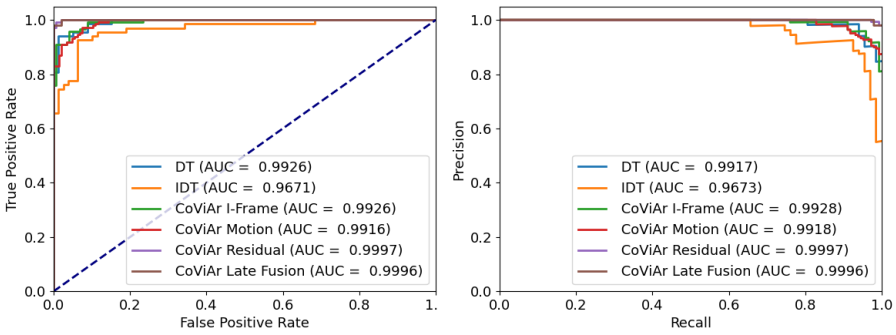

Figure 19: ROC curve (left) and PR curve (right) obtained after testing on fall_non-fall_neck_1.

In Figure 19 we show the resulting ROC curve and PR curve, along with their corresponding AUC for the different algorithms.

Classifying falls versus non-falls with the camera mounted to the neck, we observe that the residual CNN of CoViAR performs best. This is shown in the confusion matrices, ROC curves, PR curves, and the table showing the results on the test set. Comparing DT and IDT, results we observe that DT outperforms IDT. Further, the CoViAR CNNs outperform both DT and IDT.

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	79	0	79
	Fall	1	66	67
	Total	80	66	146

Table 7: Confusion matrix after executing the CoViAR residual CNN on fall_non-fall_neck_1.

Table 7 shows the confusion matrix corresponding to the best performing algorithm, the residual CNN of CoViAR.

4.3.3 *Results obtained from* fall_non-fall_waist_1

Classifier	PCA dim	Clusters	C	ROC-AUC training	ROC-AUC validation
DT MBH - Trajectory	150	384	1000	98.75 (1.22)	97.89 (1.21)
IDT HOF - MBH - Trajectory	200	256	1000	98.82 (0.56)	97.65 (1.07)

Table 8: Train and validation AUC for the traditional algorithms for the fall_non-fall_waist_1. Standard deviations are given in brackets.

Looking at Table 8, the results are similar as for training the traditional algorithms on fall_non-fall_all-cameras and fall_non-fall_all-cameras. The results suggest that the classifiers do not overfit.

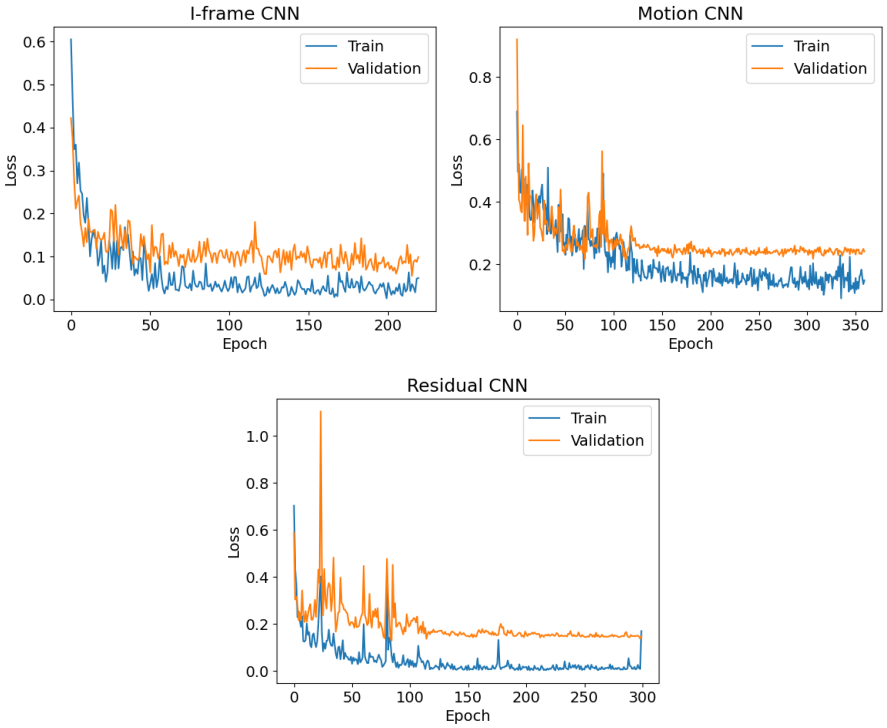

Figure 20: Training and validation losses for the CoViAR CNNs obtained after training on fall_non-fall_waist_1.

Similar to the other binary problems, similar behaviour of the losses is shown in Figure 20. The validation loss does not diverge from the path of the training loss as training loss decreases. Each CNN shows this behaviour for the losses.

Classifier	PCA dim	Clusters	ROC-AUC	Accuracy	Recall (TPR)	FPR	Precision	Specificity
DT MBH - Trajectory	150	384	97.57	91.95	90.62	7.06	90.62	92.94
IDT HOF - MBH - Trajectory	200	256	97.85	94.63	90.62	**2.35**	**96.67**	**97.65**
CoViAR I-frame CNN	x	x	99.89	**98.66**	**100.0**	**2.35**	**96.97**	**97.65**
CoViAR motion CNN	x	x	98.27	93.29	**100.0**	11.76	86.49	88.24
CoViAR residual CNN	x	x	99.96	**98.66**	**100.0**	**2.35**	**96.97**	**97.65**
CoViAR Late Fusion	x	x	**99.98**	**98.66**	**100.0**	**2.35**	**96.97**	**97.65**

Table 9: Performance metrics obtained after testing the different algorithms on fall_non-fall_waist_1

In Table 9 the performance of the algorithms are shown.

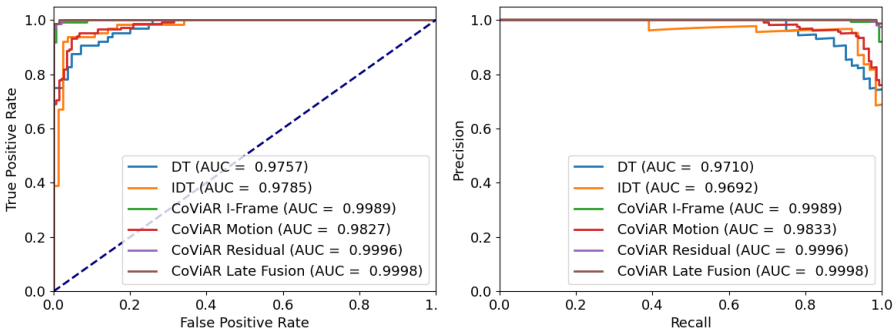

Figure 21: ROC curve (left) and PR curve (right) obtained after testing on fall_non-fall_waist_1.

In Figure 21 we show the ROC curve and the PR curve, along with their corresponding AUC for the different algorithms.

Both the residual CNN of CoViAR and CoViAR after late fusion perform good when inspecting algorithms classifying falls versus non-falls with the camera mounted to the waist. The difference between the two classifiers is that the classifier using the residual CNN has one false positive and one false negative, whereas CoViAR after late fusion has two false positives. Since detecting falls is more important than detecting non-falls, we prefer CoViAR after late fusion over the residual CNN. Further, we compare DT to IDT. We observe that IDT performs better on this data set.

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	83	2	85
	Fall	0	64	64
	Total	83	66	149

Table 10: Confusion matrix after executing CoViAR after late fusion on fall_non-fall_waist_1.

In Table 10, the confusion matrix obtained from executing CoViAR after late fusion is shown.

4.3.4 *Results obtained from* all-classes_all-cameras_1

Classifier	PCA dim	Clusters	C	ROC-AUC training	ROC-AUC validation
DT HOG - MBH - Trajectory	200	256	1000	99.00 (0.58)	92.93 (1.11)
IDT HOG - Trajectory	100	64	1000	99.66 (0.31)	93.03 (1.49)

Table 11: Train and validation AUC for the traditional algorithms for the all-classes_all-cameras_1. Standard deviations are given in brackets.

Compared to the binary problem, the difference between the AUCs is now larger for the multi-class problem. However, the difference is not large enough to suggest that the classifiers are overfitting, as we still have good average AUCs for the validation sets.

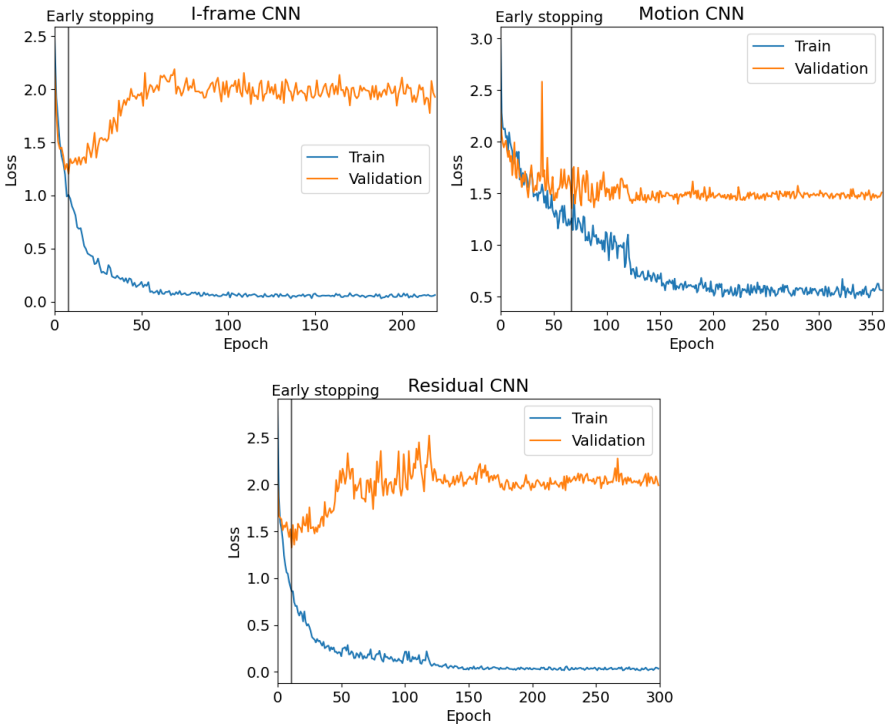

Figure 22: Training and validation losses for the CoViAR CNNs obtained after training on all-classes_all-cameras_1.

In contrast to the binary problem, we observe that the classifier starts overfitting when training on all-classes_all-cameras_1. This is the case for all three CNNs. For the CNNs, early stopping is applied at the epoch corresponding to the vertical black line.

Classifier	PCA dim	Clusters	AUC	Accuracy	Recall macro-averaged	Precision macro-averaged
DT HOG - MBH - Trajectory	200	256	93.22	**73.04**	64.32	67.04
IDT HOG - Trajectory	100	64	93.49	68.94	63.41	62.69
CoViAR I-frame CNN	x	x	93.12	62.88	55.81	53.78
CoViAR motion CNN	x	x	91.69	61.54	52.7	58.08
CoViAR residual CNN	x	x	93.85	63.21	56.68	64.6
CoViAR Late Fusion	x	x	**95.09**	70.9	**65.5**	**70.5**

Table 12: Performance metrics obtained after testing the different algorithms on all-classes_all-cameras_1.

Table 12 depicts the metrics obtained after executing the different algorithms classifying all classes.

We observe that CoViAR after late fusion outperforms the other classifiers when taking into account AUC, recall and precision. We further find that DT outperforms IDT in terms of accuracy, macro-averaged recall and macro-averaged precision.

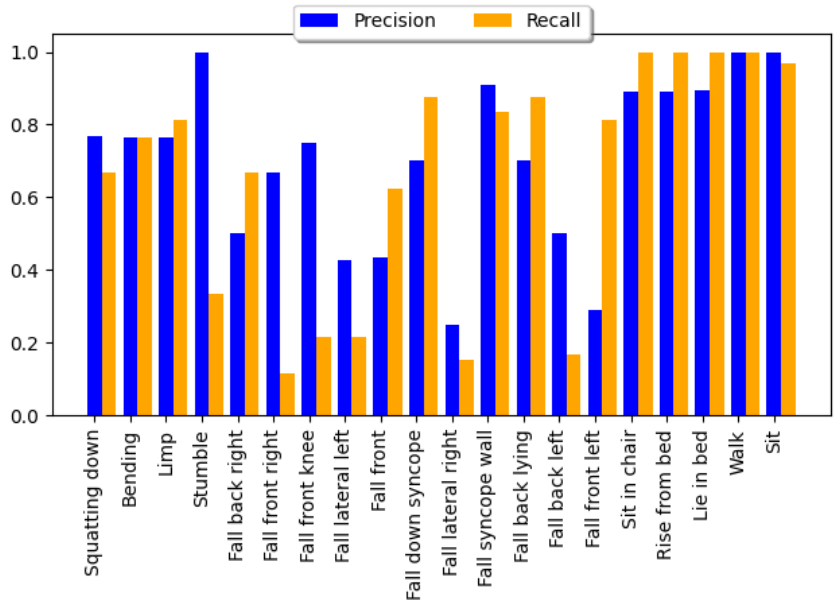

Figure 23: Precision and recall of the different classes after training CoViAR after late fusion on all-classes_all_cameras_1.

In Figure 23 we show the precision and recall for each class. We observe that most of the classes having low precision or recall are falls, indicating that these falls are hard to identify for the classifier.

