FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Automotive Interior Sensing - Anomaly Detection

Pedro Miguel Coutinho Augusto

MSC IN ELECTRICAL AND COMPUTER ENGINEERING

Supervisor @ FEUP: Jaime Cardoso

Supervisor @ Bosch: Joaquim Fonseca

July 27, 2020

Automotive Interior Sensing - Anomaly Detection

Pedro Miguel Coutinho Augusto

MSc in Electrical and Computer Engineering

July 27, 2020

Resumo

Com o surgimento dos veículos autónomos partilhados não haverá condutores nos veículos capazes de manter o bem-estar dos passageiros. Por esta razão, é imperativo que exista um sistema preparado para detetar comportamentos anómalos, como por exemplo, violência entre passageiros, e que responda de forma adequada. O tipo de comportamentos pode ser tão diverso que ter um *dataset* para treino que contenha todas as interações possíveis neste contexto é impraticável, implicando que algoritmos tradicionais de classificação não sejam ideais para esta aplicação. Por estas razões, os algoritmos de deteção de anomalias são a melhor opção para construir um bom modelo discriminativo.

Esta dissertação foca-se na utilização de técnicas de *deep learning*, mais precisamente arquiteturas baseadas em *Spatiotemporal auto-encoders* que são treinadas apenas com sequências de *frames* de comportamentos normais e testadas com sequências normais e anómalas dos *datasets* internos da Bosch. O modelo foi treinado inicialmente com apenas uma categoria das ações não violentas e as iterações finais foram treinadas com todas as categorias de ações não violentas. A rede neuronal contém camadas convolucionais dedicadas à compressão e descompressão dos dados espaciais; e algumas camadas dedicadas à compressão e descompressão temporal dos dados, implementadas com células LSTM (*Long Short-Term Memory*) convolucionais, que extraem informações relativas aos movimentos dos passageiros. A rede define como reconstruir corretamente as sequências de *frames* normais e, durante os testes, cada sequência é classificada como normal ou anómala de acordo com o seu erro de reconstrução. Através dos erros de reconstrução são calculados os *regularity scores* que indicam a regularidade que o modelo previu para cada *frame*. A *framework* resultante é uma adição viável aos algoritmos tradicionais de reconhecimento de ações visto que pode funcionar como um sistema que serve para detetar ações desconhecidas e contribuir para entender o significado de tais interações humanas.

Palavras-chave: Deteção de Anomalias, Visão Computacional, Aprendizagem de Máquina, Aprendizagem Profunda, Reconhecimento de Acções.

Abstract

With the appearance of SAVs (Shared Autonomous Vehicles) there will no longer be a driver responsible for maintaining the car interior and well-being of passengers. To counter this, it is imperative to have a system that is able to detect any abnormal behaviours, e.g., violence between passengers, and trigger the appropriate response. Furthermore, the type of activities can be so diverse, that having a dataset that incorporates most use cases is unattainable, making traditional classification algorithms not ideal for this kind of application. In this sense, anomaly detection algorithms are a good approach to build a good discriminative model.

This work focuses on the use of deep learning techniques, more precisely Spatiotemporal auto-encoder based frameworks, which are trained on normal human behavior video sequences and tested on use cases with normal and abnormal human interactions from Bosch's internal datasets. Initially, the model was trained on a single non-violent action category and final iterations considered all of the identified non-violent actions as normal data. The network architecture presents a group of convolutional layers which encode and decode spatial data; and a temporal encoder/decoder structure, implemented as a convolutional Long Short Term Memory network, responsible for learning motion information. The network defines how to properly reconstruct the normal frame sequences and during testing, each sequence is classified as normal or abnormal based on its reconstruction error. Based on these values, regularity scores are inferred showing the predicted regularity of each frame. The resulting framework is a viable addition to traditional action recognition algorithms since it can work as a tool for detecting unknown actions, strange/abnormal behaviors and aid in understanding the meaning of such human interactions.

Keywords: Anomaly detection, Computer Vision, Machine Learning, Deep Learning, Action Recognition.

Acknowledgements

I would like to thank my supervisor Prof. Jaime dos Santos Cardoso of the Faculty of Engineering of University of Porto and Eng. Joaquim Fonseca of the Bosch Car Multimedia Portugal Group. They provided continuous support, were always available to help and motivated me to make the right choices throughout the development of this dissertation. I would also like to thank Dr. Bruno Faria of the Bosch Car Multimedia Group for helping with the study of important theoretical concepts and the IVS Gen 2 team for their valuable advices.

I would also like to express my grattitude to Bosch Car Multimedia for providing a good working environment and the exceptional equipment in which I conducted all of my experiments.

I am also truly grateful for the education, work culture and friends I made at FEUP. Without them, these past five years would not have been the same.

Last but not least, I would like to express my deep appreciation for my loving family for being always present, providing me with everything I need, helping me in all my struggles and acknowledging my achievements.

Pedro Augusto

Contents

List of Figures

List of Tables

Abreviations and Acronyms

ACC	Accuracy
AAE	Adversarilly trained Autoencoder
AE	Autoencoder
AMDN	Appearance and Motion DeepNet
AMHOF	Adaptive Multi-scale Histogram Optical Flow
API	Application Programming Interface
AUC	Area Under the ROC Curve
AUC PR	Area Under the Precision-Recall Curve
BM	Bookmaker Informedness
CAE	Convolutional Auto Encoder
CBLOF	Cluster-Based Local Outlier Factor
COF	Connectivity-based Outlier Factor
ConvLSTM	Convolutional LSTM
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CUHK	Chinese University of Hong Kong
CtractAE	Contractive Autoencoder
C3D-AE	3D-Convolutional Autoencoder
EER	Equal Error Rate
ERR	Error Rate
FCN	Fully Connected Network
FDR	False Discovery Rate
FN	False Negative
FNR	False Negative Rate
FOR	False Omission Rate
FP	False Positive
FPR	False Positive Rate
F_1	F_1 score
GAN	Generative Adversarial Network
GM	Geometric Mean
GMM	Gaussian Mixture Model
GPU	Graphics Processing Unit

HBOS	Histogram-based Outlier Score
INFLO	Influenced Outlierness
Isomap	Isometric Feature Mapping
IR	Imbalance Ratio
IVS	In-Vehicle Sensing
KNN	K-Nearest Neighbours
LDA	Linear Discriminant Analysis
LDCOF	Local Density Cluster-based Outlier Factor
LLE	Locally Linear Embedding
LOCI	Local Correlation Integral
LOF	Local Outlier Factor
LSTM	Long/Short Term Memory
MCC	Matthews Correlation Coefficient
MDS	Multi-Dimensional Scaling
MHI	Motion History Image
ML	Machine Learning
MLE	Maximum Likelihood Estimate
MRI	Magnetic Resonance Imaging
MAE	Mean Absolute Error
MSE	Mean Square Error
NPV	Negative Prediction Value
OF-ConvAE-LSTM	Optical Flow Convolutional LSTM Autoencoder
PCA	Principal Component Analysis
PPV	Positive Prediction Value
PR	Precision Recall
PRC	Precision
ped	pedestrian
RAM	Random Access Memory
ReLU	Rectified Linear Unit
R-ConvVAE	Recurrent Convolutional Autoencoder
ROC	Receiver Operating Characteristic
ROI	Region of Interest
RNN	Recurrent Neural Network
SAV	Shared Autonomous Vehicle
SDAE	Stacked Denoising Autoencoder
SFA	Slow Feature Analysis
S_C-VAE	Skip Convolutional Variational Autoencoder
S_F-VAE	Stacked Fully Connected Variational Autoencoder
SGD	Stochastic Gradient Descent
SNS	Sensitivity
SPS	Specificity
SVM	Support Vector Machine
tanh	Hiperbolic tangent
TNA	True Negative Accuracy
TN	True Negative

TNR	True Negative Rate
TP	True Positive
TPR	True Positive Rate
UCSD	University of California San Diego
VAE	Variational Autoencoder
VAR	Vector Autoregressive model
YI	Youden's Index
δ	Imbalance Coefficient

Chapter 1

Introduction

"Today, driverless cars, as a new technology that allows a more accessible, dynamic and intelligent form of Shared Mobility, are expected to revolutionize urban transportation. One of the conceivable mobility services based on driverless cars is shared autonomous vehicles (SAVs). This service could merge cabs, carsharing, and ridesharing systems into a singular transportation mode.", (Vosooghi et al., 2019). However, without the driver's interior supervision, the users of this service may encounter unpleasant situations that could threaten their well-being and comfort. For that reason, an intelligent system is needed to detect abnormal behaviors and interactions. With the help of sensors placed in the interior of the vehicles, all the trips can be monitored and processed by an adequate anomaly detection algorithm that detects the abnormal behaviors and triggers pre-established counter-measures.

Anomaly detection refers to the problem of finding patterns in data that do not conform to the expected behavior (Chandola et al., 2009), more specifically of identifying abnormal data samples in datasets. These non-conforming patterns are often referred to as anomalies, outliers, discordant observations, exceptions, aberrations, surprises, peculiarities or contaminants in different application domains. There are several applications in which anomaly detection can be used such as the detection of cyber-intrusions, tumors in MRI images, frauds in credit card transactions, industrial damage and unusual data captured by sensor networks. The importance of detecting such anomalies lies in the fact that the anomalies in the data often translate to significant or even critical actionable information in the mentioned application domains.

The problem has been studied as early as the 19th century but the first definition was probably given by Grubbs in 1969 "An outlying observation, or outlier, is one that appears to deviate markedly from other members of the sample in which it occurs", (Goldstein and Uchida, 2016). Since then, several anomaly detection techniques have been developed and can be applied in most scientific domains.

The goal of the dissertation is to develop an algorithm capable of detecting anomalies inside vehicles, more specifically abnormal human interactions in the backseats of Shared Autonomous Vehicles. Bearing this in mind, the work developed here sets out to fulfill the following objectives:

- Outline essential machine and deep learning concepts.
- Describe the anomaly detection problem and the performance metrics more suited to benchmark its solutions.
- Study the main anomaly detection techniques.
- Review the current state of the art video anomaly detection algorithms and abnormal behavior detection strategies.
- Select the most suited video anomaly detection algorithm according to its performance on public datasets.
- Build a framework to prepare the available video data from Bosch's internal dataset for model training and testing.
- Evaluate the proposed solution's performance.

1.1 Structure of this document

Along with the Introduction, this document contains five more chapters:

- **Fundamental concepts of Anomaly Detection -** In chapter 2, the basic machine learning concepts are detailed along with the fundamental anomaly detection concepts and algorithms. In addition, in sections 2.6.3 and 2.6.6, an evaluation of classification performance regarding imbalanced datasets is conducted.
- **State of the Art video anomaly detection -** In chapter 3, the current state of the art video anomaly detection algorithms are covered and some important conclusions are drawn regarding the best solutions to the proposed problem.
- **Characterization of the problem -** In chapter 4, an anomaly detection framework is selected based on the study performed on chapter 3. To decide if it can be further adapted to detect abnormal behaviors inside vehicles, the approach is replicated and validated.
- **In-Vehicle Anomaly Detection -** In chapter 5, the detection of abnormal human interactions in the backseats of vehicles is covered. Details are provided on the anomaly detection dataset used for training, validation and testing, and the implementation of the framework and results are presented.
- **Conclusions and Future Work -** In chapter 6, the main conclusions are drawn regarding the work developed and the results obtained. Additionally, some thoughts on possible future work are presented.

Chapter 2

Fundamental concepts of Anomaly Detection

In this chapter, a broad overview on the Anomaly Detection problem is provided. It is structured into several sections:

- **Anomalies -** formal definition and examples, section 2.1.
- **Types of Anomalies -** categorization of the different types of anomalies according to singularity and context, section 2.2.
- **Input data -** distinction of the algorithm's input data according to time, space and representation, section 2.3.
- **Labeling -** outline of the three different anomaly detection setups according to the availability of labeled data, section 2.4.
- **Outputs -** distinction between the possible outputs of anomaly detection algorithms, section 2.5.
- **The Classification problem -** formal definitions of the most basic machine learning concepts regarding the classification problem, analysis of the most relevant classification metrics and the consequences of imbalanced data, section 2.6.
- **Algorithms -** extensive overview of the most used anomaly detection techniques and algorithms, section 2.7.
- **Choosing an algorithm -** analysis of the positive and negative aspects of the covered anomaly detection algorithms, section 2.7.7.

2.1 Anomalies

As previously mentioned, anomalies are patterns in data that do not conform to a well defined notion of normal behavior. They are different from the norm with respect to their features and rare in the dataset compared to normal instances. The two figures below illustrate anomalies in simple 2-dimensional data sets. In figure 2.1, extracted from Chandola et al. (2009), the data has two normal regions, N1 and N2, where most observations lie. Points o1, o2 and cluster O3 are sufficiently far away from the regions to be regarded as anomalies. In figure 2.2, extracted from Goldstein and Uchida (2016), two anomalies can be easily identified: x1 and x2 which are seen as **global anomalies**. When looking at the dataset globally, x3 can be seen as a normal record since it is not too far away from cluster c2. However, if we focus only on the cluster c2 and compare it with x3 ignoring all the other instances, it can be seen as an anomaly. Therefore, x3 is called a **local anomaly** as it is only abnormal when compared with its closest neighborhood. In cluster c3, the three observations could either be interpreted as three anomalies or as a (small) regular cluster. Algorithms should be able to differentiate its members from the normal instances and from the obvious anomalies, (Goldstein and Uchida, 2016).