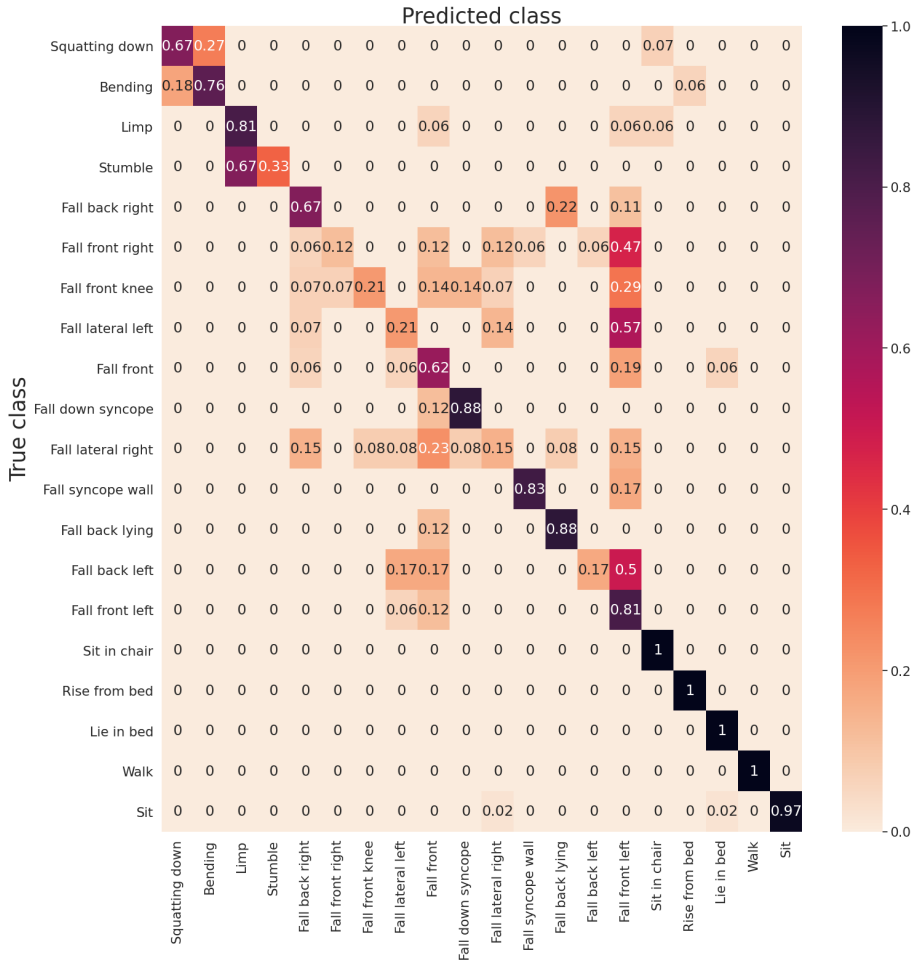

Figure 24: Confusion matrix after executing CoViAR after late fusion on all-classes_all_cameras_1.

In Figure 24, we show the normalized confusion matrix obtained after executing CoViAR after late fusion.

5 DISCUSSION

In this chapter we discuss the obtained results. Next, we discuss advantages and disadvantages of this research. This chapter ends by discussing the future research that one could conduct based on this research.

5.1 ANALYSIS OF THE RESULTS

This section discusses the results obtained in Chapter 4. The results are both quantitatively and qualitatively discussed. This is done by summarizing the quantitative results to get a clear view which classifier performed best. In the qualitative analysis, we inspect which videos the best performing classifier failed to classify correctly. We close this section by comparing our results to results found in the literature.

5.1.1 *Quantitative analysis of the results*

From the results, we observe that all algorithms can be used for fall detection, given their high performance on the data sets. Further, we find that the deep learning algorithm performs better than the traditional algorithms. For each constructed data set, CoViAR outperforms both DT and IDT.

For the traditional algorithms, we observe that the optimal descriptors are not always the same for either DT or IDT. However, one pattern emerges: Optical flow is used by the optimized classifiers, as each of the classifiers use HOF, MBH, or the trajectories. This is as expected, as we are dealing with motion detection.

Investigating CoViAR, we observe that late fusion boosts performance in most cases. For the multi-class problems, late fusion improves performances significantly. In some cases however, the residual CNN performs better by itself than combining them with others. The I-frame and motion CNN do not perform better than CoViAR after late fusion in any of the cases. Further, the motion CNN is the least performing CNN out of the three CoViAR CNNs for most data sets.

We further saw that, although being the best performing classifier, CoViAR had low precision and low recall for many fall types when being trained on all-classes_all-cameras_1. We asked ourselves whether this also meant that a lot of these fall types were misclassified as non-fall types. We investigate this by relabeling the predicted classes. Doing so, suppose that the classifier predicts a video as **Fall front left**, while its true class is **Fall front right**. Then, the classifier would be correct when we relabel the predicted class to one of the binary classes, since both a **Fall front left** and **Fall front right** are falls. After relabeling all predicted classes, we obtain the following confusion matrix

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	164	2	166
	Fall	2	131	133
	Total	164	135	299

Table 13: Confusion matrix after executing the CoViAR after late fusion on all-classes_all-cameras_1 and relabeling

Thus, in this case, we have a low number of false positives and low number of false negatives, indicating that the classifier mostly misclassified the type of a fall, but still had good performance when classifying falls or non-falls.

As CoViAR is our best performing classifier, we further trained CoViAR on the data sets fall_non-fall_neck_2, fall_non-fall_waist_2 and all-classes_all-cameras_2. We further trained DT and IDT on fall_non-fall_neck_2 and fall_non-fall_waist_2. The statistics and confusion matrices obtained are placed in the appendix, Sections A.2.5-A.2.8.

For each classifier, we can take the sum of the confusion matrix in Sections A.2.5 and A.2.6 to compare it with the confusion matrices obtained for the classifiers trained on fall_non-fall_all-cameras. These sums of the confusion matrices are shown in Section A.2.7. Comparing these summed confusion matrices to the matrices in Section 4.3.1 and A.2.1, we immediately observe that IDT performs better if we train on both camera mounting points simultaneously. This is found by comparing the confusion matrix in 17 to the confusion matrix in 43. We further observe that DT performs similar to training using both camera mounting points simultaneously. This can be observed by looking at Tables 16 and 42. For CoViAR, we find that performance does not improve. The classifier has four false positives and two false negatives. This indicates that training the classifier using videos from a specific camera mounting point does not necessarily yield better results, compared to training the classifier using videos from both camera mounting points. These results can be observed in Tables 4 and 44.

5.1.2 *Analysis of the misclassified videos of the best performing classifier*

Next, we analyse which videos failed to be correctly predicted by CoViAR for each problem. We analyse each of the misclassified videos to detect whether there is a pattern in the videos that the classifier failed to detect. When investigating the binary problems, we also compute the absolute difference in scores. This is defined as the absolute difference between the score for a fall and the score for a non-fall. Further, we use the average softmax probabilities for the predicted classes. A misclassified video can have a probability that is significantly lower than the average probabilities for the correctly predicted class. This difference in probability indicates that the classifier had problems with correctly predicting the correct class, which led to a misclassification.

We start by investigating the videos when training on fall_non-fall_all-cameras. For the combined CNN, the classifier has one false negative and five false positives. The false negative is a fall that occurs by slowly sliding down against a wall, i.e. the video has the class **Fall syncope wall**. The video is

filmed with a camera mounted to the waist. In this video, we could not clearly see whether a fall was occurring. It seems that the subject is either slowly sitting down or lying down in the videos. For the correctly classified videos having down syncope falls, we can see a fall occurring for most videos, except for one correctly classified video. Further, inspecting the mean absolute difference of the scores, we find that for the correctly classified videos, this difference is 22.76. The mean of probabilities is 0.97. If we leave out the correctly classified video not showing a fall, the mean of probabilities is 1.0. For the misclassified video, the absolute difference is 2.46 and the probability for the misclassified class is 0.92.

For the five false positives, all videos show a person limping. Four of these videos are filmed with the camera mounted to the neck. The fifth video is filmed with the camera mounted to the waist. Comparing the misclassified videos with the correctly classified videos, we cannot find large visual differences between the correctly and incorrectly classified videos. Both the correct and incorrect classified videos resemble the same strong sudden movements. Next, we investigate the scores for these videos. The mean absolute difference in scores is 20.48 for the correctly classified videos. The corresponding average probability is 1.0. For the incorrectly classified videos, the mean absolute difference between scores is 7.63. The corresponding average probability for the misclassified class is 0.90. Given the number of misclassified videos for limping, we find that the classifier is not good at distinguishing a person limping from a person falling.

Inspecting the videos in the test set of fall_non-fall_neck_1 for the CoViAR residual CNN, we observe that there is one video that the classifier failed to classify correctly. This is a video where the person is falling to the front left. Looking at the video, we see a clear fall. Comparing the video to the correctly classified videos, we do not observe visual differences. Further, the mean absolute difference in scores is 11.83. The corresponding average probability is 1.0. The absolute difference in scores for the misclassified video is 1.28 and the probability is 0.78. Thus, the classifier was uncertain whether a fall or non-fall occurs.

Next, inspecting the combined CNN, we have two false negatives and one false positive. The first false negative is a person falling to the front on its knees. Looking at the correctly classified videos and the misclassified videos, we do not observe visual differences. Next, looking at the scores, the correctly classified videos have a mean absolute difference of 27.10 and a corresponding average probability of 1.0. For the misclassified video, this absolute difference is 0.83 with a corresponding probability of 0.70. Thus, the classifier was uncertain whether the video was a fall or not.

The second false negative is a front fall to the right. Looking at the videos, we do not observe visual differences between the misclassified videos and the correctly classified videos. The mean absolute difference in scores for the correctly classified videos is 19.33. Its corresponding average probability is 1.0. For the misclassified video, this difference is 0.51 with a probability of 0.63. Hence, the classifier was not confident that this video was a non-fall.

The false positive is a video where the person is limping. The mean absolute difference in scores for the correctly classified videos is 27.02 with

a corresponding average probability of 1.0. The absolute difference for the misclassified video is 0.85. Its corresponding probability is 0.7. Thus, the classifier was not confident that the video was a fall.

Having executed CoViAR on the test set of fall_non-fall_neck_2, we find that three videos are misclassified. These videos all show a person limping. These videos were also misclassified when training on the videos filmed from both camera mounting points. The videos do not show substantial visual differences from the other videos where a person is limping. Further, the mean absolute difference for correctly classified videos is 17.91 with an average probability of 0.93. For the misclassified video, this difference is 7.56 and has a probability of 0.93.

For results obtained from training CoViAR on fall_non-fall_waist_1, two non-falls are falsely classified as falls. The first video shows a person limping. For this video, we do not observe a visual difference between the video that is misclassified and the videos that are correctly classified showing a person limping. Investigating the scores, the classifier was not very confident that this action was a fall. The mean absolute difference for the correctly classified videos is 18.26. The corresponding average probability is 1.0. The absolute difference in scores for the misclassified video is 1.39, and the probability is 0.80.

The other misclassified video shows that the person is stumbling. In this video, the action was not fully finished. The person is not looking to the front at the moment the video finishes. In other videos where the stumbling occurs, this does happen. The mean absolute difference in scores for the correctly classified videos is 8.58. It has an average probability of 1.0. For the misclassified video, the absolute difference is 1.15 and the average probability is 0.76. Comparing the score and the probability obtained for this video with the other scores and probabilities for videos where the person is stumbling, we thus observe that the classifier is not confident whether a fall or non-fall occurs for this video.

Next, looking at the results obtained after training CoViAR on Fall_non-fall_waist_2, we find that three videos are misclassified. Two misclassifications are false negatives and one misclassification is a false positive. The first false negative is a front fall on the knees. This video was not misclassified when using videos filmed from both camera mounting points. The mean absolute difference of scores for this class is 24.66 with a corresponding average probability of 1.0. The misclassified video has an absolute difference of 2.28 with a softmax probability of 0.91.

The other false negative is a down-syncope fall against the wall. This video was also misclassified when training from the videos filmed with a camera mounted to either the neck or waist. We investigate this fall. This fall against the wall has a soft fall. This soft fall is harder for the classifier to classify correctly. Observing videos in the train set where the person performs down-syncope falls as well, we notice that the videos in the test set are the only videos having soft falls. Therefore, the classifier might not be well trained on down-syncope falls against the wall having a softer fall. Further, the mean absolute difference in scores for the correctly classified videos is 19.24 with a corresponding average probability of 1.0. For the misclassified

video, this absolute difference is 3.68 with a corresponding probability of 0.98.

The false positive is a video where the person is limping. This video was also misclassified when training CoViAR on the videos from both camera mounting points. We have the same findings regarding visual differences. Investigating the scores, the mean absolute difference of correctly classified videos is 17.31 with an average probability of 1.0. For the misclassified video, this difference is 3.57. The probability of the misclassified video is 0.97.

We notice that the video of the class **Fall syncope wall** is wrongly classified for the waist, but not for the neck. We examined the videos in the test set of fall_non-fall_waist_2. We find that the only videos showing soft falls in the class **Fall syncope wall** are in the test set. These are not in the train set. The videos filmed from a camera mounted to the neck all have videos where a fall is not hard to identify by looking at the video. Therefore, the classifier might not have learned correctly about these soft falls, thus misclassyfing these videos as non-falls. This does not occur for the classifier training on fall_non-fall_neck_2, as there are no videos with soft falls having the class **Fall syncope wall** in this test set.

Finally, we investigate how the videos in the data sets all-classes_all-cameras_1 and all-classes_all-cameras_2 are classified. We find that most misclassified videos are misclassified as classes with a different fall or non-fall type. For instance, a video with class **Squatting down** is sometimes classified as **Sit in chair**. Further, for the results on All-classes_all-cameras_2, having predicted the classes, we changed the multi-class labels to binary labels. Doing this, we can infer whether the fall detection classifier improved if we let it train on multi-class problem first and then relabel the predicted classes. We investigate whether there is an improvement by comparing these results to the results on the fall_non-fall_all-cameras data set. Doing so, we obtain the following confusion matrix corresponding to the classifier CoViAR after late fusion:

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	163	0	164
	Fall	2	127	129
	Total	167	126	293

Table 14: Confusion matrix after executing the CoViAR after late fusion on all-classes_all-cameras_2 and relabeling

Comparing the confusion matrix in Table 14 to the confusion matrix in Table 4, we do not see an improvement by looking at the confusion matrix, since the number of false negatives is higher. One of these falls is the down syncope falls with the very soft falls. This fall was classified as a person limping. This video having a soft fall was not in the train set and is likely to be misclassified due to this reason. This video was also misclassified when training CoViAR after late fusion on fall_non-fall_all-cameras.

Further, we plotted the weights of the first layer of the three CNNs of CoViAR to grasp what the classifier has learned. These plots are shown in the appendix, in Section A.1.1. We observe that both the I-frame CNN and

residual CNN learned similar patterns. We cannot really observe patterns in the learned weights for the motion CNN.

5.1.3 *Comparison of results with the literature*

Taking into account the researches by Casares et al. [17], Ozcan et al. [78, 79, 80, 81] and Boudouane et al. [12, 13], we observe that our experiment shows that the selected algorithms are capable of functioning as reliable fall detectors. We use a different data set, thus we cannot directly compare our results to the other researches conducted on fall detection with a wearable camera. We further cannot compare our results directly to any other literature, since we use a different data set. However, we can compare performance between the classifiers to results obtained in the literature. Doing so, we can verify whether the current literature agrees or disagrees with the classifier performance found in this research.

In their research on IDT, Wang et al. [118] compared the results of DT with IDT. Similar to this research, both DT and IDT are executed. In their research, IDT outperformed DT. The data sets used for their comparison are filmed from a third-person viewpoint. Each data set is used to recognize actions. The data sets used are Hollywood2 [70], HMDB-51 [59], Olympic Sports [75] and UCF-50 [97]. In our case, it is unclear whether DT always outperforms IDT or vice versa. This result is in agreement with results obtained by both Li et al. [63], and Lu et al. [68]. In these papers, results are shown, where DT and IDT are compared. In the tables shown, we observe that DT outperforms IDT on a specific split of the data set, but IDT outperforms DT on another split of the data set. Both these researches are conducted on the egocentric data set GTEA Gaze. Thus, comparing with current literature, there is no real consensus whether IDT outperforms DT indefinitely. There have been cases where DT outperforms IDT for egocentric data sets and vice versa. For the data sets used in our research, there is also no consensus whether DT outperforms IDT.

Next, we look at the descriptors used for DT and IDT. We do not combine all descriptors for DT and IDT. For instance, for the data set fall_non-fall_all-cameras, we use the trajectory descriptor for DT. For IDT, we combine the HOF, HOG and MBH descriptors with each other. We cannot compare these findings to current literature, as other literature used one of the descriptors or combined all descriptors at once. In [87, 118, 119, 120], performance of individual descriptors are compared, along with the performance after combining the descriptors all together. Most of the results in these researches show that combining all descriptors give the best results. In [119, 120] however, the MBH descriptor outperforms the other descriptors or the combination of all descriptors for the KTH data set. This data set consists of six human actions, filmed from a third-person viewpoint. Our results indicate that research using DT or IDT might improve results when using a different combination of descriptors. An example of combinations could be HOG-MBH or HOF-MBH-Trajectory. Using a certain combination of descriptors, the classifier could outperform a classifier using individual descriptors or a combination of all descriptors. Furthermore, we found that using optical flow improved the results for DT and IDT. This is in agreement with the literature, shown in [87, 118, 119, 120].

In this research, we find that CoViAR outperforms both DT and IDT for multiple classification problems. We compare this to current literature. Wu et al. [125] compared their CoViAR algorithm directly to IDT. They find that CoViAR outperforms IDT on the HMDB-51 data set. Further, the performance of the proposed algorithms in [16, 116] were compared with IDT. These algorithms have been compared to CoViAR in [125]. However, for a direct comparison of these algorithms with CoViAR, the HMDB-51 and UCF-101 data sets have been used. In [16, 116], data sets ASLAN[57] and miniKinetics are used. Performance of CoViAR for these data sets are unknown. Therefore, we cannot find results where it has been shown that CoViAR outperforms IDT on data sets other than HMDB-51. Thus, we only compare our results to the results found in [125]. Our results agree with the results in [125], where CoViAR is compared, among others, to IDT. The CoViAR algorithm outperforms the algorithms DT and IDT. We further find that the motion CNN is outperformed by the residual CNN and the I-Frame CNN in most occurrences, and that CoViAR after late fusion often performs best. These findings are in agreement with results obtained in [125].

Current literature does not discuss what mounting point is optimal to detect falls from a wearable camera. Ozcan et al. [80] have chosen the waist. In their paper, it is stated that the waist is the optimal mounting point by referring to [48]. This survey paper refers to [54] where the statement is found. In this research, the authors compare fall detection algorithms for accelerometers attached to different body parts. These body parts are the wrist, head and waist. The algorithms for both head and waist obtain similar results. However, they opt to mount the accelerometer to the waist, since mounting the accelerometer to the head requires more detailed planning of the hardware. Avoiding the detailed planning ensures usability and acceptance of the application among the end-users. In our case, results for both mounting points were very similar, thus we cannot directly conclude that one mounting point is better than the other. Furthermore, we asked our selves whether classification improves when training the classifier on videos being filmed from a specific camera mounting point, instead of training the algorithm on videos independent of the camera mounting points. For the three algorithms, we did not find any improvements using this approach.

5.2 ADVANTAGES AND DISADVANTAGES OF THE PROPOSED SOLUTION

As the CoViAR method outperforms the DT and IDT methods, we propose to use CoViAR for fall detection. Besides its good performance, the algorithm only needs regular RGB-frames. It does not need to compute extra features, such as optical flow. This reduces overhead. Therefore, once final weights are obtained after training, CoViAR is a fast classifier, as investigated by Wu et al. [125]. We did not compare the processing speeds of DT and IDT to the processing speed of CoViAR, as DT and IDT were executed on the CPU and CoViAR was executed on the GPU. However, we observed that the computation of optical flow took quite some time, rendering it unable for real-time fall detection. In [125], it is stated that computation of optical flow is a bottleneck. We presume that this is the case for DT and IDT as well, as these algorithms make use of optical flow. Similar statements regarding the processing speed of optical flow are made in [20, 137].

Since CoViAR is a deep learning algorithm, it uses many different lay-

ers. Therefore, it is hard to analyze why particular videos were classified incorrectly by CoViAR. We further saw that in general, combining the CNNs in CoViAR yields the best results. However, some CNNs generated such high scores for the wrong class that as a result, the video was misclassified after late fusion, while the video was not misclassified for an individual CNN.

In this research, we found that both algorithms achieve high performance when distinguishing falls from non-falls using egocentric vision. However, the actions were performed in a 'simulation' setting. This leads to subjects being more careful when performing a fall [40, 48]. Simulated falls may be softer, as the simulators are more hesitant and do not want to risk injuries. Being more careful during a fall leads to less strong movements, as one tries to control the fall. In most of our obtained fall videos, clear falling motions are shown. Nonetheless, this is still in a controlled setting. In an uncontrolled setting, we might miss important movements occurring that did not occur during a controlled fall. Furthermore, the subjects were young adults. In reality however, it is mostly the older adults that are in danger of falling. As younger adults are often more mobile than older adults, our input data does not take into account the reduced mobility of older adults. The algorithms did not learn about the reduced mobility.

Next, as the videos are filmed both indoors and outdoors, videos have different intensities due to different lightning. This is important, as the classifiers did not focus on one particular intensity. Therefore, the risk of suddenly failing to correctly classify videos when these videos are filmed with a different intensity was reduced. Nonetheless, using merely visual input, we can not use these algorithms when there is limited or no lightning, such as in the dark or during the night.

Finally, the videos in this research showed actions being performed. The videos did not show what happened in between these actions or after these actions had ended. For instance, once a person fell, the video stopped. We note that more information can be retrieved by observing what happens after the fall. For instance, if a person quickly stands up again after a fall, no emergency help is needed. However, if a person falls and stays down or tries to get up and falls again, we are in a different situation, where one might need emergency help. This extra video information further aides in constructing a robust fall detector. The CoViAR algorithm is able to process these extra frames without too much extra additional computational cost. For example, the algorithm can process the extra frames by increasing the number of segments taken from a video.

5.3 FUTURE WORK

This research shows that falls can be detected at a high detection rate for both a deep learning method and traditional methods. As the deep learning method has better performance than the traditional methods and does not need to compute optical flow explicitly, we argue that the CoViAR algorithm is useful to detect falls. As mentioned in Section 2.2.2, we selected CoViAR as it only needs RGB-frames, does not need to compute optical flow, has high performance on other data sets and did not need a lot of computational resources, compared to other well performing deep learning algorithms in the computer vision space. In case computational resources are less of a

concern, other deep learning algorithms can be employed. We find that the motion CNN of CoViAR is often outperformed by the other two CNNs. Further, note that the motion CNN is the CNN modelling the temporal structure. From this finding, one can argue that using the motion CNN from CoViAR is not optimal for motion detection.

Zhu et al. [137] proposed a deep learning algorithm estimating motions. These motions are estimated by estimating optical flow. To estimate the optical flow, a CNN is trained with the goal to generate optical flow from a set of consecutive frames. This CNN is named MotionNet. The MotionNet is capable of approximating optical flow. MotionNet can approximate this without having the burden of computing optical flow using traditional methods, which are often slow. This leads to a huge speedup in computing optical flow, making it possible to approximate optical flow in real-time. Using this approach, RGB-frames are fed into a two-stream CNN. The first stream performs operations on the RGB-frames directly and the second stream computes optical flow using the MotionNet. The approximated optical flows are fed to the temporal CNN. Next, both streams are combined using late fusion. The authors achieved high accuracy on the benchmark data sets HMDB-51 and UCF-101. They further combined their MotionNet with TSN [121] and I3D [16] architectures. This combination further improved performance, without having to compute optical flow using traditional algorithms.

Another deep learning algorithms using a similar approach is the MARS [20] algorithm. This algorithm uses a 3D CNN architecture. Instead of learning optical flow directly, it learns the flow by modifying the loss function of the CNN such that optical flow is incorporated in the loss function. This approach is capable of processing videos at real-time, using a GPU, and achieved high performance on the data sets UCF-101, HMDB-51 and miniKinetics.

Besides examining other algorithms, we can also investigate the data that is used as input. As mentioned in Section 5.2, it would be helpful to include frames showing what happens after a fall. These extra frames add information to certain actions.

Further, this research mainly focuses on detecting falls using videos without sound as input. However, sound could be a valuable addition of information for the fall detection algorithm. For instance, a person sitting down makes a different sound than a person falling. By recognizing the sound of a fall, the fall detector could improve its performance by incorporating this knowledge. In [65, 138] sound sensing and floor vibration was used as data to recognize falls. In these researches, it was found that sound is an important factor to determine a fall.

Furthermore, accelerometers can be combined with the wearable camera to further increase performance. For instance, Ozcan et al. [79] combined video data with accelerometer data. They found that adding the accelerometer data improved performance of the algorithm. A lot of research is conducted to detect falls, using accelerometers. For instance, [14, 54, 62, 114, 134] used accelerometers to detect falls. These researches all concluded that accelerometers are useful to detect falls.

Hence, adding sound and accelerometer data might further help improving the performance of the fall detector and its data might be helpful in cases where lightning is not available, such as during the night.

6 CONCLUSION

In this research, we have utilized different algorithms to detect falls with videos filmed from a wearable camera. These algorithms were further used in a multi-class setting. Two traditional algorithms were proposed. These algorithms are DT and IDT. Further, one deep learning algorithm was proposed. This algorithm is CoViAR. We explained why these algorithms are useful for fall detection and useful for motion detection in general. Next, a detailed description of the algorithms was given. After describing the algorithms, the experimental settings were specified. Running the different algorithms, multiple results were obtained. The best performing algorithm was the CoViAR algorithm, for all classification problems. This was verified by evaluating different performance metrics. We further observed that the traditional algorithms also performed well. Given our first research question:

- Can the proposed algorithms be used for fall detection, taking false negatives and false positives into account?

We confirm that this is indeed the case. All proposed algorithms can be used for fall detection, given the low number of false negatives and low number of false positives after predicting the videos in the test set. Since the proposed algorithms have good performance, they can be used for fall detection.

Since the proposed algorithms can be used for fall detection, we asked ourselves which algorithm would perform better. Recent research suggested that the deep learning algorithm would perform better, given the results obtained in those researches. With this information, we constructed the research question:

- Does the deep learning algorithm outperform the traditional algorithms when detecting falls with a wearable camera?

Investigating our results, we can indeed verify that this is the case. CoViAR outperforms both DT and IDT for fall detection. Thus the deep learning algorithm outperforms the traditional algorithms with the wearable camera. Hence, we prefer to use the CoViAR algorithm when detecting the falls.

Next, we investigated the mounting position of the camera. We found that the CoViAR algorithm outperforms DT and IDT for both camera mounting points. We did not find whether using one mounting position is preferred over the other, given the results in this research. Therefore, given our third research question:

- Is there a difference in performance of the classification algorithms for fall detection when mounting the camera to the neck or waist?

We cannot find evidence that there is a difference in performance when mounting the camera to the neck or waist. Further, we investigated whether training on videos filmed from a specific mounting point would improve the performance of the classifiers, compared to training the classifiers on videos filmed from both camera mounting points. We did not find evidence that the performance improved. The classifier has less data when being trained

on videos filmed by a specific camera mounting point. Having less data could be a reason that the performance stagnates when comparing it to the performance obtained when training the classifiers on videos filmed from cameras independent of the mounting position. However, more data would be needed to verify this claim.

Our last investigation was done by comparing the performance of the algorithms when using more classes instead of the classes falls and non-falls. We constructed the following research question:

- How do the algorithms compare in terms of performance when taking into account the different classes of the data set?

We find that CoViAR after late fusion outperforms both DT and IDT in terms of performance. This is in agreement with the results for the binary problems and also in agreement with the literature, where deep learning algorithms such as CoViAR outperformed the traditional algorithms such as DT and IDT.

Having answered our research questions and performed analysis, we conclude that the selected algorithms are capable of distinguishing falls from non-falls, when videos are filmed with a wearable camera. We found no evidence whether there is an optimal camera mounting point. Given the performance and properties of CoViAR, we prefer to use this algorithm over DT and IDT for fall detection. This algorithm has the potential to be used in fall detectors that make use of information retrieved from wearable cameras mounted to the neck or waist.

BIBLIOGRAPHY

[1] G. Abebe and A. Cavallaro. "A long short-term memory convolutional neural network for first-person vision activity recognition". In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2017, pp. 1339–1346.

[2] G. Abebe and A. Cavallaro. "Inertial-Vision: cross-domain knowledge transfer for wearable sensors". In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2017, pp. 1392–1400.

[3] Aphex34. *typical CNN architecture — Wikipedia, The Free Encyclopedia*. 2015. URL: https://commons.wikimedia.org/wiki/File:Typical_cnn.png (visited on Jan. 5, 2021). License: CC BY-SA 4.0.

[4] F. Bagalà et al. "Evaluation of accelerometer-based fall detection algorithms on real-world falls". In: *PloS one* 7.5 (2012), e37062.

[5] S. Bambach. "A survey on recent advances of computer vision algorithms for egocentric video". In: *arXiv preprint arXiv:1501.02825* (2015).

[6] H. Bay, T. Tuytelaars, and L. Van Gool. "Surf: Speeded up robust features". In: *European conference on computer vision*. Springer. 2006, pp. 404–417.

[7] H. Bay et al. "Speeded-up robust features (SURF)". In: *Computer vision and image understanding* 110.3 (2008), pp. 346–359.

[8] N. El-Bendary et al. "FALL DETECTION AND PREVENTION FOR THE ELDERLY: A REVIEW OF TRENDS AND CHALLENGES." In: *International Journal on Smart Sensing & Intelligent Systems* 6.3 (2013).

[9] A. Betancourt et al. "The evolution of first person vision methods: A survey". In: *IEEE Transactions on Circuits and Systems for Video Technology* 25.5 (2015), pp. 744–760.

[10] B. E. Boser, I. M. Guyon, and V. N. Vapnik. "A training algorithm for optimal margin classifiers". In: *Proceedings of the fifth annual workshop on Computational learning theory*. 1992, pp. 144–152.

[11] A. L. Boskey and R. Coleman. "Aging and bone". In: *Journal of dental research* 89.12 (2010), pp. 1333–1348.

[12] I. Boudouane et al. "Fall detection system with portable camera". In: *Journal of Ambient Intelligence and Humanized Computing* (2019), pp. 1–13.

[13] I. Boudouane et al. "Wearable camera for fall detection embedded system". In: *Proceedings of the 4th International Conference on Smart City Applications*. 2019, pp. 1–6.

[14] A. Bourke, J. O'brien, and G. Lyons. "Evaluation of a threshold-based tri-axial accelerometer fall detection algorithm". In: *Gait & posture* 26.2 (2007), pp. 194–199.