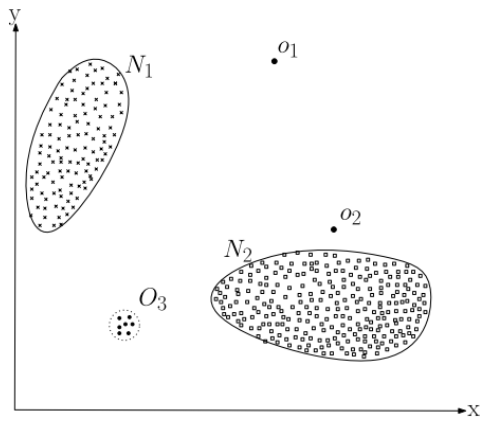


Figure 2.1: Anomalies by Chandola et al. (2009).

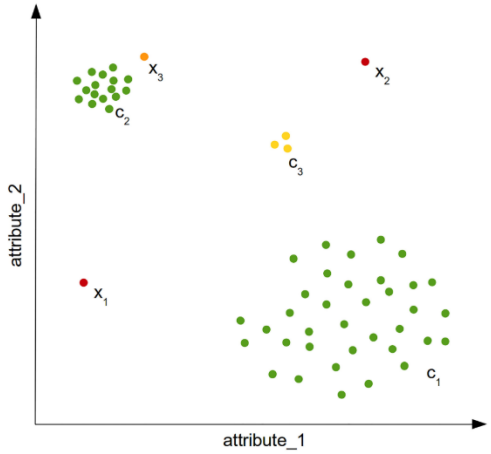


Figure 2.2: Local and global anomalies by Goldstein and Uchida (2016).

With these simple examples, a few challenges in anomaly detection can be deduced:

- **Normal region definition -** it may be difficult to define the region that encompasses every possible normal behavior.
- **Normal behavior evolution -** depending on the application, the normal behavior definition may change with the passing of time.
- **Noisy data -** the noise present in the data can be similar to the anomalies which makes the distinction between normal and anomaly even harder.

Along with these problems, several others can arise relative to the application.

2.2 Types of Anomalies

Anomalies can be classified as:

- **Point Anomalies -** If there are single/individual abnormal instances in a larger dataset these are termed as *point anomalies*. Nearly all the research in anomaly detection has been focused on detecting this type of anomaly.
- **Collective Anomalies -** If the anomaly presents itself as a collection of data instances, it is a *collective anomaly*. The individual data instances may not be abnormal by themselves but it is their occurrence together that defines the anomaly.
- **Contextual Anomalies -** If a data instance can be considered abnormal in a specific context (but not otherwise), then it is termed as a *contextual anomaly*, (Chandola et al., 2009). This type of anomalies has been commonly explored in the context of time, e.g. the temperature of 30^{o}C can be considered normal in the summer but if the measurement was made in the winter time, it would be considered an anomaly.

Fortunately, most point anomaly detection algorithms can be adapted to detect the other types of anomalies by including additional information as new features (derived or not) to the data instances. This transformation requires previous knowledge of the dataset and should result in a point anomaly dataset very different from the original. After preprocessing is complete, *normalization* should be applied to the data if proper background information is available. Typical normalization methods are min-max normalization, where every feature is normalized into a common interval (for instance [0, 1]), and standardizing, where each feature is transformed such that its mean is zero and its standard deviation is one, (Goldstein and Uchida, 2016).

2.3 Input data

The input data of the algorithm is defined as the collection of data instances. These may be objects, records, points, vectors, patterns, events, cases, samples, observations or entities. The set of attributes by which the data instance can be described are - variable, characteristic, feature, field, dimension (binary, categorical or continuous) and each instance may consist of only one atribute (**univariate**) or multiple attributes (**multivariate**). The nature of these attributes determine the applicability of anomaly detection techniques.

In terms of the relationship between the data instances, these can either have or don't have any relationship between each other. Most anomaly detection techniques deal with point data where there is no relationship. However, according to Chandola et al. (2009), if the data instances are related, these can be classified as follows:

- **Sequential data -** instances are linearly ordered. Spatio-temporal data such as videos or climate information belong to this category.
- **Spatial data -** there is a relationship between neighboring instances.
- **Graph data -** instances are represented as vertices in a graph and are connected to other vertices with edges.

2.4 Labeling

Labeling is the act of associating a data instance with a label that denotes it. Traditionally, it is often performed by a human expert and is time consuming and prohibitively expensive. As expected, getting a labeled set of abnormal behavior is more difficult than getting a labeled set of normal behavior. There are three different anomaly detection setups according to the availability of labeled abnormal data:

- **Supervised Anomaly Detection -** Techniques trained in supervised mode have access to fully labeled normal data instances and abnormal data instances. These instances can be split into two classes - the *normal* class and the *abnormal* class. This approach often leads to strongly imbalanced class distributions and, as stated before, the labeling of abnormal instances is usually challenging.
- **Semi-supervised Anomaly Detection -** This setup also makes use of labeled data for training but there are only labels available for the normal class data instances. A model of the normal class is learned and anomalies can be detected afterwards by deviating from that model (also known as "one-class" classification). Since labels for the abnormal class are not required, these algorithms are more applicable than fully supervised techniques.
- **Unsupervised Anomaly Detection -** This is the most flexible setup as it does not require labels. The anomaly scores, see section 2.5, are based solely on the intrinsic properties of the dataset, e.g. the frequency of normal and abnormal data instances.

2.5 Outputs

The anomalies can be reported in one of two ways:

- **Scores -** Depending on the degree to which an instance is considered an anomaly, a score is attributed. The output of these techniques is a ranked list of anomalies from which the normal instances can be ruled out according to a previously established cut-off threshold, (Chandola et al., 2009).
- **Labels -** The output of this technique is a binary result in which the data instances are either classified as normal or abnormal. A consequence of this kind of results is the inherent difficulty to select the most relevant anomalies.

2.6 The Classification Problem

The classification problem focuses on the ability to predict the class of an unknown data sample based on the experience gathered while training a model, usually in a **supervised** manner. The outputs of such classification models can either be discrete or continuous.

The operation is conducted in two phases:

- **Training Phase -** In this phase, a classifier is learnt using the available training data. The parameters of the model can be adjusted in order to minimize the training error (how well the classifier fits the training data).
- **Testing Phase -** After the classifier is built, the testing phase is conducted on the testing data (unknown to the classifier).

It should also be noted that an additional **validation set** is often used during the model's design to optimize 'user-defined' parameters. The models are trained with the training set, validated with the validation set and tested on the testing set or on the field, depending on the application. The performance of the classification model is measured by classification performance metrics and most of them are represented by scalar values. The most common are the **Accuracy**, **Sensitivity** and **Specificity**. In terms of graphical performance metrics, the Receiver Operating Characteristic (**ROC**) or the Precision Recall (**PR**) Curve are the most relevant ways of representing the performance of a classifier.

As previously stated, in the anomaly detection problem, the normal and abnormal class distributions used for training the model are very imbalanced. This imbalance may bias the performance metrics used for evaluating the classifier. This topic is analyzed in detail in section 2.6.3.

2.6.1 Types of classification problems

There are two types of classification problems:

- **Binary classification -** In this type of problem, the unknown data sample is classified as belonging to one of two classes: the P (or positive) class and the N (or negative) class. If the classifier's output is discrete, it represents the predicted class label of the unknown/test sample while a continuous output represents the estimation of the sample's class membership probability (Tharwat, 2018).
- **Multi-class classification -** In this setting, there are more than two classes. In the context of anomaly detection, it is assumed that the observations belong to multiple normal classes. A classifier is learnt to distinguish between each normal class against the rest of the classes. An anomaly is a test instance that is not classified as normal by any of the classifiers, (Chandola et al., 2009).

2.6.2 The Confusion Matrix

The confusion matrix is a simple table layout that allows the visualization of the performance of a supervised algorithm. In the case of a **binary classification problem**, the resulting confusion matrix is a 2x2 table, table 2.1.

Table 2.1: 2x2 confusion matrix.

Predicted Class	True/Actual Class: Positive (P)	True/Actual Class: Negative (N)
True(T)	True Positive (TP)	False Positive (FP)
False(F)	False Negative (FN)	True Negative (TN)
	P = TP + FN	N = FP + TN

As seen in table 2.1, adapted from Tharwat (2018), the output of the classification can be qualified as one of four options. If the sample is positive and is classified as positive, i.e., the classifier correctly predicts the positive sample, it is added as a true positive (TP); if it is classified as negative, it is counted as a false negative (FN) or Type II error. If the sample is negative and the classifier correctly predicts it as negative it is considered a true negative (TN); if it is classified as positive, it is counted as false positive (FP), false alarm or Type I error, (Tharwat, 2018).

The case of **multi-class classification**, is not so direct, table 2.2.

Table 2.2: Multi-class confusion matrix.

Predicted Class	True Class: A	True Class: B	True Class: C
A	TP_A	E_{BA}	E_{CA}
B	E_{AB}	TP_B	E_{CB}
C	E_{AC}	E_{BC}	TP_C

The number of true positive samples in class A, or TPA, is the number of samples that are correctly classified from class A, and EAB and EAC are the samples from class A that were incorrectly classified as class B or class C respectively. Thus, the false negative in class A (FNA) is the sum of EAB and EAC ($FNA = EAB + EAC$) indicating the sum of all class A samples that were incorrectly classified as class B or C. This reasoning can also be done for class B and C where the FN of the respective class, located in a column, can be calculated by adding the errors in that class/column. On the other hand, the false positive for any predicted class which is located in a row represents the sum of all errors in that row. For instance, the false positive in class A

(FPA) is calculated as $FPA = EBA + ECA$. With an m x m confusion matrix there are m correct classifications and $m^2 - m$ possible errors, (Tharwat, 2018).

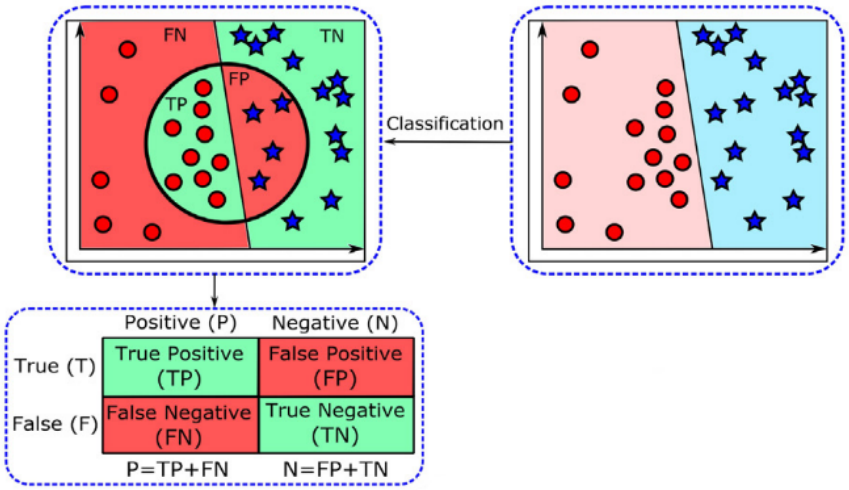


Figure 2.3: The binary classification problem (adapted from (Tharwat, 2018)).

2.6.3 Imbalanced data

Having imbalanced data means that the distribution of data instances across the different classes is not equal. In other words, imbalanced/skewed data reflects that its distribution's curve is distorted either to the left or to right. This mismatches the mean, median and mode that would be aligned at the centre of the distribution otherwise. The data may be right skewed or left skewed as seen in figure 2.4

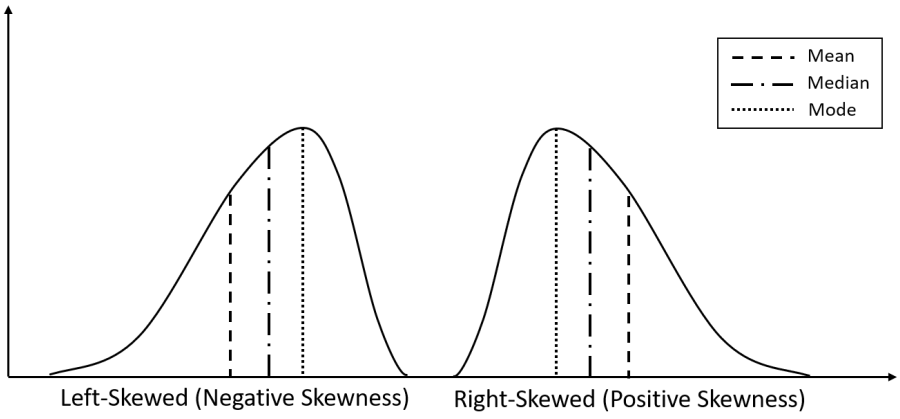


Figure 2.4: Data imbalance/skew.

One way of measuring the class imbalance is to calculate the imbalance ratio (IR), or *skew*, between the negative and positive samples (N/P) which represents the relationship between the right column to the left column, of the 2x2 confusion matrix. An assessment method is said to

be sensitive to imbalanced data if its value changes when the samples of one class in a dataset outnumber the samples of the other class(es), i.e., if the IR is different than 1. The majority of metrics that use values from both columns of the confusion matrix are sensitive to the imbalanced data, e.g. Accuracy and Precision, section 2.6.4, because if a class distribution changes, these metrics will also change despite the classifier performance remaining the same. Some metrics like the Geometric Mean (GM) and Youden's index (YI) use values from both columns and can be used with balanced and imbalanced data, section 2.6.4. This is due to the fact that these metrics are calculated in a way that the changes in the class distribution cancel each other.

2.6.4 Scalar classification assessment metrics

Scalar metrics can focus on the positive class, negative class or both. In terms of results, the interest may lie on the correct classification (Successes), incorrect classification (Errors) or the two situations.

- **Accuracy (ACC) and Error Rate (ERR) -** Accuracy is one of the most commonly used classification performance measures and is defined as the ratio of correctly classified samples to the total number of available samples, equation 2.1. It focuses on the positive and negative classes and the successes.

$$ACC = \frac{TP+TN}{TP+TN+FP+FN} \tag{2.1}$$

  Assuming that the negative class samples are increased by α times, the TN and FP values will be increased to αTN and αFP resulting in the following equation:

$$ACC' = \frac{TP+\alpha TN}{TP+\alpha TN+\alpha FP+FN} \tag{2.2}$$

  By looking at equations 2.1 and 2.2 it is clear that the accuracy metric is affected by changes in data distribution which means that it is **sensitive to imbalanced data**. Another problem with the accuracy metric is that two classifiers can yield the same accuracy but perform differently with respect to the types of correct and incorrect decisions they provide, (Tharwat, 2018).

  The complement of the accuracy metric (1 - ACC) is the Error Rate (ERR) or misclassification rate. Due to its relationship to the accuracy metric, the ERR is also **sensitive to imbalanced data**.

  NOTE: The imbalanced data test performed for this metric (or the complementary - an increase of the positive class samples of an α factor) will be the reference test performed in all the other metrics discussed in this section.

- **Sensitivity (SNS) and Specificity (SPC) -** The sensitivity, True Positive Rate (TPR), hit rate or recall, represents the proportion of the positive samples that were correctly classified, equation 2.3. Hence, it focuses on the positive class and on the successes.

$$SNS = TPR = \frac{TP}{TP+FN} = \frac{TP}{P} \tag{2.3}$$

The Specificity, True Negative Rate (TNR), or inverse recall, represents the proportion of the negative samples that were correctly classified, equation 2.4. It focuses on the negative class and on the successes.

$$SPC = TNR = \frac{TN}{FP+TN} = \frac{TN}{N} \tag{2.4}$$

Testing with imbalanced data,

$$TPR' = \frac{\alpha TP}{\alpha TP+\alpha FN} = TPR \qquad TNR' = \frac{\alpha TN}{\alpha TN+\alpha FP} = TNR \tag{2.5}$$

As mentioned in section 2.6.3, the Sensitivity and Specificity **are not sensitive to imbalanced data** because they depend on TP, FN and TN, FP respectively which belong to the same columns of the confusion matrix.

- **False Positive and False Negative Rates -** The False Positive Rate (FPR), also called false alarm rate, represents the proportion of the negative samples that were incorrectly classified, equation 2.6. Thus, it focuses on the negative class and on the errors.

$$FPR = 1 - TNR = \frac{FP}{FP+TN} = \frac{FP}{N} \tag{2.6}$$

The False Negative Rate (FNR), or miss rate, represents the proportion of the positive samples that were incorrectly classified, equation 2.7. It focuses on the positive class and on the errors.

$$FNR = 1 - TPR = \frac{FN}{FN+TP} = \frac{FN}{P} \tag{2.7}$$

It is clear on equations 2.6 and 2.7 that these metrics complement the Specificity and Sensitivity respectively. For this reason, they **are not sensitive to imbalanced data**.

- **Predictive values -** The Positive Prediction Value (PPV) or **precision (PRC)** represents the proportion of positive samples correctly classified from the total number of positive predicted samples, equation 2.8. Thus, PPV focuses on the positive and negative classes

and on the successes and errors. The False Discovery Rate (FDR) is the complementary measure of the PPV.

$$PPV = Precision = \frac{TP}{FP+TP} = 1 - FDR \tag{2.8}$$

The Negative prediction value (NPV), **inverse precision** or the True Negative Accuracy (TNA) measures the proportion of negative samples that were correctly classified from the total number of negative predicted samples, equation 2.9. Hence, focusing on the positive and negative classes and on the successes and errors. The False Omission Rate (FOR) is the complementary measure of the NPV.

$$NPV = Inverse\ Precision = \frac{TN}{FN+TN} = 1 - FOR \tag{2.9}$$

Testing with imbalanced data,

$$PPV' = \frac{\alpha TP}{FP+\alpha TP} \neq PPV \qquad NPV' = \frac{\alpha TN}{FN+\alpha TN} \neq NPV \tag{2.10}$$

Equation 2.10 clearly shows that the PPV and NPV are **sensitive to imbalanced data**.

- F_1**-score -** It is used as a weighted average of the precision and recall values, equation 2.11. It focuses on the positive and negative classes and on the successes and errors.

$$F_1 = 2 \times \frac{PPV \times TPR}{PPV + TPR} \tag{2.11}$$

As the metric depends on the precision metric, the F_1 score **is sensitive to imbalanced data**.

- **Younden's index - YI** or **Bookmaker Informedness (BM)** is one of the diagnostic tests mentioned in the previous metric. It is used to evaluate the discriminative power of the test. Combining the sensitivity (TPR) and the specificity (TNR), it is defined in equation 2.12. Thus, YI focuses on the positive and negative classes and on the successes.

$$YI = TPR + TNR - 1 \tag{2.12}$$

The metric ranges from 0 to 1. If for a particular test, the Youden's index equals 0, it means that the test is poor but if it equals 1, it means that the test is perfect. It is **not sensitive to imbalanced data**.

- **Matthews correlation coefficient (MCC) -** this metric represents the correlation between the observed and predicted classifications, (Tharwat, 2018). It is calculated directly from

the confusion matrix, equation 2.13. Focuses on the positive and negative classes and on the successes and errors.

$$\begin{aligned} MCC &= \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \\ &= \frac{\frac{TP}{N} - TPR \times PPV}{\sqrt{PPV \times TPR(1-TPR)(1-PPV)}} \end{aligned} \tag{2.13}$$

A result of 0 means that the prediction is not better than random guess, a result of 1 means that the prediction is perfect and a result of -1 reveals a total disagreement between the predicted and observed classifications, (Tharwat, 2018).

Testing with imbalanced data:

$$MCC' = \frac{\alpha TP \times TN - FP \times \alpha FN}{\sqrt{(\alpha TP+FP)(\alpha TP+\alpha FN)(TN+FP)(TN+\alpha FN)}} \neq MCC \tag{2.14}$$

It is clear that this metric is **sensitive to imbalanced data**.

- **Geometric Mean (GM) -** "The main goal of all classifiers is to improve the sensitivity, without sacrificing the specificity.", (Tharwat, 2018). However, this task is not straightforward because having both is often conflicting especially when the dataset is imbalanced. For this reason, the Geometric Mean, as the name implies, is the geometric mean of the sensitivity (TPR) and specificity (TNR) measures and is defined in equation 2.15. By looking at the equation, it can be seen that GM focuses on the positive and negative classes and on the successes.

$$GM = \sqrt{TPR \times TNR} = \sqrt{\frac{TP}{TP+FN} \times \frac{TN}{TN+FP}} \tag{2.15}$$

Both sensitivity (TPR) and specificity (TNR) are not sensitive to imbalanced data so the Geometric Mean is also **not sensitive to imbalanced data**.

2.6.5 Graphical classification assessment metrics

- **Receiver operating characteristics (ROC) -** the ROC curve is a two dimensional graph in which the y-axis represents the TPR (sensitivity/recall) and the x-axis represents the FPR (1 - specificity). It is used to perceive the balance between the benefits (true positives) and the costs (false positives). It has been used to evaluate several systems from the areas of diagnosis, medical and of course, machine learning. In figure 2.5, extracted from Tharwat (2018), is an example of a ROC curve.

Point A (FPR = 0,TPR = 0) - the classifier doesn't positively classify any sample but all the negative samples are correctly classified.

Point B (FPR = 0,TPR = 1) - the classifier correctly classifies all positive and negative samples, i.e., perfect classification.

Point C (FPR = 1,TPR = 1) - the classifier correctly classifies all positive samples but all the negative samples are incorrectly classified.

Point D (FPR = 1,TPR = 0) - the classifier incorrectly classifies all positive and negative samples which is the worst case scenario.

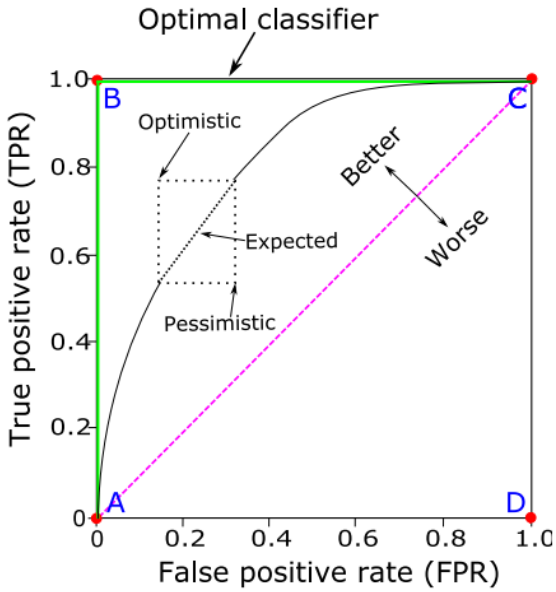


Figure 2.5: An example of the ROC curve, extracted from Tharwat (2018).

The green line in the graph reflects a classifier that correctly ranks the positive samples relative to the negative samples, meaning that the ROC curve should lie the closest to point B as possible. To move the ROC curve upwards, the TPR must be increased. To build the curve, the samples are sorted according to their classification scores and then the threshold is reduced from $+\infty$ to $-\infty$. As the threshold reduces, the samples are classified into the positive class if their scores are higher than the threshold or into the negative class if their scores are inferior to the threshold, (Tharwat, 2018). With each of the aforementioned iterations, the (FPR, TPR) points of the curve are determined. Important conclusions that can be drawn from the process are that the ROC curve is in fact a step function and as it depends mainly on changing the threshold value, comparing classifiers with different score ranges is meaningless, (Tharwat, 2018).

The ROC depends on the TPR and FPR metrics which means that the ROC **is not sensitive to imbalanced data**, but according to the study performed on Jeni et al. (2013), it may

mask poor performance. An important metric that can be extracted from the ROC curve is the equal error rate (EER) which corresponds to the point where the false positive rate (FPR) equals the false negative rate (FNR). It is also worth mentioning that a comparison between different classifiers using the ROC curve is only valid if the test is conducted with the same dataset or using multiple datasets with the same data size and class distribution, (Tharwat, 2018).

- **Area under the ROC curve (AUC) -** As mentioned above, the comparison between classifiers using the ROC curve can only be done under restrictive circumstances. The main reason is the fact that there is no scalar value representing the expected performance. The AUC measure solves the problem as it is used to calculate the area under the ROC curve and is a numerical value bounded between 0 and 1. No realistic classifier has an AUC lower than 0.5, as a good enough classifier must have its ROC curve as close as possible to the point B mentioned in section 2.6.5. Assuming a higher score for anomalies and a lower score for normal instances, the AUC score can be viewed as "the probability that an anomaly detection algorithm will assign a randomly chosen normal instance a lower score than a randomly chosen abnormal instance", (Goldstein and Uchida, 2016). Figure 2.6, extracted from Tharwat (2018), shows the AUC of two different classifiers, A and B. The two classifiers have a similar gray shaded area but the extra region in red shows that the classifier B has a larger overall area which means that B has a superior AUC value and the better performance between the two. As the AUC value depends on the ROC curve, **it is not sensitive to imbalanced data but may mask poor performance**.

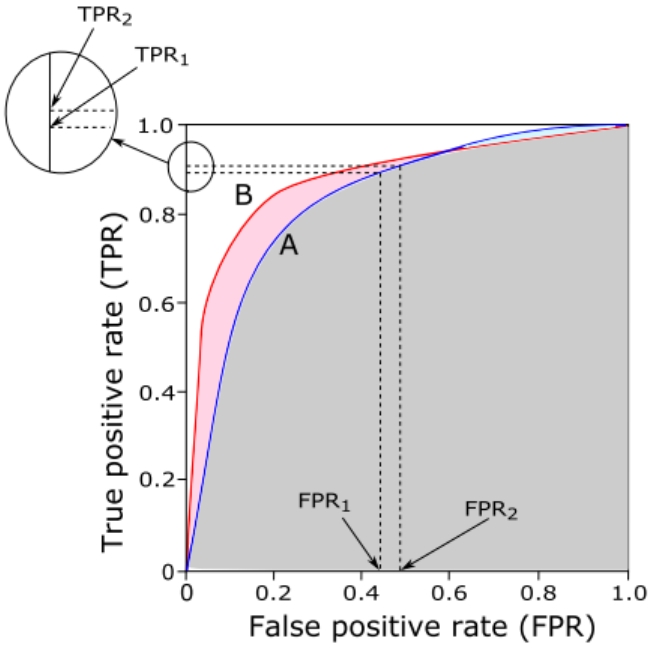


Figure 2.6: Comparison between the AUC metric of two classifiers, A and B, by Tharwat (2018).

- **Precision-Recall (PR) curve -** The precision (PPV) and recall (sensitivity or TPR) measures, displayed together, are also useful to determine the performance of a classifier. The PR curve has the recall in the x-axis and the precision in the y-axis. As in the ROC curve, it can be generated by varying the threshold value. However, the major difference in determining the curve compared to the ROC, is the fact that the first point is undefined due to the lack of positive predictions, i.e. TP = 0 and FP = 0. The problem can be easily solved by estimating the first point in the PR curve from the second point. If the second point is (0,0) the first point is also (0,0), if the number of true positives of the second point is not zero, the first point can be estimated by drawing a horizontal line from the second point to the y-axis, (Tharwat, 2018). As shown in figure 2.7, extracted from Tharwat (2018), the PR curve is a zigzag curve. A PR curve translates a better classifier performance if it lies above other classifier curves, and close to the green lines shown in the figure.