[15] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[16] J. Carreira and A. Zisserman. "Quo vadis, action recognition? a new model and the kinetics dataset". In: *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 6299–6308.

[17] M. Casares et al. "Automatic fall detection by a wearable embedded smart camera". In: *2012 Sixth International Conference on Distributed Smart Cameras (ICDSC)*. IEEE. 2012, pp. 1–6.

[18] D. Castro et al. "Predicting daily activities from egocentric images using deep learning". In: *proceedings of the 2015 ACM International symposium on Wearable Computers*. 2015, pp. 75–82.

[19] K.-H. Chen et al. "Evaluating the specifications of built-in accelerometers in smartphones on fall detection performance". In: *Instrumentation Science & Technology* 46.2 (2018), pp. 194–206.

[20] N. Crasto et al. "Mars: Motion-augmented rgb stream for action recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2019, pp. 7882–7891.

[21] N. Dalal and B. Triggs. "Histograms of oriented gradients for human detection". In: *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*. Vol. 1. IEEE. 2005, pp. 886–893.

[22] N. Dalal, B. Triggs, and C. Schmid. "Human detection using oriented histograms of flow and appearance". In: *European conference on computer vision*. Springer. 2006, pp. 428–441.

[23] A. G. Del Molino et al. "Summarization of egocentric videos: A comprehensive survey". In: *IEEE Transactions on Human-Machine Systems* 47.1 (2016), pp. 65–76.

[24] Y. S. Delahoz and M. A. Labrador. "Survey on fall detection and fall prevention using wearable and external sensors". In: *Sensors* 14.10 (2014), pp. 19806–19842.

[25] A. P. Dempster, N. M. Laird, and D. B. Rubin. "Maximum likelihood from incomplete data via the EM algorithm". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 39.1 (1977), pp. 1–22.

[26] J. Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[27] J. Donahue et al. "Long-term recurrent convolutional networks for visual recognition and description". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 2625–2634.

[28] S. Elliott, J. Painter, and S. Hudson. "Living alone and fall risk factors in community-dwelling middle age and older adults". In: *Journal of community health* 34.4 (2009), p. 301.

[29] Ennepetaler86. *Two classes given as vectors with two possible separation lines and their corresponding margin areas between the class areas. Line A has a larger empty margin area than line B. — Wikipedia, The Free Encyclopedia*. 2010. URL: https://commons.wikimedia.org/wiki/File:Svm_intro.svg (visited on Jan. 5, 2021). License: CC BY 3.0.

[30] C. Fan and D. J. Crandall. "Deepdiary: Automatically captioning lifelogging image streams". In: *European Conference on Computer Vision*. Springer. 2016, pp. 459–473.

[31] G. Farnebäck. "Two-frame motion estimation based on polynomial expansion". In: *Scandinavian conference on Image analysis*. Springer. 2003, pp. 363–370.

[32] A. Fathi, Y. Li, and J. M. Rehg. "Learning to recognize daily actions using gaze". In: *European Conference on Computer Vision*. Springer. 2012, pp. 314–327.

[33] T. Fawcett. "ROC graphs: Notes and practical considerations for researchers". In: (2004).

[34] C. Feichtenhofer, A. Pinz, and A. Zisserman. "Convolutional two-stream network fusion for video action recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 1933–1941.

[35] M. A. Fischler and R. C. Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". In: *Communications of the ACM* 24.6 (1981), pp. 381–395.

[36] K. Gaßner and M. Conrad. "ICT enabled independent living for elderly". In: *A status-quo analysis on products and the research landscape in the field of Ambient Assisted Living (AAL) in EU-27. VDI* (2010).

[37] X. Glorot, A. Bordes, and Y. Bengio. "Deep sparse rectifier neural networks". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, pp. 315–323.

[38] L. Gorelick et al. "Actions as space-time shapes". In: *IEEE transactions on pattern analysis and machine intelligence* 29.12 (2007), pp. 2247–2253.

[39] D. Graham et al. "Convolutional drift networks for video classification". In: *2017 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE. 2017, pp. 1–8.

[40] M. A. Habib et al. "Smartphone-based solutions for fall detection and prevention: challenges and open issues". In: *Sensors* 14.4 (2014), pp. 7181–7208.

[41] T. Hadjistavropoulos, K. Delbaere, and T. D. Fitzgerald. "Reconceptualizing the role of fear of falling and balance confidence in fall risk". In: *Journal of aging and Health* 23.1 (2011), pp. 3–23.

[42] A. Hamid, A. Brahim, O. Mohammed, et al. "A survey of activity recognition in egocentric lifelogging datasets". In: *2017 International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS)*. IEEE. 2017, pp. 1–8.

[43] D. J. Hand and R. J. Till. "A simple generalisation of the area under the ROC curve for multiple class classification problems". In: *Machine learning* 45.2 (2001), pp. 171–186.

[44] K. He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[45] F. Hijaz et al. "Survey of fall detection and daily activity monitoring techniques". In: *2010 International Conference on Information and Emerging Technologies*. IEEE. 2010, pp. 1–6.

[46] C.-W. Hsu, C.-C. Chang, C.-J. Lin, et al. *A practical guide to support vector classification*. 2003.

[47] S. Hua and Z. Sun. "Support vector machine approach for protein subcellular localization prediction". In: *Bioinformatics* 17.8 (2001), pp. 721–728.

[48] R. Igual, C. Medrano, and I. Plaza. "Challenges, issues and trends in fall detection systems". In: *Biomedical engineering online* 12.1 (2013), p. 66.

[49] S. Ioffe and C. Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167* (2015).

[50] Y. Iwashita et al. "First-person animal activity recognition from egocentric videos". In: *2014 22nd International Conference on Pattern Recognition*. IEEE. 2014, pp. 4310–4315.

[51] Y.-G. Jiang et al. "Trajectory-based modeling of human actions with motion reference points". In: *European Conference on Computer Vision*. Springer. 2012, pp. 425–438.

[52] R. Kahani, A. Talebpour, and A. Mahmoudi-Aznaveh. "Time series correlation for first-person videos". In: *2016 24th Iranian Conference on Electrical Engineering (ICEE)*. IEEE. 2016, pp. 805–809.

[53] M. Kalfaoglu, S. Kalkan, and A. A. Alatan. "Late Temporal Modeling in 3D CNN Architectures with BERT for Action Recognition". In: *arXiv preprint arXiv:2008.01232* (2020).

[54] M. Kangas et al. "Comparison of low-complexity fall detection algorithms for body attached accelerometers". In: *Gait & posture* 28.2 (2008), pp. 285–291.

[55] D. P. Kingma and J. Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[56] A. Klaser, M. Marszałek, and C. Schmid. "A spatio-temporal descriptor based on 3d-gradients". In: 2008.

[57] O. Kliper-Gross, T. Hassner, and L. Wolf. "The action similarity labeling challenge". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.3 (2011), pp. 615–621.

[58] A. KROONENBERG. "Hip impact velocities and body configurations for experimental falls from standing height". In: *39th Annual Meeting, Orthopaedic Research Society*. 1993.

[59] H. Kuehne et al. "HMDB: a large video database for human motion recognition". In: *2011 International Conference on Computer Vision*. IEEE. 2011, pp. 2556–2563.

[60] I. Laptev et al. "Learning realistic human actions from movies". In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2008, pp. 1–8.

[61] Y. J. Lee, J. Ghosh, and K. Grauman. "Discovering important people and objects for egocentric video summarization". In: *2012 IEEE conference on computer vision and pattern recognition*. IEEE. 2012, pp. 1346–1353.

[62] Q. Li et al. "Accurate, fast fall detection using gyroscopes and accelerometer-derived posture information". In: *2009 Sixth International Workshop on Wearable and Implantable Body Sensor Networks*. IEEE. 2009, pp. 138–143.

[63] Y. Li, Z. Ye, and J. M. Rehg. "Delving into egocentric actions". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 287–295.

[64] R. Ling. "Exclusion or self-isolation? Texting and the elderly users". In: *The information society* 24.5 (2008), pp. 1–9.

[65] D. Litvak, Y. Zigel, and I. Gannot. "Fall detection of elderly through floor vibrations and sound". In: *2008 30th annual international conference of the IEEE engineering in medicine and biology society*. IEEE. 2008, pp. 4632–4635.

[66] S. Lloyd. "Least squares quantization in PCM". In: *IEEE transactions on information theory* 28.2 (1982), pp. 129–137.

[67] I. Loshchilov and F. Hutter. "Decoupled weight decay regularization". In: *arXiv preprint arXiv:1711.05101* (2017).

[68] M. Lu et al. "Deep attention network for egocentric action recognition". In: *IEEE Transactions on Image Processing* 28.8 (2019), pp. 3703–3713.

[69] M. Ma, H. Fan, and K. M. Kitani. "Going deeper into first-person activity recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 1894–1903.

[70] M. Marszalek, I. Laptev, and C. Schmid. "Actions in context". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2009, pp. 2929–2936.

[71] T. P. Moreira, D. Menotti, and H. Pedrini. "First-person action recognition through visual rhythm texture description". In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2017, pp. 2627–2631.

[72] M. Mubashir, L. Shao, and L. Seed. "A survey on fall detection: Principles and approaches". In: *Neurocomputing* 100 (2013), pp. 144–152.

[73] H. Nait-Charif and S. J. McKenna. "Activity summarisation and fall detection in a supportive home environment". In: *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.* Vol. 4. IEEE. 2004, pp. 323–326.

[74] T.-H.-C. Nguyen, J.-C. Nebel, F. Florez-Revuelta, et al. "Recognition of activities of daily living with egocentric vision: A review". In: *Sensors* 16.1 (2016), p. 72.

[75] J. C. Niebles, C.-W. Chen, and L. Fei-Fei. "Modeling temporal structure of decomposable motion segments for activity classification". In: *European conference on computer vision*. Springer. 2010, pp. 392–405.

[76] J. C. Niebles, H. Wang, and L. Fei-Fei. "Unsupervised learning of human action categories using spatial-temporal words". In: *International journal of computer vision* 79.3 (2008), pp. 299–318.

[77] W. H. Organization, W. H. O. Ageing, and L. C. Unit. *WHO global report on falls prevention in older age*. World Health Organization, 2008.

[78] K. Ozcan, A. K. Mahabalagiri, and S. Velipasalar. "Fall detection and activity classification using a wearable smart camera". In: *2013 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE. 2013, pp. 1–6.

[79] K. Ozcan and S. Velipasalar. "Wearable camera-and accelerometer-based fall detection on portable devices". In: *IEEE Embedded Systems Letters* 8.1 (2015), pp. 6–9.

[80] K. Ozcan, S. Velipasalar, and P. K. Varshney. "Autonomous fall detection with wearable cameras by using relative entropy distance measure". In: *IEEE Transactions on Human-Machine Systems* 47.1 (2016), pp. 31–39.

[81] K. Ozcan et al. "Automatic fall detection and activity classification by a wearable embedded smart camera". In: *IEEE journal on emerging and selected topics in circuits and systems* 3.2 (2013), pp. 125–136.

[82] F. Özkan et al. "Boosted multiple kernel learning for first-person activity recognition". In: *2017 25th European Signal Processing Conference (EUSIPCO)*. IEEE. 2017, pp. 1050–1054.

[83] A. Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[84] K. Pearson. "LIII. On lines and planes of closest fit to systems of points in space". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572.

[85] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[86] X. Peng et al. "Action recognition with stacked fisher vectors". In: *European Conference on Computer Vision*. Springer. 2014, pp. 581–595.

[87] X. Peng et al. "Bag of visual words and fusion methods for action recognition: Comprehensive study and good practice". In: *Computer Vision and Image Understanding* 150 (2016), pp. 109–125.

[88] A. J. Perez, S. Zeadally, and S. Griffith. "Bystanders' privacy". In: *IT Professional* 19.3 (2017), pp. 61–65.

[89] F. Perronnin, J. Sánchez, and T. Mensink. "Improving the fisher kernel for large-scale image classification". In: *European conference on computer vision*. Springer. 2010, pp. 143–156.

[90] J. T. Perry et al. "Survey and evaluation of real-time fall detection approaches". In: *2009 6th International Symposium on High Capacity Optical Networks and Enabling Technologies (HONET)*. IEEE. 2009, pp. 158–164.

[91] A. Piergiovanni, C. Fan, and M. S. Ryoo. "Learning latent sub-events in activity videos using temporal attention filters". In: *arXiv preprint arXiv:1605.08140* (2016).

[92] H. Pirsiavash and D. Ramanan. "Detecting activities of daily living in first-person camera views". In: *2012 IEEE conference on computer vision and pattern recognition*. IEEE. 2012, pp. 2847–2854.

[93] Y. Poleg, C. Arora, and S. Peleg. "Temporal segmentation of egocentric videos". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 2537–2544.

[94] Y. Poleg et al. "Compact cnn for indexing egocentric videos". In: *2016 IEEE winter conference on applications of computer vision (WACV)*. IEEE. 2016, pp. 1–9.

[95] D. Purwanto, Y.-T. Chen, and W.-H. Fang. "Temporal aggregation for first-person action recognition using Hilbert-Huang transform". In: *2017 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE. 2017, pp. 895–900.

[96] I. Putra et al. "An event-triggered machine learning approach for accelerometer-based fall detection". In: *Sensors* 18.1 (2018), p. 20.

[97] K. K. Reddy and M. Shah. "Recognizing 50 human action categories of web videos". In: *Machine vision and applications* 24.5 (2013), pp. 971–981.

[98] I. E. Richardson. *H. 264 and MPEG-4 video compression: video coding for next-generation multimedia*. John Wiley & Sons, 2004.

[99] I. Rodrıéguez-Moreno et al. "Video activity recognition: State-of-the-art". In: *Sensors* 19.14 (2019), p. 3160.

[100] C. Rougier et al. "Monocular 3D head tracking to detect falls of elderly people". In: *2006 international conference of the IEEE engineering in medicine and biology society*. IEEE. 2006, pp. 6384–6387.

[101] M. S. Ryoo, B. Rothrock, and L. Matthies. "Pooled motion features for first-person videos". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 896–904.

[102] S. Sadanand and J. J. Corso. "Action bank: A high-level representation of activity in video". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2012, pp. 1234–1241.

[103] S. Sadigh et al. "Falls and fall-related injuries among the elderly: a survey of residential-care facilities in a Swedish municipality". In: *Journal of community health* 29.2 (2004), pp. 129–140.

[104] C. Schuldt, I. Laptev, and B. Caputo. "Recognizing human actions: a local SVM approach". In: *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.* Vol. 3. IEEE. 2004, pp. 32–36.

[105] J. Shi et al. "Good features to track". In: *1994 Proceedings of IEEE conference on computer vision and pattern recognition*. IEEE. 1994, pp. 593–600.

[106] S. Singh, C. Arora, and C. Jawahar. "First person action recognition using deep learned descriptors". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2620–2628.