  Point A (TPR = 0, PPV = 0) - worst case scenario.

  Point B (TPR = 0, PPV = 1) - the classifier has 100% precision but 0% recall.

  Point C (TPR = 1, PPV = 1) - the classifier has 100% precision and 100% recall. This is the ideal point in the PR curve.

  Point D (TPR = 1, PPV = 0) - the classifier has 0% precision but 100% recall .

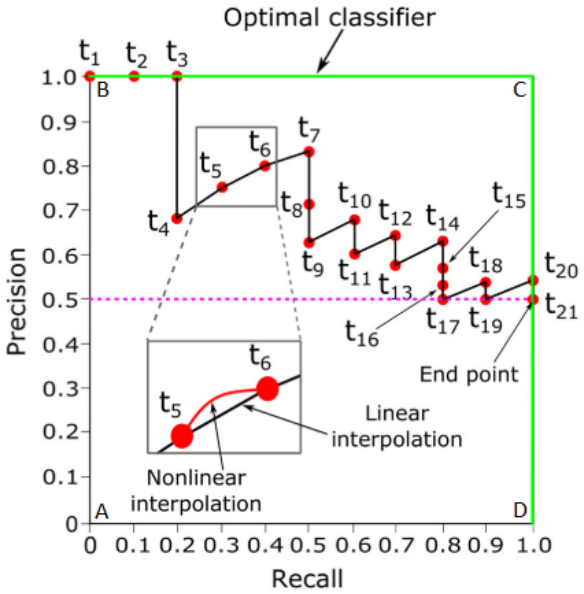


Figure 2.7: An example of the PR curve, extracted from Tharwat (2018).

The horizontal line in figure 2.7, represents a classifier with random performance level. Above the line are the classifiers with good performance and below the line lie the classifiers with poor performance. However, as the ratio of positive and negative classes changes the referred line and the classification performance metrics, it is clear that the PR curve (and the AUC values) are **sensitive to imbalanced data**.

2.6.6 Imbalanced data countermeasures

Several studies were made regarding the imbalanced data consequences on the performance metrics, scalar and graphical. The study done on Jeni et al. (2013) revealed interesting results on how the metrics effectively change with the varying *skew*, equation 2.16. A *skew* of 1 translates to a fully balanced dataset while a *skew* > 1 and a *skew* < 1 represent a majority of negative samples and a majority of positive samples, respectively. The authors concluded that except the area under the ROC curve (AUC), metrics such as the Area under the PR curve, the F_1 score and Accuracy are all affected by skewed distributions.

$$skew = \frac{\text{\# negative instances}}{\text{\# positive instances}} \tag{2.16}$$

Another study, (Luque et al., 2019), extensively studied the "Impact of class imbalance in classification performance metrics based on the binary confusion matrix". It defined bias as the root mean square of the imbalance coefficient, equation 2.17, and according to the obtained results, table 2.3, it is reinforced that in the first cluster of metrics, the best one-dimensional performance metrics are the sensitivity (SNS) and specificity (SPC). Geometric mean (GM) and bookmaker informedness (BM) are not one-dimensional and constitute good alternatives to be used with imbalanced datasets, but they have the drawback of focusing only on the classification successes and fail to directly consider the classification errors. The second best cluster of metrics is comprised of accuracy (ACC) and Matthews correlation coefficient (MCC). These have a global (not partial) perspective, since classification results on both positive and negative classes are considered. From among these 2 metrics, ACC focuses only on the classification successes and has the highest bias. The metrics in the third and fourth clusters, precision (PRC), negative predictive value (NPV), and F1 score (F1), are highly biased and should be avoided for use in imbalanced datasets.

$$IR = \frac{1+\delta}{1-\delta} \iff \delta = \frac{IR-1}{IR+1} \tag{2.17}$$

Based on these studies, it can be concluded that when dealing with imbalanced datasets, the choice of the best metric depends on several factors:

- If the focus is on the positive class and on the successes or errors, the best choice is **SNS** or **FNR** respectively.
- If the focus is on the negative class and on the successes or errors, the best choice is **SPC** or **FPR** respectively.
- For positive and negative classes, if the sole **focus on successes presents no limitation** for the specific application, **GM and BM are the best performance metrics**.
- For positive and negative classes, **if classification successes and errors must be considered, MCC arises as the best choice**.

Table 2.3: Behavior of scalar classification metrics with imbalanced datasets (P for positive, N for Negative, S for Successes and E for Errors).

Cluster	Metric	Bias	Focus on classes (P, N)	Focus on results (S, E)
I	SNS	Null	P	S
	SPC	Null	N	S
	GM	Null	P&N	S
	BM	Null	P&N	S
	FPR	Null	N	E
	FNR	Null	P	E
II	ACC	Medium	P&N	S
	MCC	Medium	P&N	S&E
III	PRC	High	P&N	S&E
	NPV	High	P&N	S&E
IV	F_1	High	P&N	S&E

However, if the use of an imbalanced dataset for training is not an option, a simple solution to mitigate the effects of imbalanced data is to sample the input data prior to training to re-balance the class distribution, (Jeni et al., 2013). It can be done in several different ways such as random over-sampling or random under-sampling. Random Oversampling involves supplementing the training data with multiple copies of some of the elements in the minority class whereas random under-sampling tries to balance the class distribution through random elimination of the majority class elements. Depending on the degree of skewness, such an approach could imply an extreme reduction of the dataset especially for anomaly detection applications. Even worse, it could discard important data instances for the performance metric. A smarter approach is to use cost-sensitive learning which applies different cost matrices that factor in the costs for misclassifying any data point. In the following section, the most common anomaly detection algorithms are explained.

2.7 Algorithms

There is a huge variety of algorithms and the practical requirements dictate which one should be selected. In some cases, the algorithm has to be very fast, e.g. in real-time applications, and in others the detection performance is the most important factor as there is a high cost of missing an anomaly. In this section, the most relevant anomaly detection algorithms and techniques are explained.

2.7.1 Support Vector Machines (SVMs)

In the context of anomaly detection, these have been applied mostly in one class classification settings. Support vectors are defined as the data instances that lie closest to the decision surface (or **hyperplane**), i.e., are the closest points of the two classes, and the most difficult to classify. SVMs

perform maximum-margin classification by optimizing a hinge-loss function, which translates to maximizing the margin around the separating hyperplanes and finding the optimal one for the linearly separable patterns, (Jeni et al., 2013). The hyperplanes are strictly linear and roughly equivalent to feed forward neural networks without an activation function. To apply to patterns that are not linearly separable, a kernel function can be used to build non-linear classifiers, known as the kernel trick.

2.7.2 Neural Networks

Neural networks are based on a collection of connected nodes (or neurons) which are modeled similarly to the neurons of a biological brain. Each connection transmits information to the other neurons in the network via signals which are usually real numbers, and the output of each neuron is computed by a non-linear function (**activation function**) of the sum of its inputs. Neurons have a weight that adjusts during the learning/training phase with the goal of minimizing a **loss function** with the help of an adequate **optimizer**. Neurons are aggregated into layers and the actual change in weight of any of its connections has a reverberating effect across all the other neurons in the network and their activations in the subsequent layers. The first layer is connected directly to the inputs and the last layer is responsible for producing the outputs of the network. The layers in between are called the hidden layers, see figure 2.8.

Forward propagation is the method by which successful neuron activations lead the overall prediction to the output of the neural network. The output itself is the ultimate source of error as it allows for the subtraction of the network's prediction of the actual/real expected outcome (output activation). **Back propagation** is the method by which the error is moved backwards through the network via its connections penalizing the offending neurons, i.e., responsible for the greater impact on the model's total error.

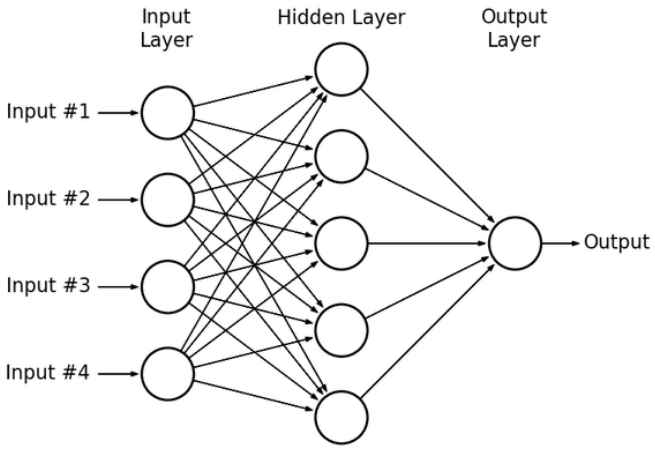


Figure 2.8: An example of a neural network.

In the sections below, a thorough analysis is made on the different aspects of neural networks.

Activation functions, loss functions, optimization algorithms, the overfitting and underfitting phenomenons and important neural network architectures, in the context of anomaly detection, are covered in detail.

2.7.2.1 Activation functions

As mentioned in section 2.7.2, the output Y of each neuron is determined by an activation function:

$$Y = Activation\ function\left(\sum(weight \times input + bias)\right) \tag{2.18}$$

The output Y of a neuron can be of any real value from $-\infty$ to $+\infty$. The neuron itself should either switch on or off according to the applied activation function. Below are the most commonly used activation functions:

- **Sigmoid function**

$$A = \frac{1}{1+e^{-x}} \tag{2.19}$$

  This activation function is nonlinear and its combination with other activation functions is also nonlinear. The closer the input x value is to $+\infty$, the closer the output is to 1 and the closer the x value is to $-\infty$, the closer the output is to 0. The output is then bounded to the 0 to 1 range.

- **Hiperbolic tangent (tanh) function**

$$tanh(x) = \frac{2}{1+e^{-2x}} - 1 \tag{2.20}$$

  This function is a scaled *sigmoid* function:

$$tanh(x) = 2 \times sigmoid(2x) - 1 \tag{2.21}$$

  The output is bounded between -1 and 1 and has a steeper derivative function when compared with the *sigmoid* function which implies a stronger gradient.

- **Rectified Linear Unit (ReLU) function**

$$A(x) = max(0, x) \tag{2.22}$$

  With this function, the output is equal to the input, if the input is positive, or 0 otherwise, (0, $+\infty$) bounded. The function is nonlinear as well as its combinations. This provides

sparser activation, i.e. less frequent, lighter and a more efficient neural network as its operations are not as computationally expensive (when compared to the dense activation functions previously mentioned). However, a problem associated with it is the "dying ReLU" problem which reflects the death of several neurons during descents for the activations in the negative x range. This problem makes a substantial part of the network passive but can be solved by making the horizontal part of the function non-horizontal with a slight incline - **Leaky ReLU** function.

- **Softmax function**

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \sigma : IR^K \to IR^K, for\ i = 1,...,K\ and\ \mathbf{z} = (z_1,...,z_k) \in IR^K \tag{2.23}$$

This function normalizes the outputs for each class between 0 and 1 by dividing them by their sum, giving the probability of the input value being in a specific class (outputs a probability distribution between the available classes). It is particularly useful for the output neurons in neural networks whose function is to classify the inputs into one of multiple classes.

2.7.2.2 Loss functions

Loss functions map events or values of one or more variables onto a real number intuitively representing some 'cost' associated with the event. In terms of Machine Learning and Deep Learning, the optimization problem of training the model seeks to minimize the loss function leading to parameter optimization. If the predictions $\hat{y}$ deviate too much from the actual observations y, the loss function outputs a very large number which is gradually reduced with the help of an optimization function. The functions detailed below are some of the most commonly used to optimize neural network architectures:

- **Mean Square Error (MSE) -** The MSE is the average of the squared difference between the observations y and the predictions $\hat{y}$:

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n} \tag{2.24}$$

The predictions which are far off from the actual values are penalized heavily in comparison to less deviated predictions. This function does not consider the direction of the error.

- **Mean Absolute Error (MAE) -** The MAE is the average of the sum of the absolute differences between the observations y and the predictions $\hat{y}$:

$$MAE = \frac{\sum_{i=1}^{n}|y_i - \hat{y}_i|}{n} \tag{2.25}$$

This function also does not consider the direction of the error.

2.7.2.3 Optimization

Optimization is present in most neural network based algorithms. The task refers to minimizing a loss function $f(x)$ by altering x, with the goal of finding x^* so that $x^* = argmin f(x)$. The technique on which most of the optimizers are based on is called **gradient descent**. For functions with a single input x, the technique revolves around using their derivative (e.g. $f'(x)$) to minimize it. By giving information on the slope of the function on each location (x), it allows to carefully move x in small steps and in the opposite sign of the derivative until a minima is found ($f'(x) = 0$). In the context of machine learning and deep learning, there are most likely multiple minima of the function that are not optimal making the optimization problem difficult, especially when the input of the function is multidimensional, (Goodfellow et al.). Therefore, the goal is to find a value of f that is very low, but not necessarily minimal in any formal sense, see figure 2.9.

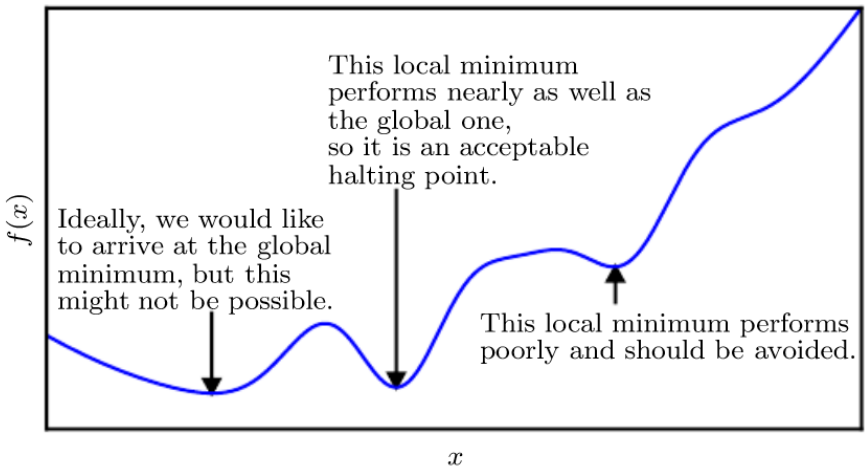


Figure 2.9: Optimizers may fail to find the global minimum but such solutions are accepted as long as they correspond to significantly low values of the loss function (figure extracted from Goodfellow et al.).

For functions with multiple inputs, the concept is generalized to the **gradient** - a vector holding all of the partial derivatives of a function, $\nabla_x f(x)$. With multiple dimensions, the critical points, in which the minima are included, are points where every element of the gradient equals zero. To minimize a multiple input function h, the objective is to find the direction in which h decreases the fastest. The gradient points uphill and the negative gradient points downhill, so h is decreased by moving it in the direction of the negative gradient, hence the name of **gradient descent**:

$$x' = x - \varepsilon \nabla_x f(x) \tag{2.26}$$

In equation 2.26, ε refers to the **learning rate**, a positive scalar determining the size of the step that can be set as a small enough constant, or by evaluating several different values and selecting the one that results in the smallest objective function value, (Goodfellow et al.).

Below are the most relevant optimization algorithms:

- **Stochastic Gradient Descent -** This optimization technique samples a minibatch of m examples from the training set, computes the aforementioned gradient estimate and applies the update on the parameters. Simple extensions of the original algorithm decrease the learning rate gradually over time to counter the SGD's gradient estimator introduction of noise (its random sampling of m training examples). One of the most attractive features of this optimization algorithm is the fixed computation time per update whatever the number of training examples, allowing convergence even for a large number of training examples. In terms of behavior across epochs, it makes rapid initial progress while evaluating the gradient for only very few examples but has a slow asymptotic convergence, (Goodfellow et al.).

- **Momentum -** As mentioned above, Stochastic Gradient Descent (SGD) can be slow, especially when the gradient becomes consistently small. The method of **Momentum** was designed to accelerate learning by accumulating an exponentially decaying moving average of past gradients and continuing to move in their direction. In mathematical terms, the variable $\mathbf{v}$ is added (v for velocity) and represents the direction and speed at which the parameters move through space. When applied to SGD, it reduces the variance in the stochastic gradient by changing the step depending on how large and how aligned the sequence of gradients is. A variant of the momentum algorithm is the **Nesterov Momentum**, inspired by Nesterov's accelerated gradient . The main difference when compared to the original **Momentum** algorithm is that the gradient is evaluated after the current velocity $\mathbf{v}$ is applied, adding a correction factor to the standard method of momentum. Despite mitigating several issues, it does it at the expense of introducing another hyperparameter, (Goodfellow et al.).

  The optimizers below do not have additional hyperparameters and use separate learning rates for each parameter that automatically adapt throughout the course of learning:

- **AdaGrad -** This optimizer individually adapts the learning rates of all model parameters by scaling them inversely proportional to the square root of the sum of all of their historical squared values. The parameters with the largest loss partial derivative have a correspondingly rapid decrease in their learning rate, while the parameters with small partial derivatives have relatively small decreases in their learning rates. The result is a greater progress in the more gently sloped directions of parameter space, (Goodfellow et al.). However, it was empirically found that the accumulation of squared gradients from the beginning of training may result in a premature and excessive decrease in the effective learning rate, (Goodfellow et al.).

- **RMSProp -** This optimizer modifies AdaGrad to perform better by changing the gradient accumulation into an exponentially weighted moving average. This way, history from the extreme past is discarded so that the exponentially decaying average can converge rapidly. Despite adding an additional hyperparameter ρ, when compared to SGD, it has been empirically proven that it is an effective and practical optimization algorithm, particularly for deep neural networks, (Goodfellow et al.).

- **Adam -** This algorithm can be seen as a variant on the combination of RMSProp and momentum with some distinctions. Here, momentum is incorporated directly as an estimate of the first order moment (with exponential weighting) of the gradient. It is generally regarded as being fairly robust to the choice of hyperparameters, though the learning rate sometimes needs to be changed from the default value, (Goodfellow et al.).

Unfortunately, there is currently no consensus on which optimization algorithm should be chosen and the choice seems to depend on the ease and familiarity with hyperparameter tuning of the algorithms.

2.7.2.4 Overfitting and Underfitting

Overfitting and Underfitting refer to the deficiencies that a model's performance might suffer from. To allow for a simpler explanation of the two situations, the linear regression example is used, see figure 2.10. Linear regression allows to map numeric inputs to numeric outputs, fitting a line into the data points. This line fitting process is the medium/average of both overfitting and underfitting. During the training of a linear regression model, the goal is to minimize the total distance (loss) between the line to fit and the actual data points.

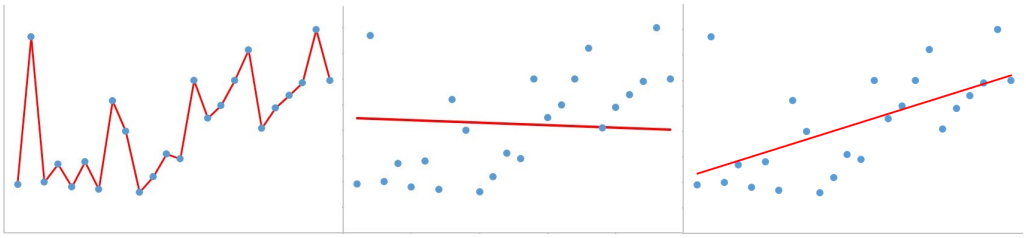


Figure 2.10: Overfitting, Underfitting, and Ideal linear regression models.

- **Overfitting -** Overly long training leads to minimal loss, but the line is fit to all points including noise and gathering secondary patterns that may not be needed for the generability of the model. An overfitted model captures all trends but not the dominant one because it contains more parameters than can be justified by the data. There is an overall small cost but unreliable predictions.
- **Underfitting -** One solution to prevent overfitting is stopping the training earlier. However, this could lead to the model not learning enough patterns from the training data, and possibly not capturing the dominant trend. When compared to a correctly specified model, it generally misses some of its parameters or terms.

Ultimately, having a network with great generalization ability is preferred over a network with a reduced training loss but unable to generalize well. Generalization is bounded by High Bias (Underfitting) and High Variance (Overfitting). In order to achieve a model with proper generalization, the tuning of several hyperparameters related to the network itself as well as the test of different optimizers and learning rates should be performed.

Neural Networks have been applied to anomaly detection in multi-class and one-class settings, (Chandola et al., 2009). In one-class classification, the network is trained and validated solely on normal data. Then, both normal and abnormal test instances (unknown to the model) are provided leading to the rejection of the anomalies. Multi-class settings for anomaly detection is much less common because as stated previously, it is unpractical to label all the possible abnormal scenarios.

For complicated problems, *neural network ensembles* can be used where many neural networks are fused together to combine different sources of knowledge in order to solve a problem, (Eunbyung Park, 2016). This fusion can either be performed in the first stages of the network (early fusion) or by dedicating two or more networks to learn separate data representations and make predictions by combining the information outputted by the different networks (late fusion).

2.7.2.5 Most relevant neural network architectures

In the context of anomaly detection, there are many neural networks that can be employed. LSTMs, AEs and GANs have been used extensively to detect anomalies. Details on their architectures are provided below:

- **Long-Short Term Memory (LSTMs) networks -** LSTM networks are a type of Recurrent Neural Network (RNN) which have multiple LSTM cells in sequence where the inputs are not independent of each other. Instead, the values of their output vectors are influenced not only by the corresponding input vector but also on the entire history of inputs which is very useful to learn temporal dependencies between sequential inputs or time series data. LSTMs overcome the **vanishing gradients problem** from traditional RNNs by incorporating a **forget gate** that prevents backpropagated errors from vanishing or exploding, (Chong and Tay, 2017). The LSTM cell structure contains an input gate, a forget gate and an output gate, figure 2.11. These gates regulate the flow of information - the input gate updates the cell state C_t (main information path), the forget gate decides what information should be kept or discarded and the output gate decides what the next hidden state h_t (information on previous inputs) should be. The formulation of a standard LSTM cell is summarized in equations 2.27 through 2.32

$$f_t = sigmoid(W_f \otimes [h_{t-1}, x_t] + b_f) \tag{2.27}$$

$$i_t = sigmoid(W_i \otimes [h_{t-1}, x_t] + b_i) \tag{2.28}$$

$$\hat{C}_t = tanh(W_C \otimes [h_{t-1}, x_t] + b_C) \tag{2.29}$$

$$C_t = f_t \otimes C_{t-1} + i_t \otimes \hat{C}_t \tag{2.30}$$

$$o_t = sigmoid(W_o \otimes [h_{t-1}, x_t] + b_o) \tag{2.31}$$

$$h_t = o_t \otimes tanh(C_t) \tag{2.32}$$

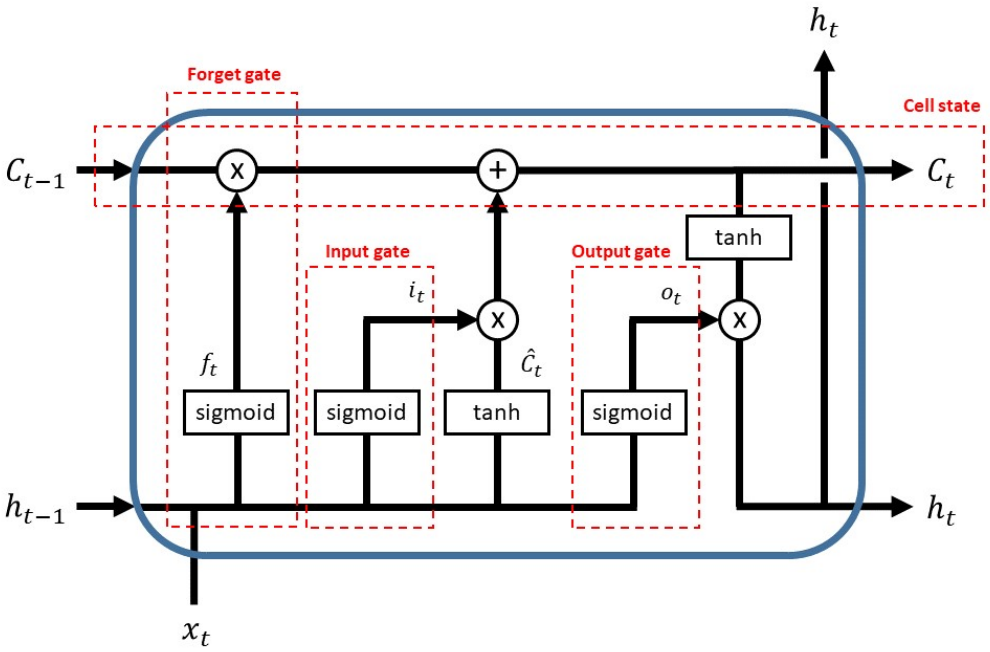


Figure 2.11: LSTM cell and its respective gates.

Equation 2.27 represents the forget layer, 2.28 and 2.29 are where new information is actually added, 2.30 combines the previous and new information and 2.31 and 2.32 output what has been learned so far to the LSTM unit at the next timestep. The variable x_t denotes the input vector at time t. W are the trainable weight matrices, b are the bias vectors, and the $\otimes$ symbol denotes the Hadamard product. An important variation of this fully connected architecture is the Convolutional LSTM or **ConvLSTM**. When compared with the structure explained above, the ConvLSTM has its matrix operations replaced with convolutions, more precisely, equations 2.27, 2.28, 2.29 and 2.31 have the Hadamard product operation $\otimes$ replaced by the convolution $*$. The ConvLSTM has displayed favorable results in video frame prediction as the data input, the video, is a sequence of frames each representing the evolution of a scene in time.

- **Autoencoders (AEs) -** Autoencoders are **unsupervised** neural networks that learn how to compress and encode the input data and to reconstruct it as close to the original input as possible. Besides the input and output layers with the same number of neurons, an autoencoder also has hidden layers h that describe a code used to represent the input. The network

consists of two parts: an encoder function $h = f(x)$ and a decoder that produces a reconstruction $r = g(h)$, figure 2.12 extracted from (Goldstein and Uchida, 2016). The autoencoders are unable to learn to copy perfectly but as the model is forced to prioritize which aspects of the input should be copied, it often learns useful properties of the data. This means that if the input data is correlated, autoencoders work very well because the encoding operation relies on the correlated features to compress the data. This neural network architecture is very useful for anomaly detection because an unknown (abnormal) data sample is poorly reconstructed leading to high reconstruction losses. These networks tend to be also implemented with other more simpler networks such as Feed Forward networks, LSTM Networks or Convolutional Neural Networks, Goodfellow et al..