[107] S. Singh, C. Arora, and C. Jawahar. "Trajectory aligned features for first person action recognition". In: *Pattern Recognition* 62 (2017), pp. 45–55.

[108] I. Sobel and G. Feldman. "A 3x3 isotropic gradient operator for image processing". In: *a talk at the Stanford Artificial Project in* (1968), pp. 271–272.

[109] K. Soomro, A. R. Zamir, and M. Shah. "UCF101: A dataset of 101 human actions classes from videos in the wild". In: *arXiv preprint arXiv:1212.0402* (2012).

[110] E. H. Spriggs, F. D. L. Torre, and M. Hebert. *Temporal Segmentation and Activity Classification from First-person Sensing*. 2009.

[111] E. H. Spriggs, F. De La Torre, and M. Hebert. "Temporal segmentation and activity classification from first-person sensing". In: *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. IEEE. 2009, pp. 17–24.

[112] S. Sudhakaran, S. Escalera, and O. Lanz. "Lsta: Long short-term attention for egocentric action recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 9954–9963.

[113] A. Takamine, Y. Iwashita, and R. Kurazume. "First-person activity recognition with C3D features from optical flow images". In: *2015 IEEE/SICE International Symposium on System Integration (SII)*. IEEE. 2015, pp. 619–622.

[114] L. Tong et al. "HMM-based human fall detection and prediction method using tri-axial accelerometer". In: *IEEE Sensors Journal* 13.5 (2013), pp. 1849–1856.

[115] D. Tran et al. "A closer look at spatiotemporal convolutions for action recognition". In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2018, pp. 6450–6459.

[116] D. Tran et al. "Learning spatiotemporal features with 3d convolutional networks". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 4489–4497.

[117] A. Vedaldi and B. Fulkerson. *VLFeat: An Open and Portable Library of Computer Vision Algorithms*. http://www.vlfeat.org/. 2008.

[118] H. Wang and C. Schmid. "Action recognition with improved trajectories". In: *Proceedings of the IEEE international conference on computer vision*. 2013, pp. 3551–3558.

[119] H. Wang et al. "Action recognition by dense trajectories". In: *CVPR 2011*. IEEE. 2011, pp. 3169–3176.

[120] H. Wang et al. "Dense trajectories and motion boundary descriptors for action recognition". In: *International journal of computer vision* 103.1 (2013), pp. 60–79.

[121] L. Wang et al. "Temporal segment networks: Towards good practices for deep action recognition". In: *European conference on computer vision*. Springer. 2016, pp. 20–36.

[122] X. Wang et al. "Deep appearance and motion learning for egocentric activity recognition". In: *Neurocomputing* 275 (2018), pp. 438–447.

[123] X. Wang, J. Ellul, and G. Azzopardi. "Elderly fall detection systems: A literature survey". In: *Front. Robot. AI* 7 (2020), p. 71.

[124] J. Willems et al. "How to detect human fall in video? An overview". In: *Positioning and context-awareness international conference-POCA 2009, Date: 2009/05/28-2009/05/28, Location: Antwerp, Belgium*. 2009.

[125] C.-Y. Wu et al. "Compressed video action recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 6026–6035.

[126] G. Wu. "Distinguishing fall activities from normal activities by velocity characteristics". In: *Journal of biomechanics* 33.11 (2000), pp. 1497–1500.

[127] M. Xu et al. "Fully-coupled two-stream spatiotemporal networks for extremely low resolution action recognition". In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2018, pp. 1607–1615.

[128] T. Xu, Y. Zhou, and J. Zhu. "New advances and challenges of fall detection systems: A survey". In: *Applied Sciences* 8.3 (2018), p. 418.

[129] Y. Yan et al. "Egocentric daily activity recognition via multitask clustering". In: *IEEE Transactions on Image Processing* 24.10 (2015), pp. 2984–2995.

[130] G. Yavuz et al. "A smartphone based fall detector with online location support". In: *International Workshop on Sensing for App Phones; Zurich, Switzerland*. 2010, pp. 31–35.

[131] B. Yin et al. "Indirect human activity recognition based on optical flow method". In: *2012 5th International Congress on Image and Signal Processing*. IEEE. 2012, pp. 99–103.

[132] H. F. Zaki, F. Shafait, and A. Mian. "Modeling sub-event dynamics in first-person action recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 7253–7262.

[133] K. Zhan, F. Ramos, and S. Faux. "Activity recognition from a wearable camera". In: *2012 12th International Conference on Control Automation Robotics & Vision (ICARCV)*. IEEE. 2012, pp. 365–370.

[134] T. Zhang et al. "Fall detection by embedding an accelerometer in cellphone and using KFD algorithm". In: *International Journal of Computer Science and Network Security* 6.10 (2006), pp. 277–284.

[135] Z. Zhang, C. Conly, and V. Athitsos. "A survey on vision-based fall detection". In: *Proceedings of the 8th ACM international conference on PErvasive technologies related to assistive environments*. 2015, pp. 1–7.

[136] Y. Zhou et al. "Cascaded interactional targeting network for egocentric video analysis". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 1904–1913.

[137] Y. Zhu et al. "Hidden two-stream convolutional networks for action recognition". In: *Asian Conference on Computer Vision*. Springer. 2018, pp. 363–378.

[138] Y. Zigel, D. Litvak, and I. Gannot. "A method for automatic fall detection of elderly people using floor vibrations and sound—Proof of concept on human mimicking doll falls". In: *IEEE transactions on biomedical engineering* 56.12 (2009), pp. 2858–2867.

A

REMAINING RESULTS

A.1 TRAINING RESULTS

A.1.1 *ROC-AUC obtained after performing cross-validation on the* CoViAR *CNNs*

CNN / Learning rate	0.0003	0.001	0.005
I-frame	99.59 (0.30)	**99.74 (0.26)**	99.64 (0.32)
Motion	93.05 (3.40)	94.85 (2.47)	**96.65 (1.69)**
Residual	98.49 (1.07)	99.08 (0.77)	**99.25 (0.63)**

Table 15: Cross validation results under different hyperparameter settings for CoViAR.

In Table 15 we show the average ROC-AUC of each CNN for different learning rates. Standard deviations of the average AUC are given in parentheses. The selected hyperparameters are selected based on the highest ROC-AUC. The highest ROC-AUC is depicted in bold.

A.1.2 *Learned weights of* CoViAR *after training on* fall_non-fall_all-cameras

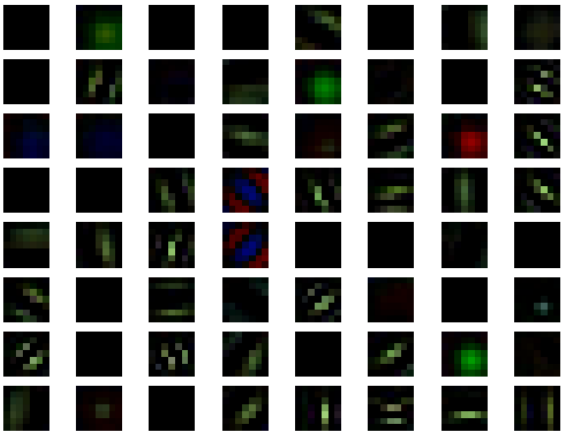

Figure 25: Weights learned by the CoViAR I-frame CNN.

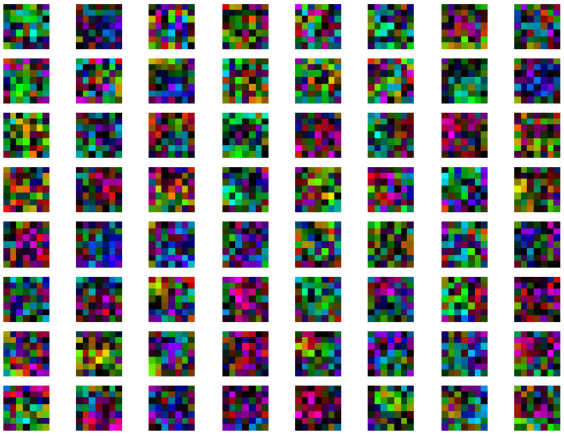

Figure 26: Weights learned by the CoViAR motion CNN.

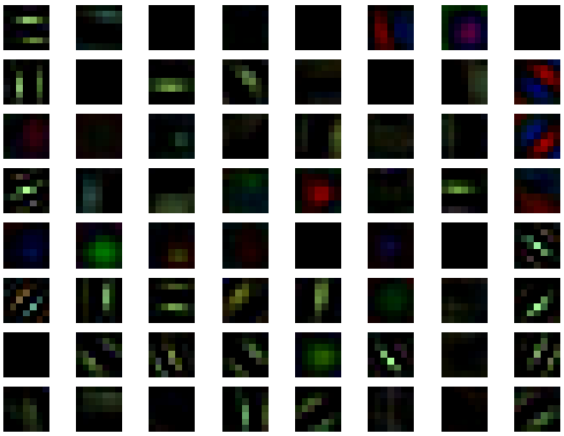

Figure 27: Weights learned by the CoViAR residual CNN.

A.2 TEST RESULTS

A.2.1 fall_non-fall_all-cameras

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	152	12	164
	Fall	15	114	129
	Total	167	126	293

Table 16: Confusion matrix after executing DT on fall_non-fall_all-cameras.

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	158	6	164
	Fall	13	116	129
	Total	171	122	293

Table 17: Confusion matrix after executing IDT on fall_non-fall_all-cameras.

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	159	5	164
	Fall	1	128	129
	Total	160	133	293

Table 18: Confusion matrix after executing the CoViAR I-frame CNN on fall_non-fall_all-cameras.

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	155	9	164
	Fall	2	127	129
	Total	157	136	293

Table 19: Confusion matrix after executing the CoViAR motion CNN on fall_non-fall_all-cameras.

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	159	5	164
	Fall	5	124	129
	Total	164	129	293

Table 20: Confusion matrix after executing the CoViAR residual CNN on fall_non-fall_all-cameras.

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	159	5	164
	Fall	1	128	129
	Total	160	133	293

Table 21: Confusion matrix after executing the CoViAR after late fusion on fall_non-fall_all-cameras.

A.2.2 fall_non-fall_neck_1

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	72	7	79
	Fall	2	65	67
	Total	74	72	146

Table 22: Confusion matrix after executing DT on fall_non-fall_neck_1.

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	70	9	79
	Fall	3	64	67
	Total	73	73	146

Table 23: Confusion matrix after executing IDT on fall_non-fall_neck_1.

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	76	3	79
	Fall	3	64	67
	Total	79	67	146

Table 24: Confusion matrix after executing the CoViAR I-frame CNN on fall_non-fall_neck_1.

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	72	7	79
	Fall	1	66	67
	Total	73	73	146

Table 25: Confusion matrix after executing the CoViAR motion CNN on fall_non-fall_neck_1.

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	79	0	79
	Fall	1	66	67
	Total	80	66	146

Table 26: Confusion matrix after executing the CoViAR residual CNN on fall_non-fall_neck_1.

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	78	1	79
	Fall	2	65	67
	Total	80	66	146

Table 27: Confusion matrix after executing the CoViAR after late fusion on fall_non-fall_neck_1.

A.2.3 fall_non-fall_waist_1

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	79	6	85
	Fall	6	58	64
	Total	85	64	149

Table 28: Confusion matrix after executing DT on fall_non-fall_waist_1.

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	83	2	85
	Fall	6	58	64
	Total	89	60	149

Table 29: Confusion matrix after executing IDT on fall_non-fall_waist_1.

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	83	2	85
	Fall	0	64	64
	Total	83	66	149

Table 30: Confusion matrix after executing the CoViAR I-frame CNN on fall_non-fall_waist_1.

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	75	10	85
	Fall	0	64	64
	Total	75	74	149

Table 31: Confusion matrix after executing the CoViAR motion CNN on fall_non-fall_waist_1.

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	83	2	85
	Fall	0	64	64
	Total	83	66	149

Table 32: Confusion matrix after executing the CoViAR residual CNN on fall_non-fall_waist_1.

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	83	2	85
	Fall	0	64	64
	Total	83	66	149

Table 33: Confusion matrix after executing CoViAR after late fusion on fall_non-fall_waist_1.

A.2.4 all-classes_all_cameras_1

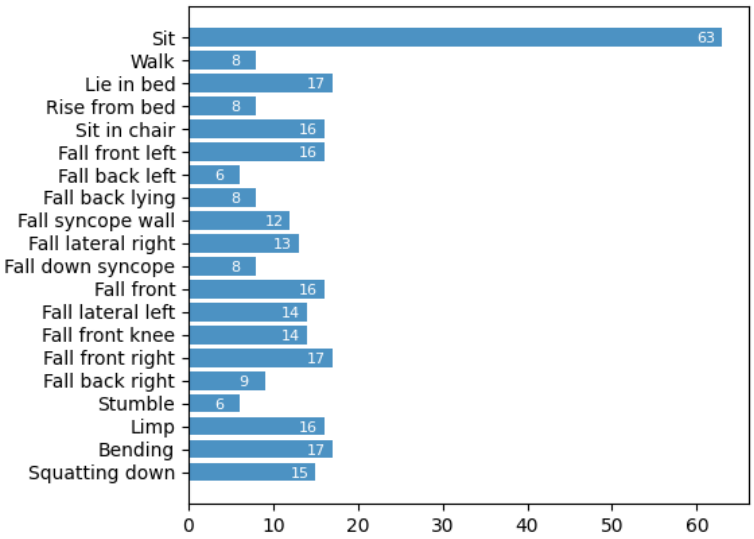

Figure 28: Distribution of the test set of classes of all-classes_all_cameras_1

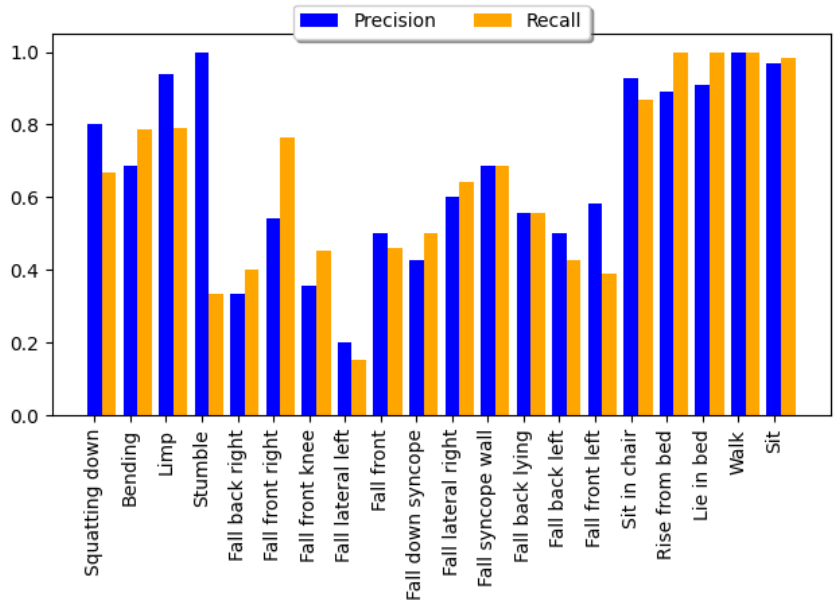

Figure 29: Precision and recall of the different classes after training DT on all-classes_all_cameras_1.