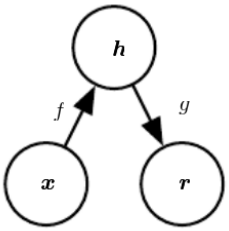


Figure 2.12: An Autoencoder maps inputs x to outputs **r** through a representation or code h.

- **Generative Adversarial Networks (GANs) -** The GAN is a generative modeling method, mostly used for generating realistic samples of natural images. In more technical terms, it is an approach to generative modeling where two models are trained simultaneously: a generator and a discriminator. The discriminator classifies an input as either the output of the generator ("fake" data), or actual samples from the underlying data distribution ("real" data). The generator produces outputs that are classified by the discriminator as "real", or as coming from the underlying data distribution.

2.7.3 Nearest Neighbor Based

The assumption this family of techniques is based on is that the normal instances occur in dense neighborhoods while anomalies occur far from their closest neighbors. To measure distance between data instances, several different computations can be made. If the data samples have continuous attributes, the Euclidian distance is a common choice but for categorical attributes using simple matching coefficients is a better choice. It all depends on the type of data. The measures are required to be positive-definite and symmetric, (Chandola et al., 2009). Unlike classification based techniques, the algorithms discussed here **can also be used in unsupervised settings**.

The techniques can be split into two categories:

- **Distance to the kth Nearest Neighbor -** Techniques from this category calculate the anomaly score of a given data instance as the distance to its k^{th} nearest neighbor in a data set. The choice of the $\boldsymbol{k}$ value is very important - if chosen too low, the density estimation of the records might not be reliable and if it is too large, the density estimation may be too coarse. As an alternative, the average of the distance to the k nearest neighbors can be used instead of the distance to the k^{th} nearest neighbor to calculate the anomaly score, (Goldstein and Uchida, 2016). There are also other techniques that calculate the anomaly score of a data instance as the sum of the distances of its k nearest neighbors, or as the number of nearest neighbors n that are not more than d distance apart from the given data instance, (Chandola et al., 2009).

- **Using Relative Density -** These techniques estimate the density of the neighborhood of each data instance. A data instance is considered to be normal if it lies in a dense neighborhood and is considered to be an anomaly if it lies in a low density neighborhood. The distance to the k^{th} nearest neighbor, as the radius of a hyper-sphere centered at a data instance containing k other instances, can be viewed as an estimate of the inverse of the density of the instance in the data set. The most well known algorithms from this category are the Local Outlier Factor (LOF), the connectivity-based outlier factor (COF), the Influenced Outlierness (INFLO) and the computational heavy, non-deterministic, Local Correlation Integral (LOCI).

2.7.4 Clustering Based

As the name implies, these techniques focus on grouping similar data instances into *clusters*. Like the *nearest neighbor techniques*, the algorithms primarily work on unsupervised settings. There are three main categories based on different assumptions:

- **Normal data instances belong to clusters and anomalies do not -** These are the most traditional clustering algorithms and have the disadvantage of not being optimized for anomaly detection because their main focus is to find the clusters, (Chandola et al., 2009).

- **Normal data instances lie closer to the center of the clusters than the anomalies -** These are two step techniques where a clustering algorithm is run first to find the clusters and the second step calculates the anomaly score of each data instance as its distance to the center of the closest cluster, (Chandola et al., 2009). The most relevant algorithm that belongs to this category is the *k-means clustering* which aims to partition the data instances into several clusters in which each instance belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into *Voronoi cells* which are partitions of the data plane into regions consisting of all the points of the plane closer to a data instance than to any other data instance. The only disadvantage of these algorithms is the fact that they do not detect anomalies that form clusters by themselves. The technique is commonly used not only due to its effectiveness but also to its linear computational complexity.

- **Normal data instances belong to large and dense clusters and anomalies belong to small or sparse clusters -** With these algorithms, the data instances that belong to clusters whose density is below a certain threshold are considered anomalies. The most relevant algortihms belonging to this category are the Cluster-Based Local Outlier Factor (CBLOF) and the Local Density Cluster-based Outlier Factor (LDCOF) for detecting local anomalies.

2.7.5 Statistical Techniques

The statistical techniques assume that the anomalies are observations which are suspected of being partially or wholly irrelevant due to not being generated by the stochastic model assumed initially, (Chandola et al., 2009). These techniques fit a statistical model to the given data and apply statistical inference tests. Instances that have a low probability of being generated by the learnt model are declared as anomalies. There are parametric as well as non-parametric techniques that have been applied to fit a statistical model. Parametric techniques assume the knowledge of an underlying distribution and estimate the parameters from the given data while non-parametric techniques do not generally assume knowledge of an underlying distribution, (Chandola et al., 2009).

- **Parametric techniques -** In terms of parametric techniques, there are Gaussian Model based, Regression Model based and Mixtures of Parametric Distributions. For Gaussian Model based techniques, the parameters are estimated using *Maximum Likelihood Estimates* (MLE) where the distance of a data instance to the estimated mean is the anomaly score for that instance. In Mixtures of Parametric Distributions, normal instances and anomalies are modeled as separate parametric distributions. Alternatively, normal instances may be modeled as a mixture of parametric distributions and anomalies can be seen as data instances that do not belong to the mixture. Gaussian Mixture Models (GMMs) are the most common among these techniques.

- **Non parametric techniques -** The non parametric techniques determine the model's structure from the given data instead of defining it before training and are used more often in practice than the parametric techniques. Histogram based techniques are the most simple and use histograms to keep a profile of the normal data. According to the study performed by Goldstein and Uchida (2016), the Histogram-based Outlier Score, or HBOS, was trained in a few minutes where nearest-neighbor and clustering algorithms took many hours to finish processing the training data. In theory, this means that it could be used in real-time anomaly detection. Kernel Functions also belong to this family of techniques.

2.7.6 Other Techniques

There are also Information Theoretic and Spectral anomaly detection techniques but these tend to be highly complex in terms of computation and do not offer any significant improvements when compared with the mentioned algorithms.

2.7.7 Choosing an algorithm

Different types of techniques face different challenges. Nearest neighbor and clustering based techniques suffer the most when the number of dimensions is high because the distance measures struggle to differentiate between normal and abnormal instances. Classification based techniques such as SVMs and Neural Networks can be better choices in this scenario but typically require labels for both normal and abnormal instances, which are not often available. Even with labels for both normal and abnormal instances, the imbalance in the distribution makes learning and testing a classifier quite challenging as seen in section 2.6.6 . Nearest neighbor and clustering techniques have the advantage of being unsupervised in nature and can often be more effective than the classification based techniques but require distance computation between a pair of data instances which can be sometimes difficult leading to the choice of other algorithms. Statistical techniques, though usually unsupervised, are effective only when the dimensionality of data is low and statistical assumptions hold. The computational complexity of an anomaly detection technique is a key aspect, especially when the technique is applied to a real domain. While SVMs and neural networks, clustering based algorithms, and statistical techniques can have expensive training times, testing is usually fast which is normally acceptable since models can be trained offline while testing can be performed in real time. In contrast, techniques such as nearest neighbor based do not have a training phase but have expensive testing which can be a limitation in a real setting, (Chandola et al., 2009).

The choice of an algorithm is not straightforward and depends on a number of factors as mentioned above. In the next section, the state of the art video anomaly detection algorithms are reviewed which allows for a better understanding of the algorithms that have been applied to detect anomalies in video data.

Chapter 3

State of the art video anomaly detection

In this chapter, video anomaly detection is analyzed. It builds upon the fundamental concepts provided in the previous chapter as it applies them in the context of video input data and closely relates them to the dissertation's thematic. It is divided into several different sections:

- **Introduction -** motivation for video anomaly detection, section 3.1.
- **Definition -** formal definition of video anomaly detection, section 3.2.
- **Feature extraction and dimensionality reduction -** explanation of feature extraction in videos and overview of the most common dimensionality reduction techniques, section 3.3.
- **Supervised vs Unsupervised video anomaly detection -** explanation of the different supervised/unsupervised settings in video anomaly detection along with some example algorithms, section 3.4.
- **Deep autoencoder architectures -** presentation of the most relevant public datasets used for benchmarking video anomaly detection algorithms and study of deep autoencoder based architectures, section 3.5.
- **Final remarks -** analysis of the most relevant algorithm regarding the dissertation's objectives, section 3.6.

3.1 Introduction

Nowadays, there are security cameras in many public places. These are normally used to detect security issues in which the detection of abnormal events is traditionally done manually by a security person. However, the effectiveness of the detection of such abnormal events highly depends on the skill and concentration of the individual and therefore varies from person to person. This variability has emphasized the need of an automatic and intelligent procedure for the detection of anomalies in videos for all sorts of applications.

With the emergence of SAVs, there is a need for algorithms capable of detecting anomalies inside vehicles. The anomalies can be of different sorts but in the scope of Shared Autonomous Vehicles, these translate to abnormal behaviors between human subjects. The only way to capture such abnormal behaviors is through video footage whose frames depict the spatial and temporal relationships between interactions.

3.2 Definition

Video anomaly detection is defined as the process of detecting abnormal events in video data and, for that reason, is a subclass of video event detection. As mentioned above, videos are usually processed in groups of frames because events always span more than one frame.

Most of the research done on this topic covers video anomaly detection in crowded scenes. The problem presented here is somewhat different due to the fact that only two seated people will be featured in the training videos, not moving in a large open area. Notwithstanding, the dilemma of trade-off between high discrimination ability and low computational complexity will still be present.

The majority of methods can be partitioned in two categories: trajectory- based methods and non-object centric methods, from which *object* is the name given to the things or human subjects included in the video.

3.3 Feature extraction and dimensionality reduction

In the context of image/video processing, *feature extraction* focuses on the efficient representation of the interesting/useful parts of a frame (e.g. points, edges or objects) as a compact feature vector. Features are the derived values from the initial set of data and are intended to be informative and non-redundant facilitating learning and leading to better human interpretation. *Feature engineering* is a related technique in which new features are generated from existing features by applying transformations or other operations.

Methods for feature extraction vary according to the surveillance target. There are two main groups of features: hand-crafted and learned.

The hand-crafted features either result from target tracking and analyses of individual moving objects in the scene (complex motion extraction), or directly from the image at the pixel level

(region-based), (Ribeiro et al., 2018). One technique from this category is **optical flow** which is used to estimate the motion of objects in videos. Unlike still images, videos have the additional temporal component, i.e., information is not only encoded spatially but also sequentially according to a specific order. In equation 3.1, the pixel characteristic is defined - it remains the same, simply moving from pixel to pixel across time. This change of pixel location is what is predicted by the flow field. For every two frames, a new image is calculated with a distinct color pattern. Elements in the frame moving to the upper left corner are presented in blue, moving to the lower left corner are presented in green, moving to the upper right corner are presented in purple and moving to the lower right corner are presented in orange.

$$I(x,y,t) = I(x+\Delta x, y+\Delta y, t+\Delta t) \tag{3.1}$$

In contrast to hand-crafted features, learned features are obtained directly from the raw input data by making use of deep learning methods that adaptatively learn them through multi-layer nonlinear transformations. Techniques belonging to these two groups can make use of **dimensionality reduction** techniques that focus on the reduction of the working feature set. These techniques were initially developed to fight the *Curse of dimensionality* which refers to the problems that arise when working with data in the higher dimensions. The larger the number of features, the more complex the models tend to become which also tends to increase the risk of *overfitting*. With *dimensionality reduction*, the amount of data is decreased which leads to a faster training phase and a reduction of the required storage space. An optimized feature extraction algorithm, preferably with dimensionality reduction, is therefore crucial for an effective model construction. Below, are the most common dimensionality reduction techniques:

3.3.1 Linear dimensionality reduction techniques

In this category, the performed transformations on the data are linear:

- **Principal Component Analysis (PCA) -** This technique rotates and projects data along the direction of increasing variance. This transformation results in the Principal Components which are the features with the maximum variance. In other words, PCA reduces high-dimensional data by computing dependencies between the attributes and representing it in a more tractable, lower-dimensional form, without losing much information, (Tripathi et al., 2015)

- **Linear Discriminant Analysis (LDA) -** This technique projects data in a way that the class separability is maximized. It first calculates the separability between different classes (i.e the distance between the mean of different classes) also known as between-class variance. The second step is to calculate the distance between the mean and sample of each class, the within class variance. The third and final step is to construct the lower dimensional space which maximizes the between class variance and minimizes the within class variance. These

can be viewed as putting examples from the same class close together by the projection and putting examples from different classes far apart by the projection.

It is important to note that **PCA is unsupervised** which means that it tries to find discriminatory information, without prior knowledge of any classes that are part of the data. In practical terms, PCA infers new features based on all the ones provided. **LDA is supervised** because it tries to find a projection in which labeled data is optimally separated, (Spulak et al., 2015).

3.3.2 Non-Linear dimensionality reduction techniques

In this category, the performed transformations on the data are non-linear. Some common methods are the Multi-dimensional scaling (MDS), Isometric Feature Mapping (Isomap) and Locally Linear Embedding (LLE). These techniques are not explained here in detail because they are not employed often in the video anomaly detection problem.

Along with the two categories explained above, variations on **Autoencoders** have been gaining popularity not only in the dimensionality reduction research field but also for semi-supervised video anomaly detection, (Yan et al., 2020).