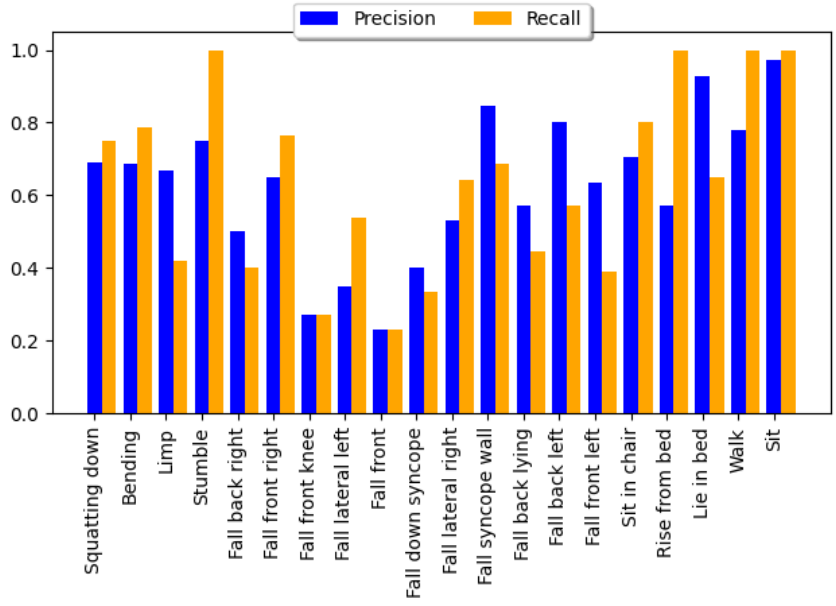

Figure 30: Precision and recall of the different classes after training IDT on all-classes_all_cameras_1.

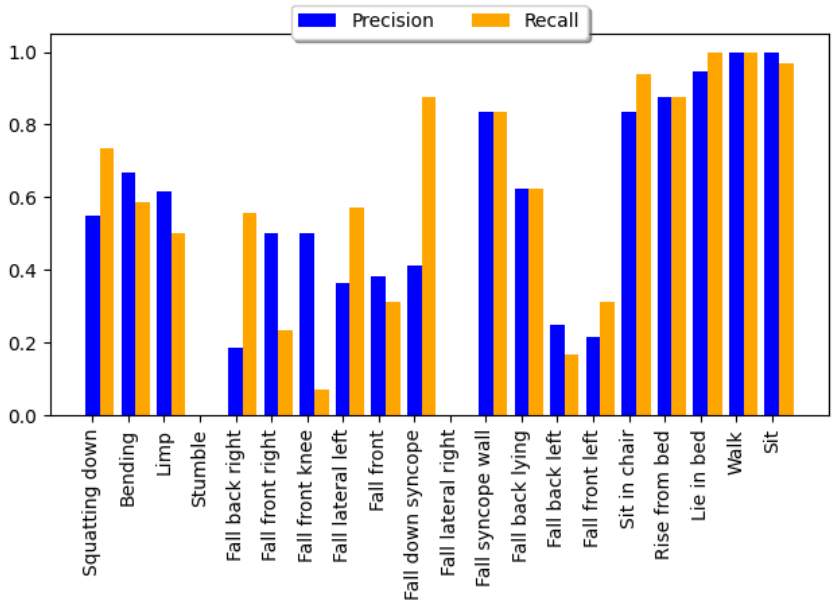

Figure 31: Precision and recall of the different classes after training the CoViAR I-frame CNN on all-classes_all_cameras_1.

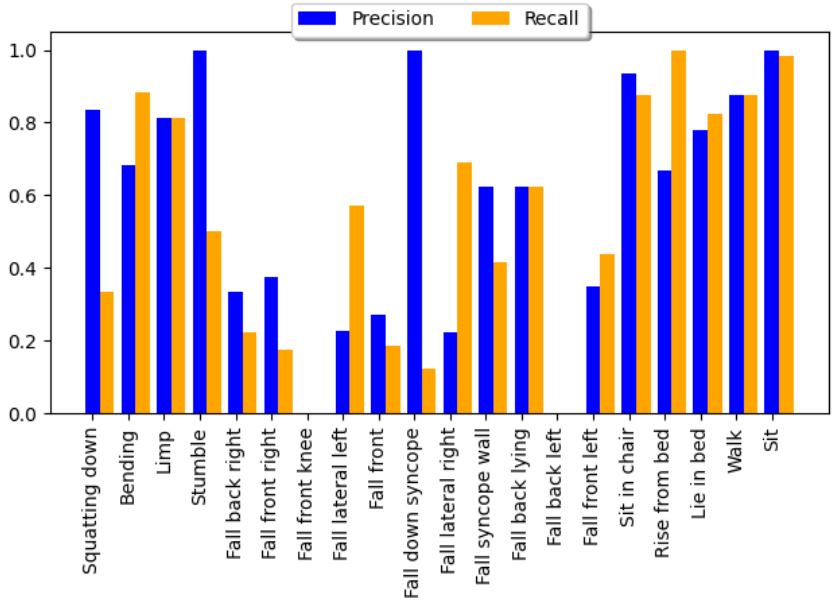

Figure 32: Precision and recall of the different classes after training the CoViAR motion CNN on all-classes_all_cameras_1.

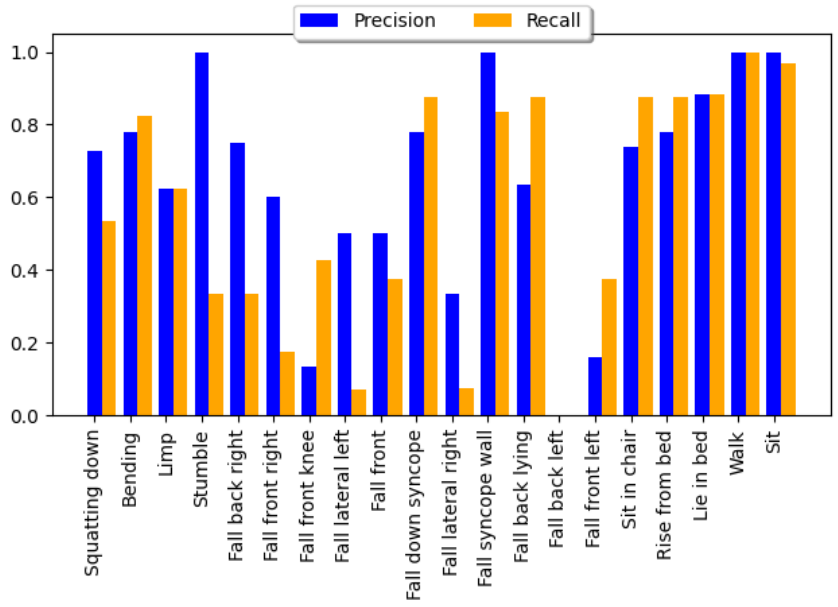

Figure 33: Precision and recall of the different classes after training the CoViAR residual CNN on all-classes_all_cameras_1.

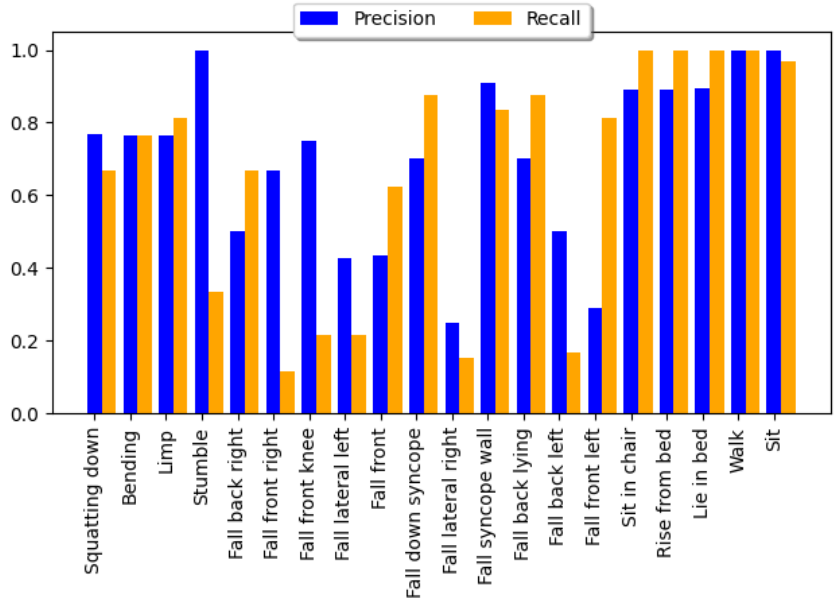

Figure 34: Precision and recall of the different classes after training CoViAR after late fusion on all-classes_all_cameras_1.

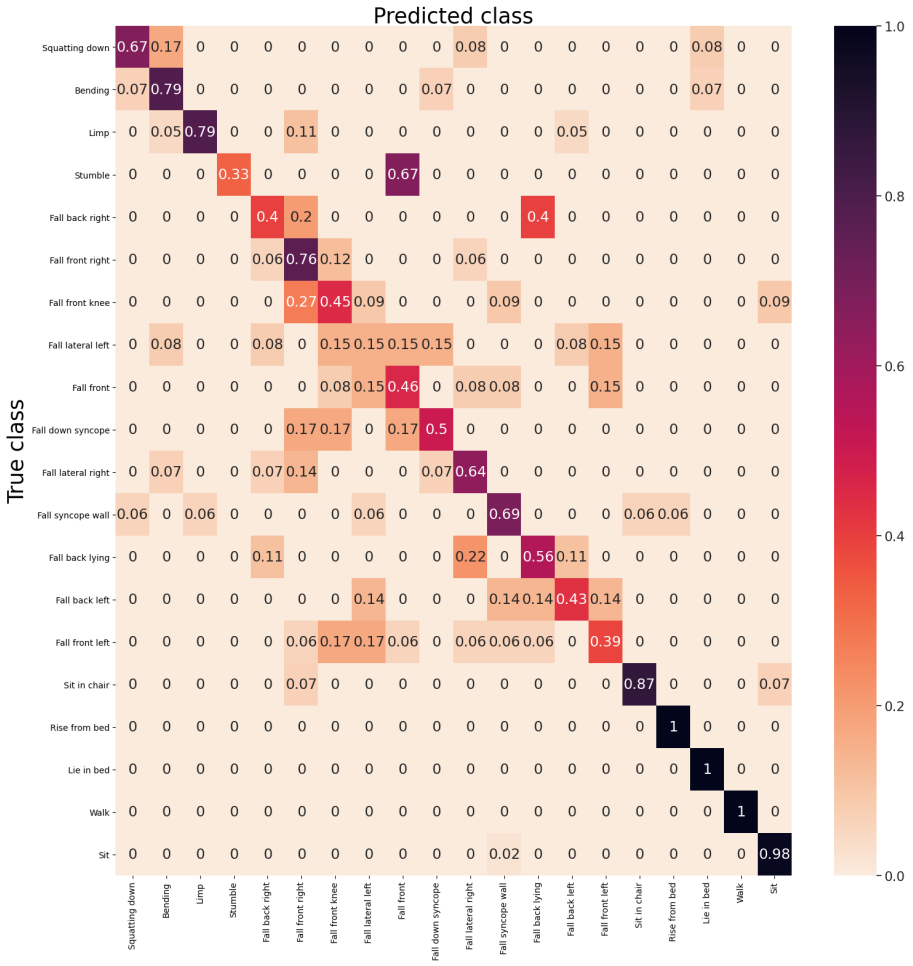

Figure 35: Normalized confusion matrix after executing DT on all-classes_all_cameras_1.

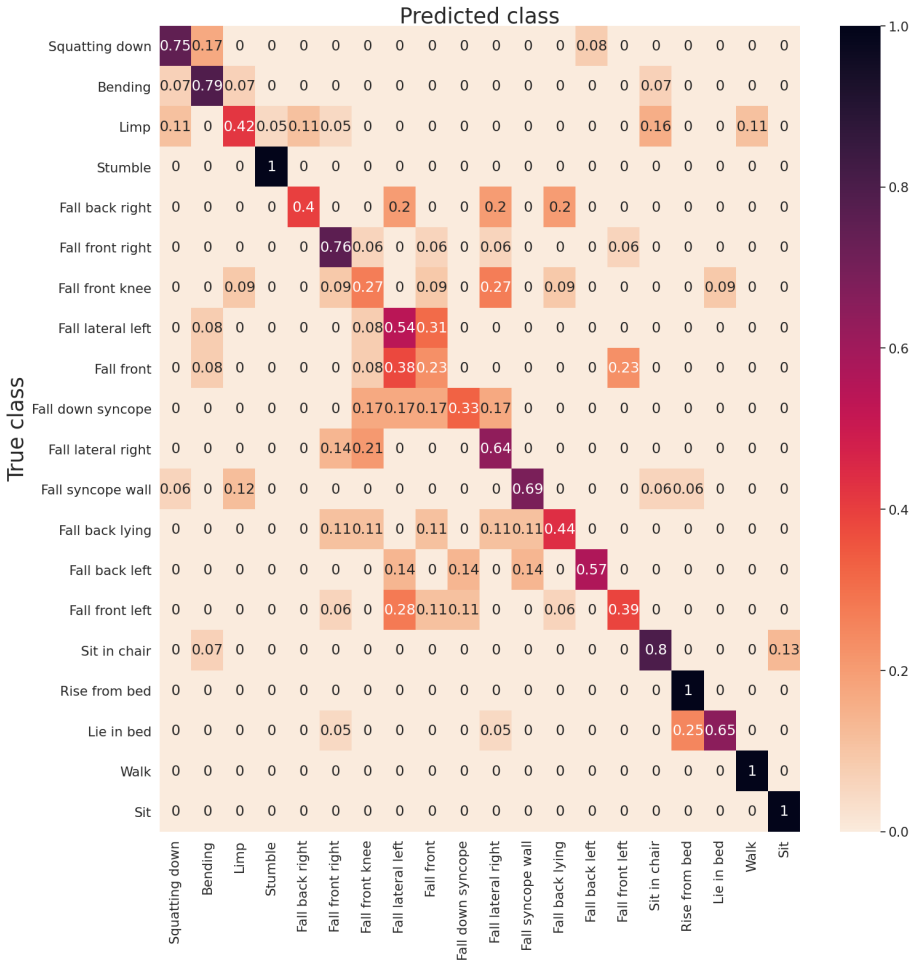

Figure 36: Normalized confusion matrix after executing IDT on all-classes_all_cameras_1.

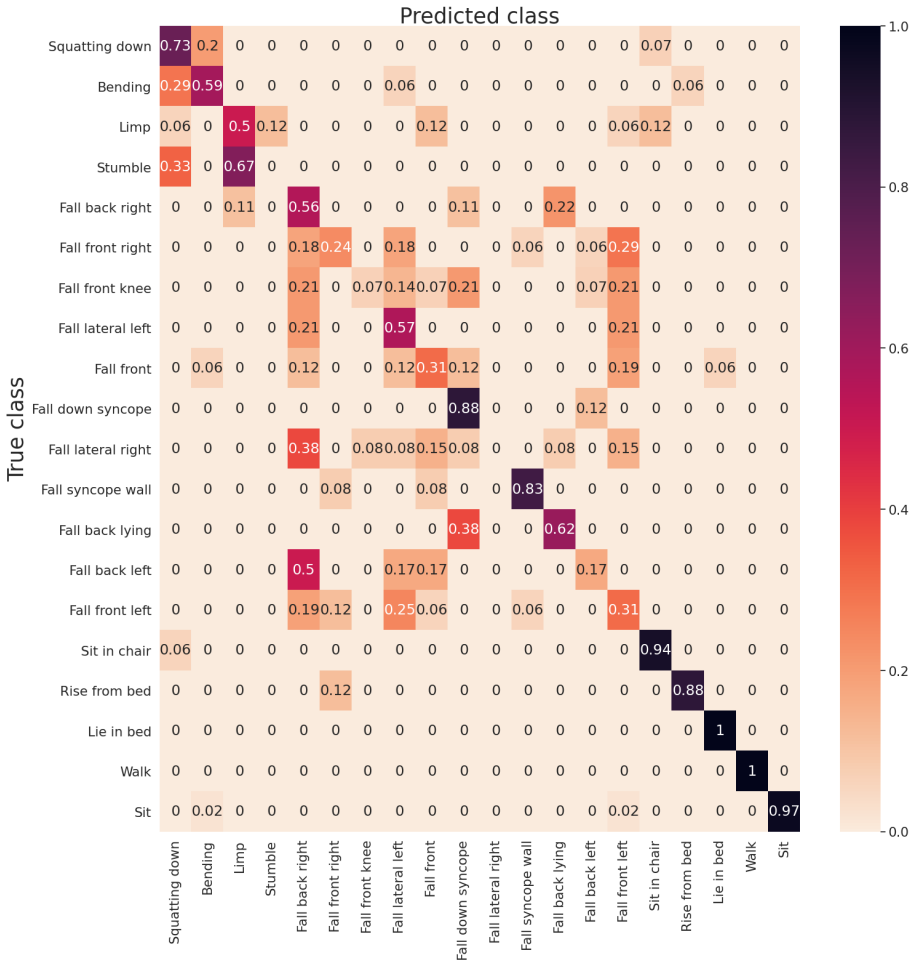

Figure 37: Normalized confusion matrix after executing the CoViAR I-frame CNN on all-classes_all_cameras_1.

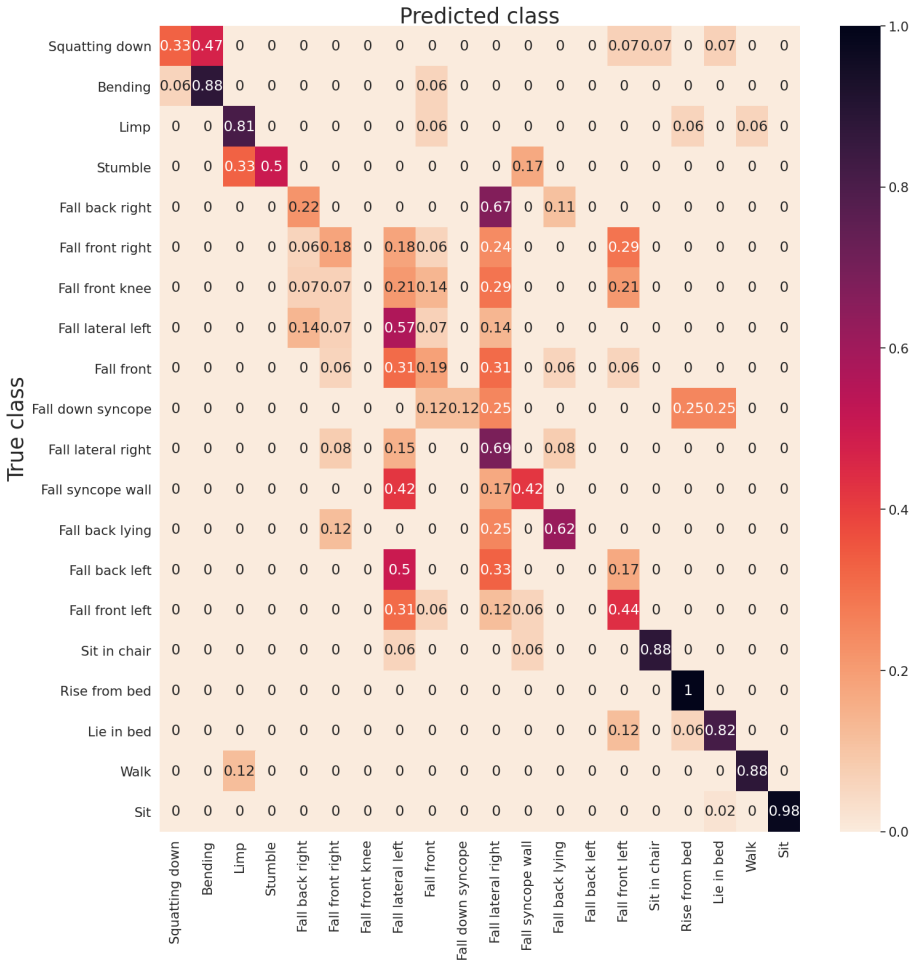

Figure 38: Normalized confusion matrix after executing the CoViAR motion CNN on all-classes_all_cameras_1.

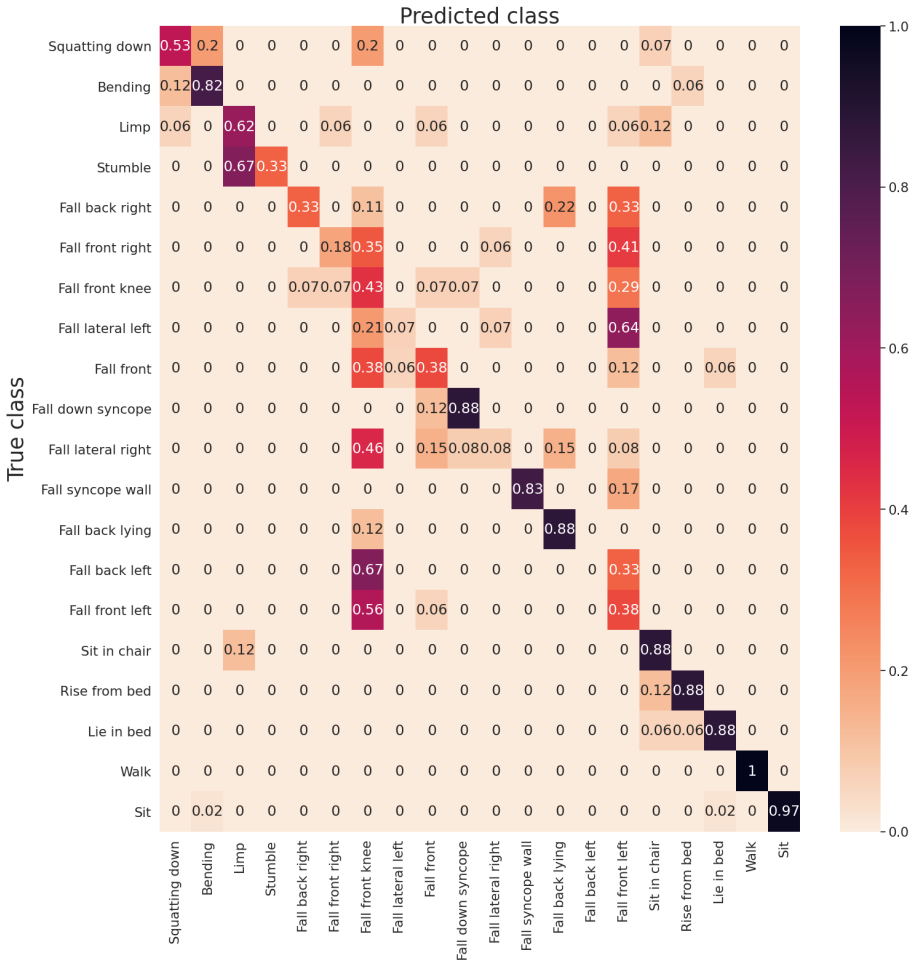

Figure 39: Normalized confusion matrix after executing the CoViAR residual CNN on all-classes_all_cameras_1.

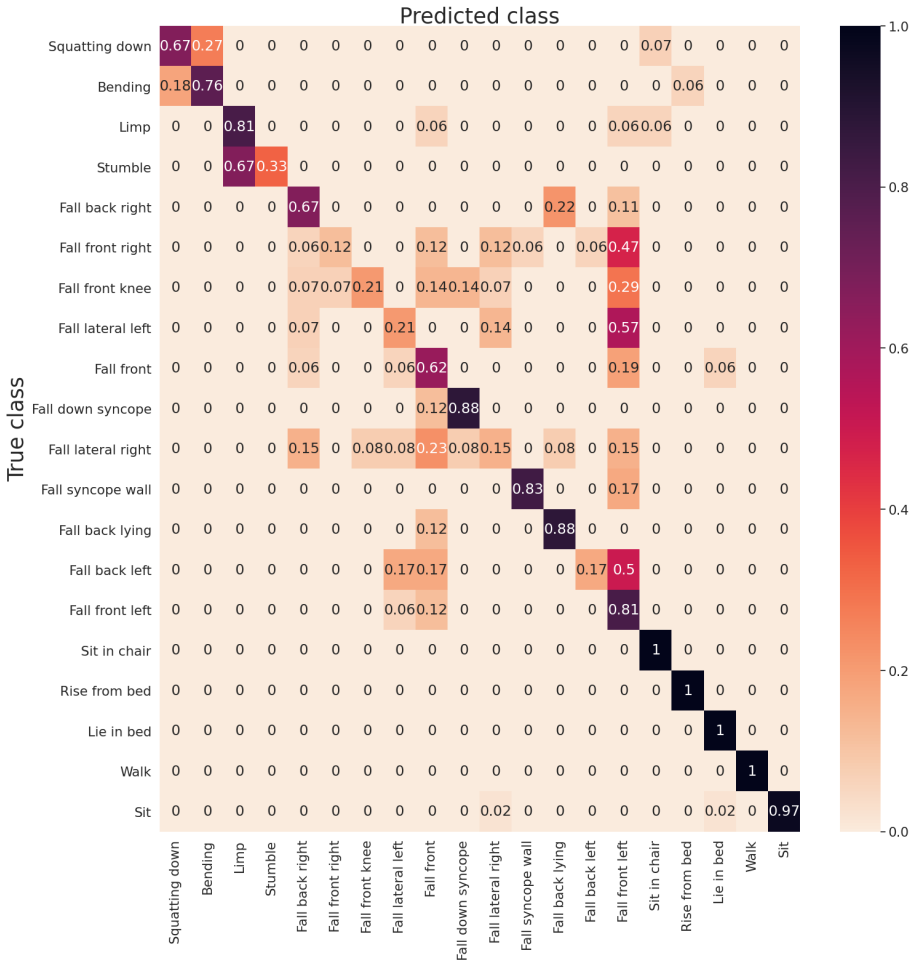

Figure 40: Normalized confusion matrix after executing CoViAR after late fusion on all-classes_all_cameras_1.

A.2.5 fall_non-fall_neck_2

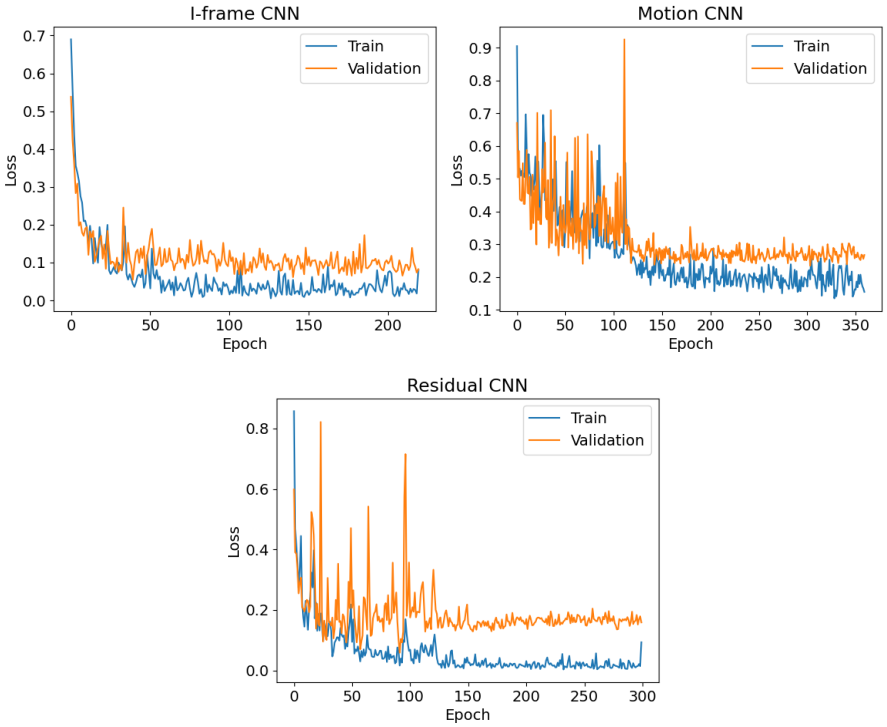

Figure 41: Training and validation losses for the CoViAR CNNs obtained after training on fall_non-fall_neck_2.

Classifier	PCA dim	Clusters	ROC-AUC	Accuracy	Recall (TPR)	FPR	Precision	Specificity
DT Trajectory	25	128	98.36	93.29	94.37	7.69	91.78	92.31
IDT HOF - MBH - Trajectory	150	128	97.36	92.62	87.32	**2.56**	**96.88**	**97.44**
CoViAR Late Fusion	x	x	**99.63**	**97.99**	**100.0**	3.85	95.95	96.15

Table 34: Performance metrics obtained testing the different algorithms on fall_non-fall_neck_2.