3.4 Supervised vs Unsupervised video anomaly detection

Despite efforts to equate anomaly detection with a binary classifier, the scheme is often unrealistic in real-world applications. Video sequences that contain abnormal events are rare making the training of conventional classifiers in a fully supervised way impractical, (Yan et al., 2020). For this reason, the majority of video anomaly detection algorithms have **semi-supervised** or **unsupervised** natures, (Varghese et al., 2017). The semi-supervised methods have an explicit training phase, in which models are constructed from normal samples, and the unsupervised do not require offline training.

Learning temporal and visual characteristics of meaningful or salient moments is challenging as the definition of such moments is normally ill-defined, i.e., visually unbounded. On the contrary, learning temporal visual characteristics of ordinary moments is relatively easier as they often exhibit temporally regular dynamics, (Hasan et al., 2016). Hence, **semi-supervised** algorithms are commonly employed where abnormal events are usually defined as not having any similarity with the events used for training in terms of appearance, velocity, motion or trajectory, (Chaudhary et al., 2018). For instance, if a video that contains a person walking is used for training, then a video containing a stationary or running person represents abnormal events.

A study done on Tran et al. (2015) shows that convolution networks can be used to extract good features from videos, especially 3D convolutional deep networks which are good feature learning machines that model appearance and motion simultaneously. In another paper, (Boiman and Irani, 2007), the problem of detecting anomalies is viewed as the problem of creating the new unknown image using spatio-temporal patches extracted from previous known images (learned during the training phase). Patches of the training images are stored in a database during training and if

the testing image can be constructed using large contiguous chunks of data from the database, it is considered as normal and abnormal otherwise, (Varghese et al., 2017). Figure 3.1, extracted from Varghese et al. (2017), illustrates the method. In Tripathi et al. (2015), motion history image (MHI) and Hu moments are used to extract relevant features from the video footage. The dimensionality of the features is reduced by **Principle Component Analysis (PCA)** and normal and abnormal events are classified using **SVM**. In Saligrama and Chen (2012), motion descriptors are first extracted and quantified into small blocks. Then, Spatio-temporal filters at various scales are applied to find the smooth estimates at each spatio-temporal location for each feature descriptor. **Local K-Nearest Neighbors (KNN)** distance for each location is computed for the training and testing video. These neighborhood KNN distances are totaled to create a composite score for the test and training video. The scores are ranked to determine anomalies.

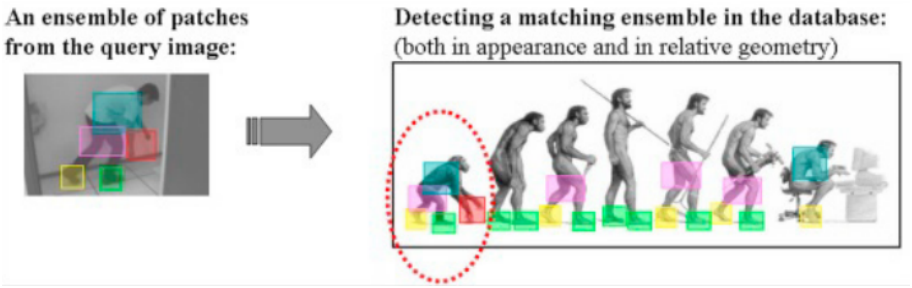


Figure 3.1: Detection of a matching ensemble of patches according to Boiman and Irani (2007).

Although less common, there are also some algorithms belonging to the **unsupervised category**. A solution explained in Roshtkhari and Levine (2013), groups frames and divides each group into spatio-temporal volumes. A collection of such volumes is called *ensemble* and is taken for processing in which the probability of each ensemble is computed. If an ensemble of the video has a large variation in probability when compared to the other computed ensembles, it is marked as an abnormal ensemble. Lin et al. (2016) makes use of flow vectors quantified within Regions Of Interest (ROI) using k-means clustering to obtain Adaptive Multi-scale Histogram Optical Flow (AMHOF) features. Each ROI is characterized by AMHOF and spatial location. An online weighted clustering of ROIs is performed to discern abnormal clusters from the normal ones. The algorithm is able to catch slow changes of normal motion patterns in an online adaptive manner but not fast enough for real-time operation. To detect anomalies in pedestrian motion, Chen and Shao (2015) compute their weighted speed based on optical flow to detect escape motion patterns using an empirically set threshold. A divergent center analysis is used to detect the center of crowd dispersion via paths intersection of escaping pedestrians. It is a method capable of detecting high speed motion but not suited to detect other types of anomalies (e.g., direction, appearance, and interaction).

Based on the algorithms explained above, it is clear that both the supervised and unsupervised scenarios have their advantages and disadvantages:

- **Semi-Supervised algorithms** only need to train a model for the normal video samples transforming the video anomaly detection problem into a one-class classification problem. This is a very interesting property as it removes the need to define all the abnormal behaviors which is time-consuming and practically unfeasible. However, if the normal class is redefined/extended, the model needs to be re-trained which can take a long time.

- **Unsupervised algorithms** typically group features into clusters without relying on any manually labeled training data, lacking any kind of train/test split. The algorithms in this category can be easily extended to handle new normal/abnormal motion patterns and are suitable for real-time applications. The major disadvantages of the algorithms from this category are their common use of hand-crafted features, implying a very strict use-case definition and the acquisition of specific knowledge required to design effective representations.

Some approaches rely on hand-crafted appearance and motion features for the automatic analysis of complex video scenes. Adopting these user defined representations is sub-optimal, as it is desirable to learn descriptors specific to the scene of interest, (Xu et al., 2017). With the emergence of **Deep Learning**, neural networks are now complex enough to learn video representations automatically and extract features from both spatial and temporal dimensions. The State-of-the-Art image recognition algorithms make use of deep convolutional neural networks (CNNs) that learn features that would be much difficult to grasp otherwise, (Krizhevsky et al.) and (Yan et al., 2020). Similar algorithms can be applied to video data but have to be adapted to handle its high dimensionality, noise and most importantly motion information. Models need to perceive the changes in appearance and motion resultant from the wide variety of events that can occur in between frames.

The majority of the State-of-the-Art Video Anomaly Detection algorithms today apply deep learning techniques, (Wang et al., 2019). Previous literature has shown that semi-supervised deep learning models are prevalent and making progress in performance day by day, (Duman and Erdem, 2019). From this point on, only deep learning semi-supervised methods will be discussed due to their demonstration of better results.

In all of these methods, and as stated in Kiran et al. (2018), the normal class distribution D is estimated using the training samples $x_i \in X_{train}$, by building a representation $f_\theta : X_{train} \rightarrow R$ that minimizes model prediction loss error over all the training samples, over all i.

$$\theta^* = arg_\theta min \sum_{x_i \in X_{train}} L_D(\theta; x_i) = arg_\theta min \sum_{x_i \in X_{train}} ||f_\theta(x_i) - x_i||^2 \tag{3.2}$$

The deviation of the test samples $x_j \in X_{test}$ under the representation of f^*_θ is evaluated as the anomaly score, $a(x_j) = ||f^*_\theta(x_j) - x_j||_2$. For said models, the abnormal points are samples that are poorly approximated by the estimated model f^*_θ and are detected by evaluating a threshold on the anomaly score $a_j > T_{thresh}$.

The deep learning frameworks for video anomaly detection can be categorized based on the type of model and criteria of detection, (Kiran et al., 2018):

- **Representation learning for reconstruction -** Methods from this category are Principal component analysis (PCA) and variations of the Autoencoder (AE) - Convolutional AE, Contractive AE, De-noising AE, and Stacked De-noising AEs (which learns a joint representation from correlations between appearance and motion). These algorithms perform linear (PCA) and non-linear transformations. Such transformations can be applied on appearance (raw image) or motion (flow), modeling the normal behavior in videos. Anomalies represent deviations that are poorly reconstructed.

- **Predictive modeling -** These architectures view video frames as temporal patterns or time series. The goal is to model a conditional distribution on the normal frames. In contrast to the reconstruction methods, where the goal is to learn a generative model that successfully reconstructs frames of a video, the goal here is to predict the current frame or its encoded representation using the past frames. Examples include convolutional Long-Short-Term-Memory models, 3D-Autoencoders and Predictors and Slow Feature Analysis (SFA).

- **Generative models -** Variational Autoencoders (VAE), Generative Adversarial Networks (GAN) and Adversarially trained AutoEncoders (AAE) are used to model the likelihood of normal video samples in an end-to-end deep learning framework. More precisely, the goal is to generate samples from the training distribution while minimizing the reconstruction error as well as the distance between generated and training distribution.

Kiran et al. (2018) studied how models from the three categories mentioned above approximated background estimation and temporal appearance. The reconstruction schemes were performed either on raw frame values or on the optical flow between consecutive frame pairs. According to the authors, reconstructing raw image values simply models the background image as minimizing the reconstruction error in fact evaluates the background. On the contrary, by learning feature representations on optical flow, the operation is indirectly done on two frames providing a greater sense of motion. The performance of the models was measured on the widely known UCSD and CUHK-Avenue public datasets, the latter having the smaller number of anomalies, more details on these datasets are provided in section 3.5. The reconstruction of a dense optical flow by PCA with 150 components was used as a baseline. The representative model for the predictive family was a vector autoregressive model (VAR), referred to as LinPred. Other tested models from the predictive family were Contractive autoencoders (CtractAE), simple 3D autoencoders (C3D-AE), ConvLSTM and ConvLSTM autoencoder. From the generative models, VAE was the baseline model used for comparison. For more details regarding the architecture of each specific network, see the original paper Kiran et al. (2018).

The results of each model on the datasets mentioned are listed on table 3.1. All the scores were calculated for either raw images or optical flow as the input channels, each normalized by their maximum values.

Based on the results, VAEs can be considered the best performing model of the ones tested. This observation has also been made in other studies (Yan et al., 2020), (Wang et al., 2019).

Table 3.1: Area under the ROC curve (AUC) of various deep learning video anomaly detection algorithms, (Kiran et al., 2018)

Methods Feature	PCA flow	LinPred (raw,flow)	C3D-AE (raw,flow)	ConvLSTM (raw,flow)	ConvLSTM-AE (raw,flow)	CtractAE (raw,flow)	VAE (raw,flow)
UCSDped1	0.75	(0.71, 0.71)	(0.70, 0.70)	(0.67, 0.71)	(0.43, 0.74)	(0.66, **0.75**)	(0.63, 0.72)
UCSDped2	0.80	(0.73, 0.78)	(0.64, 0.81)	(0.77, 0.80)	(0.25, 0.81)	(0.65, 0.79)	(0.72, **0.86**)
CUHK-avenue	0.82	(0.74, 0.84)	(**0.86** , 0.18)	(0.84, 0.18)	(0.50, 0.84)	(0.83, 0.84)	(0.78, 0.80)

Convolutional LSTM and LSTM-AE on raw image values follow behind as the predictive models performing as good as PCA. Such architectures have also been proved to perform well in Luo et al. (2017). Finally, the 3D convolutional autoencoder tested in the study performed as well as PCA on optical flow for modeling local motion patterns, (Kiran et al., 2018). These results may vary according to the surveillance scenario, but suggest that deep autoencoders provide some of the best overall results for the purpose of video anomaly detection. The next section focuses on 5 specific state-of-the-art video anomaly detection algorithms based on deep autoencoder architectures that show some of the best results on public datasets.

3.5 Deep autoencoder architectures

There are many studies that cover autoencoder based algorithms for video anomaly detection. To efficiently compare the performance of the algorithms, these are usually evaluated on public datasets.

3.5.1 Datasets

There are many datasets for benchmarking different machine learning algorithms. For video anomaly detection, the datasets available generally feature only normal activities on the training sets and normal and abnormal activities on the testing sets. The algorithms presented in section 3.5.2 were tested on the 3 following datasets:

- **UCSD ped 1 -** This dataset has 34 training videos and 36 testing videos. Each video has 200 frames with a 360 x 240 resolution and consists of groups of people walking towards or away from the camera on a public pathway. Anomalies include the appearance of bikers, skaters, carts, wheelchairs and people walking in an unknown direction.

- **UCSD ped 2 -** This dataset has 16 training videos and 12 testing videos with the same resolution as in UCSD ped 1 but the number of frames varies between videos and the pedestrians walk parallel to the camera plane. In the testing videos, the anomalies are of the same nature as in UCSD ped 1.

- **CUHK Avenue -** In this dataset, there are a total of 16 training, 21 testing videos and each of the sequences is short with about 1 to 2 minutes long. Resolution of each frame

is 640 x 360 pixels and the training videos consist of people walking between a staircase and a subway entrance, whereas the abnormal events feature people running, walking in the opposite direction, loitering, etc.

3.5.2 Algorithms

The most relevant video anomaly detection algorithms based on deep autoencoder architectures, trained and tested on the datasets mentioned above, are detailed below:

- **Detecting abnormal events in videos by learning deep representations of appearance and motion -** This study was the first attempt to address the abnormal event detection task using deep learning architectures, (Xu et al., 2017). It introduced the Appearance and Motion DeepNet (AMDN) - a double fusion framework that combines the benefits of traditional early fusion and late fusion strategies. Stacked denoising autoencoders (SDAE) were proposed to separately learn appearance and motion features as well as a joint representation (early fusion). Based on the learned features, multiple one-class SVM models are used to predict the anomaly scores of each input. The late fusion strategy combines the computed scores and detects abnormal events.