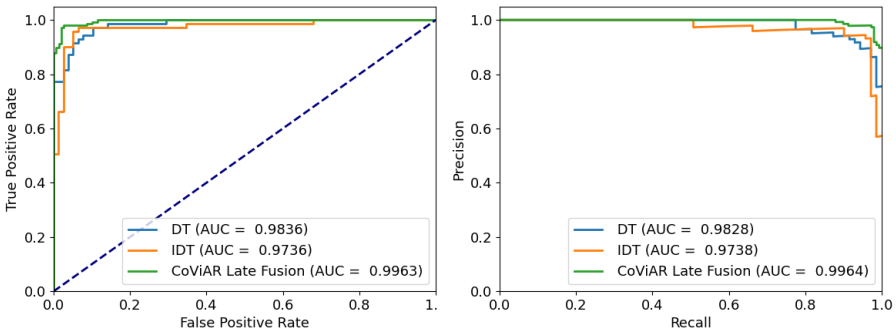

Figure 42: ROC curve (left) and PR curve (right) obtained after testing on fall_non-fall_neck_2.

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	72	6	78
	Fall	4	67	71
	Total	76	73	149

Table 35: Confusion matrix after executing DT on fall_non-fall_neck_2.

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	76	2	78
	Fall	9	62	71
	Total	85	64	149

Table 36: Confusion matrix after executing IDT on fall_non-fall_neck_2.

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	75	3	78
	Fall	0	71	71
	Total	75	74	149

Table 37: Confusion matrix after executing CoViAR after late fusion on fall_non-fall_neck_2.

A.2.6 fall_non-fall_waist_2

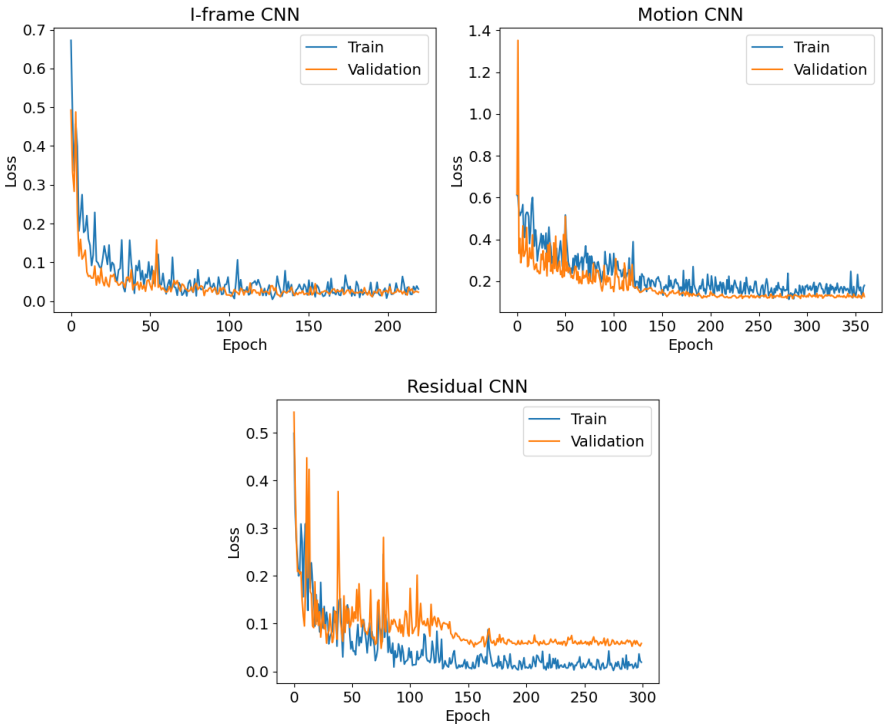

Figure 43: Training and validation losses for the CoViAR CNNs obtained after training on fall_non-fall_waist_2.

Classifier	PCA dim	Clusters	ROC-AUC	Accuracy	Recall (TPR)	FPR	Precision	Specificity
DT Trajectory	150	384	94.65	88.89	86.21	9.3	86.21	90.7
IDT HOF - MBH - Trajectory	200	256	92.2	89.58	79.31	3.49	93.88	96.51
CoViAR Late Fusion	x	x	**99.96**	**97.92**	**96.55**	**1.16**	**98.25**	**98.84**

Table 38: Performance metrics obtained testing the different algorithms on fall_non-fall_waist_2.

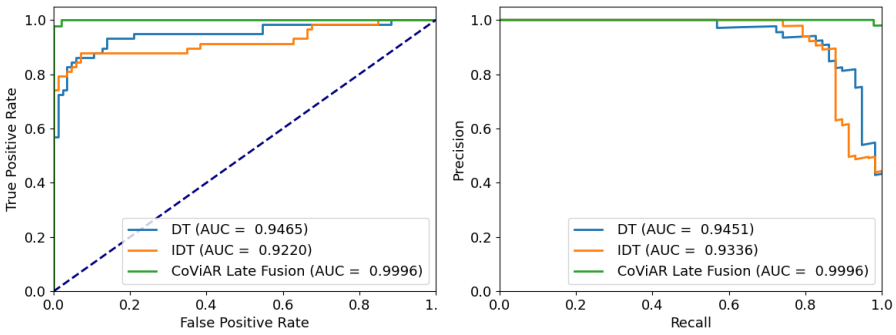

Figure 44: ROC curve (left) and PR curve (right) obtained after testing on fall_non-fall_waist_2.

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	78	8	86
	Fall	8	50	58
	Total	86	58	144

Table 39: Confusion matrix after executing DT on fall_non-fall_waist_2.

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	83	3	86
	Fall	12	46	58
	Total	95	49	144

Table 40: Confusion matrix after executing IDT on fall_non-fall_waist_2.

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	85	1	86
	Fall	2	56	58
	Total	87	57	144

Table 41: Confusion matrix after executing CoViAR after late fusion on fall_non-fall_waist_2.

A.2.7 *Sum of confusion matrices from the results of* fall_non-fall_neck_2 *and* fall_non-fall_waist_2

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	150	14	164
	Fall	12	117	129
	Total	162	131	293

Table 42: Sum of confusion matrices after executing DT on fall_non-fall_neck_2 and fall_non-fall_waist_2.

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	159	5	164
	Fall	21	108	129
	Total	180	113	293

Table 43: Sum of confusion matrices after executing IDT on fall_non-fall_neck_2 and fall_non-fall_waist_2.

		Predicted class		
		Non-fall	Fall	Total
True class	Non-fall	160	4	164
	Fall	2	127	129
	Total	162	131	293

Table 44: Sum of confusion matrices after executing CoViAR after late fusion on fall_non-fall_neck_2 and fall_non-fall_waist_2.

A.2.8 all-classes_all-cameras_2

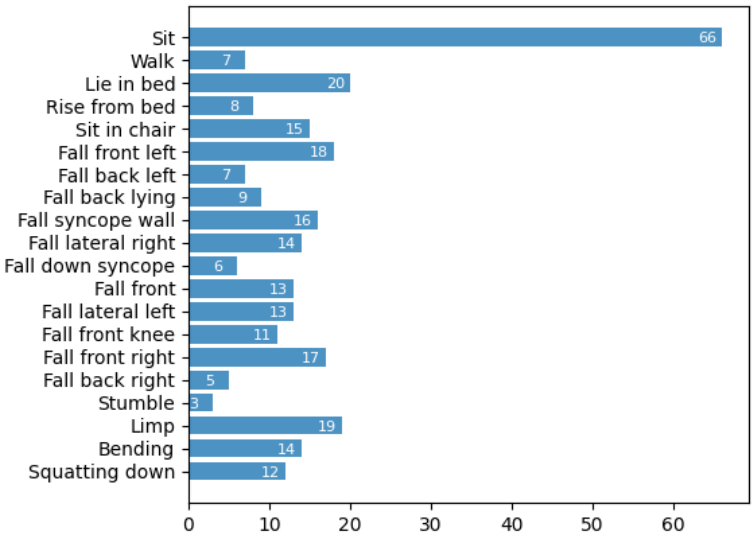

Figure 45: Distribution of the test set of classes of all-classes_all_cameras_2

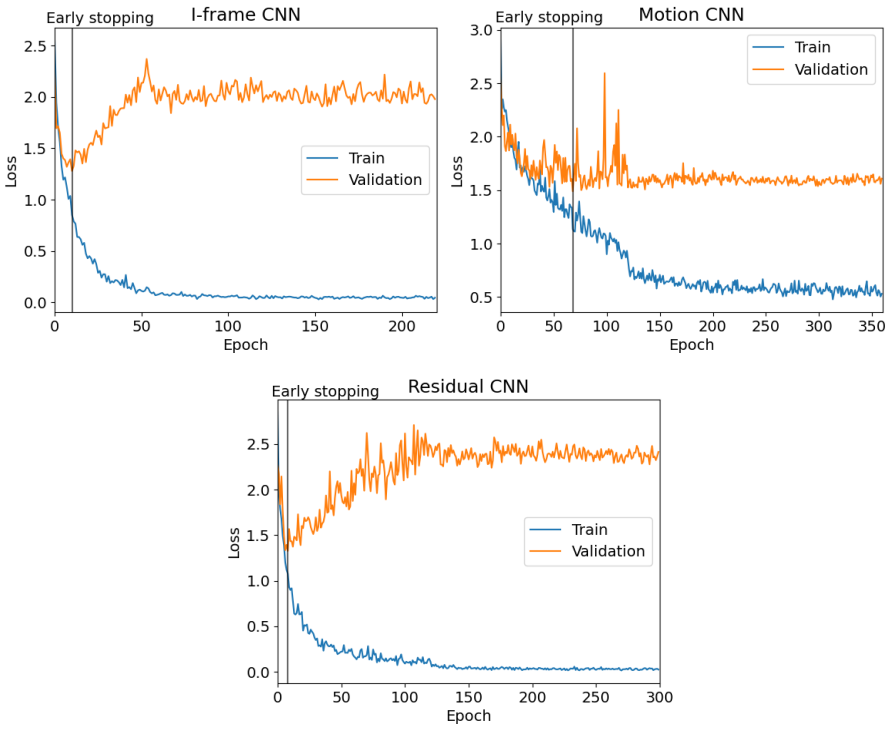

Figure 46: Training and validation losses for the CoViAR CNNs obtained after training on all-classes_all-cameras_2.

Classifier	AUC	Accuracy	Recall macro-averaged	Precision macro-averaged
CoViAR I-frame CNN	94.08	66.55	56.52	55.92
CoViAR motion CNN	91.98	67.92	60.7	59.1
CoViAR residual CNN	92.5	64.16	56.72	61.62
CoViAR Late Fusion	95.0	72.7	65.04	62.99

Table 45: Performance metrics obtained after testing CoViAR on all-classes_all-cameras_2.

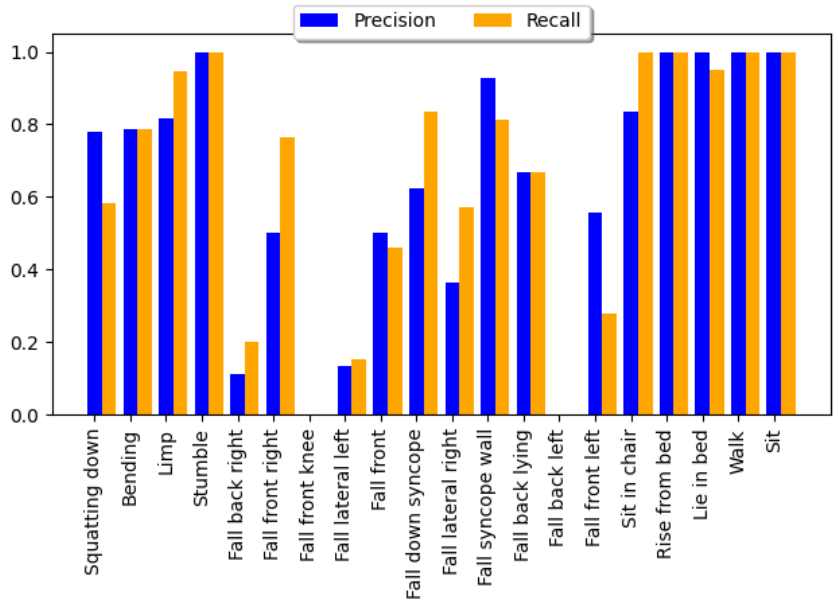

Figure 47: Precision and recall of the different classes after training CoViAR after late fusion on all-classes_all_cameras_2.

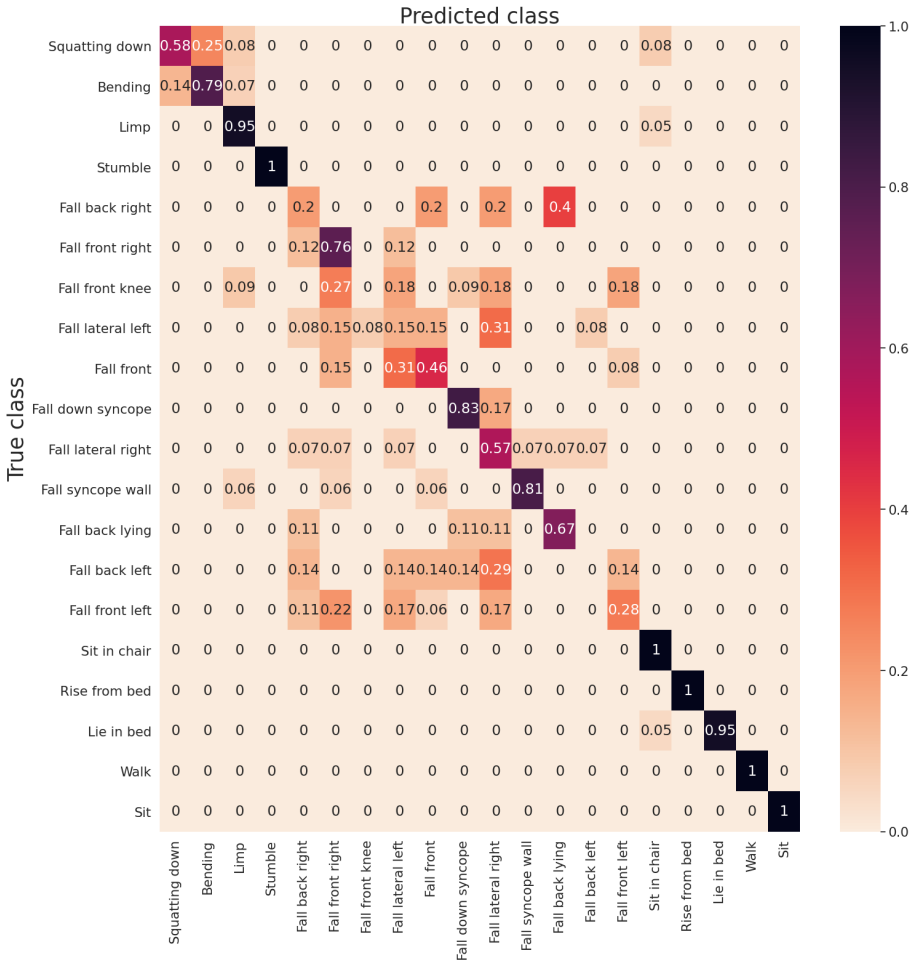

Figure 48: Normalized confusion matrix after executing CoViAR after late fusion on all-classes_all_cameras_2.