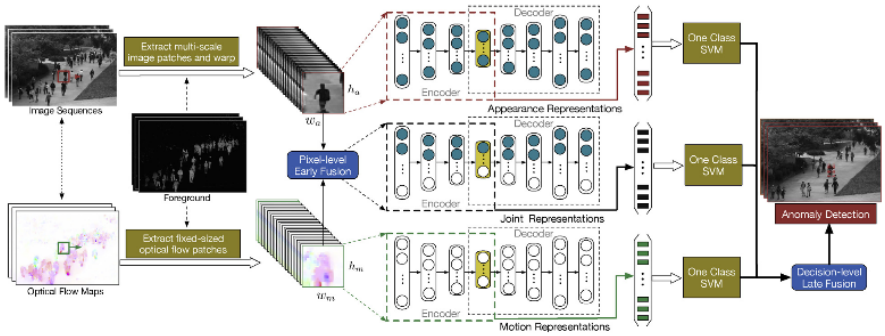


Figure 3.2: Architecture of the AMDN proposed in Xu et al. (2017)

Stochastic gradient descent was employed during the training of each SDAE and the training set was divided into mini-batches. In the testing phase, to improve the computational speed of the framework, the authors introduced a foreground detection approach.

During experiments, the early fusion was tested by only considering the learned joint appearance/motion representation and its one-class SVM, obtaining an AUC value of 0.849 for UCSD ped1 and 0.815 for UCSD ped2. For the late fusion, the two separate appearance, motion and their late fusion scheme are used but the joint representation pipeline is discarded, resulting in an AUC value of 0.891 for UCSD ped1 and 0.873 for UCSD ped2. It was observed that the late fusion strategy outperformed the early fusion but it was the combination of the two schemes that led to the best results with an AUC value of 0.921 for UCSD ped1 and 0.908 for UCSD ped2.

The authors acknowledged that similarly to other deep learning applications, the gain in terms of accuracy obtained with the deep architecture presented comes at the price of an increased computational cost.

- **Generative Neural Networks for Anomaly Detection in Crowded Scenes -** This paper, (Wang et al., 2019), proposed an end-to-end network called S^2-VAE, for anomaly detection in video data. As it has been already recognised, deep neural networks, **specially the generative models** (e.g, VAE) can yield better performance for abnormal event detection, (Wang et al., 2019). The proposed framework consists of two neural networks: a Stacked Fully Connected Variational AutoEncoder (S_F-VAE), with a low resolution input, and a Skip Convolutional VAE (S_C-VAE), with a high resolution input. The S_F-VAE is a shallow generative network with the purpose of obtaining a model similar to a **Gaussian mixture** to fit the distribution of the normal data. It was designed to filter out some palpable normal samples quickly, so that the S_C-VAE can learn a model from the remaining samples in a more efficient way. The S_C-VAE is also a deep generative network that takes advantage of the benefits of CNNs, VAEs and the **skip connection concept**. Skip connections feed the output of one layer to another, skipping a few layers in between, to help traverse information faster in deep neural networks. Furthermore, from information theory, the fusion of low-level and high-level information achieved by skip-connections can reduce the information loss caused by its transmission across the layers of the generative network, (Wang et al., 2019). The framework's architecture can be seen on figure 3.3.

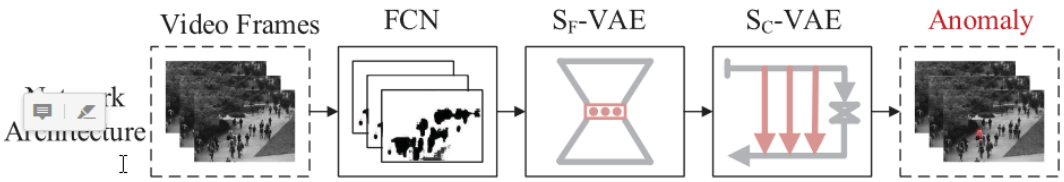


Figure 3.3: Architecture proposed in Wang et al. (2019) for video anomaly detection. Fully convolutional network (FCN), S_F-VAE and S_C -VAE.

The authors confirmed that both S_F-VAE and S_C-VAE are efficient and effective at detecting both local abnormal events and global abnormal events and it was assumed that the abnormal events happen in the foreground. The algorithm obtained a pixel-level AUC score of 0.9425 in UCSD ped1 and a frame-level score of 0.876 in CUHK Avenue.

- **Abnormal Event Detection in Videos using Spatiotemporal Autoencoder -** In this paper, Chong and Tay (2017) presented a framework that represents video data by a set of general features, inferred automatically through a deep neural network composed of a stacked convolutional temporal autoencoder. More specifically, the end-to-end model processes video frames in a semi-supervised manner capturing spatial structures first through convolutional layers and then grouping them together to compose a video representation that is fed to an

encoder-decoder structure with a convolutional LSTM, which together learn regular temporal patterns. The usage of a deep learning architecture simplified the feature engineering process while allowing for effective machine learning.

As previously stated, the method is semi-supervised meaning that it learns only on normal samples. As it is an autoencoder based model, it tries to minimize the reconstruction error between the input video volumes and the outputted reconstructed input volumes. During testing, when an abnormal event occurs, the most recent frames of the video are significantly different from the previous frames leading to higher reconstruction errors. Normal video volumes are expected to have low reconstruction errors, whereas video volumes consisting of abnormal scenes are expected to have high reconstruction errors. By thresholding on the error produced by each testing input volume, the system is able to detect when an abnormal event occurs.

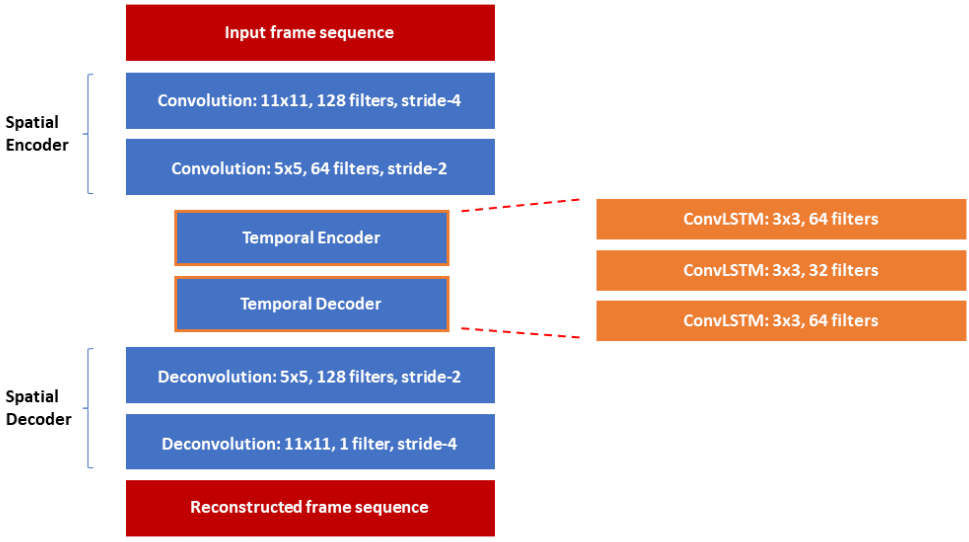


Figure 3.4: Architecture of the convolutional spatio temporal autoencoder proposed in Chong and Tay (2017)

Figure 3.4 illustrates the architecture proposed in Chong and Tay (2017). The temporal decoder-encoder module is the most crucial part of the network as it is responsible for learning the normal sequence of events, via the compressed features extracted by the previous layers.

The method is domain free as it does not require any additional human effort and can be easily applied to different scenes. The proposed method is applied on UCSD ped1, ped2 and CUHK Avenue displaying AUC values of 0.899, 0.874 and 0.803.

- **Anomaly Detection in Videos using Optical Flow and Convolutional Autoencoder -** In the study done by Duman and Erdem (2019), a similar framework to the one presented in (Chong and Tay, 2017) was proposed. It combines the same Convolutional Autoencoder and Convolutional Long Short-Term Memory network, but instead of using raw frames as input, it uses dense optical flow maps capturing extra velocity and direction information.

  The framework consists of three main stages. The first stage of the framework rescales 8 consecutive frames to 320x180, keeping the 16:9 aspect ratio (contrary to the 10 frames and 224x224 rescaling performed in Chong and Tay (2017)), extracts dense optical flow maps and converts each resulting frame to grayscale images. In the second stage, the convolutional autoencoder is used to obtain the spatial structure of each dense optical flow map volume. The last stage includes a convolutional long short-term memory network to learn the temporal patterns of encoded optical flow maps.

  The experiments were conducted on CUHK Avenue, UCSD Ped1, and UCSD Ped2 obtaining AUC values of 0.895, 0.924 and 0.929, respectively.

- **Abnormal Event Detection from Videos using a Two-Stream Recurrent Variational Auto-Encoder -** In Yan et al. (2020), a convolutional LSTM is embedded into a VAE to model video sequence for abnormal event detection. The model is named R-ConvVAE and has a two-stream structure composed of appearance and motion streams. For each stream, a recurrent variational autoencoder is used to model the probabilistic distribution of the normal data in a semi-supervised learning scheme. The idea of employing the two-stream architecture is that the temporal regularity of the appearance features and the motion features can complement each other in deciding which are the abnormal events. According to the authors, the VAE's primary advantage over the traditional AE is its encoding of the original images into a prior distribution instead of into deterministic features. However, traditional VAEs targeted at static image reconstruction cannot be directly applied to videos. It is necessary to model sequence dynamics successfully and the best performing networks to have done it are Recurrent neural networks. To this end, a convolutional LSTM network was employed to capture temporal dependencies while preserving spatial information. Regarding the framework's architecture, the spatial stream reconstructs the spatial frames using a recurrent VAE while the temporal stream reconstructs stacked optical flows with a similar VAE network, figure 3.5. With the information from the two fused streams, the model showed improved results when compared with a single stream. Notwithstanding, between early fusion, late fusion and double fusion, it was the latter configuration that showed the best results. The spatial stream and temporal stream were trained jointly and independently but after testing, it was verified that joint training performed slightly worse than independent training leading to the choice of training the two streams independently. As with Hasan et al. (2016), training was performed on all the available datasets and then tested individually for each one. The AUC results on the UCSD Ped1, Ped2 and CUHK Avenue were 0.75, 0.91 and 0.796.

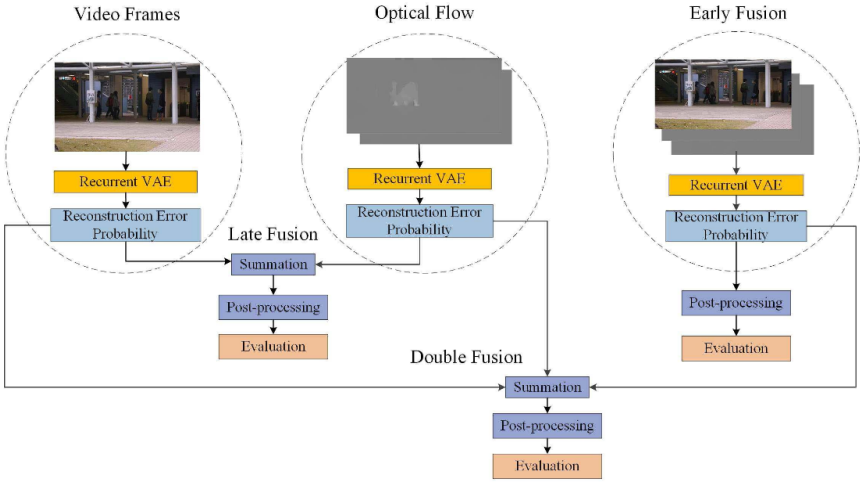


Figure 3.5: Two-stream architecture of the R-ConvVAE proposed in Yan et al. (2020)

3.6 Final remarks

Table 3.2: AUC of the studied deep learning video anomaly detection algorithms in section 3.5.

Algorithms	UCSD ped1	UCSD ped2	CUHK Avenue
AMDN (Xu et al., 2017)	0.921	0.908	–
SpatioTemporal AE (Chong and Tay, 2017)	0.899	0.874	0.803
S^2-VAE (Wang et al., 2019)	**0.9425 (pixel-level)**	–	0.876
OF-ConvAE-LSTM (Duman and Erdem, 2019)	0.924	**0.929**	**0.895**
R-ConvVAE (Yan et al., 2020)	0.75	0.91	0.796

This chapter showed that there is a great variety of video anomaly detection algorithms. It became clear that studying the video data is very important to devise a suitable feature extraction method in order to capture the most relevant information. The features themselves can be of different kinds: hand-crafted or learned through representation learning.

Deep-learning architectures, specifically predictive and generative models, have shown the best overall results for video anomaly detection applications in recent years. As demonstrated in Kiran et al. (2018), VAEs and Convolutional LSTMs performed best among all the frameworks studied throughout the paper. Similar conclusions can be drawn by looking at the results from the architectures reviewed in section 3.5 and displayed on table 3.2. Multiple stream approaches tend to present better overall results but there are also some single stream architectures equally effective. The S^2-VAE proposed in Wang et al. (2019) and the OF-ConvAE-LSTM proposed in Duman and Erdem (2019) showed the best AUC scores in the UCSD ped 1, UCSD ped 2 and CUHK Avenue datasets.

Chapter 4

Characterization of the problem

In this chapter, an anomaly detection framework is selected based on the study performed on chapter 3. To decide if it can be further adapted to detect abnormal behaviors inside vehicles, the approach is replicated and validated. The chapter is divided into two different sections:

- **Problem Formulation -** formulation of the anomaly detection problem for detecting abnormal behaviors inside vehicles and justification for the choice of the Spatiotemporal autoencoder framework, section 4.1.
- **Replication of the Spatiotemporal autoencoder framework -** explanation of all the functional blocks in the selected framework, details about its implementation and results, section 4.2.

4.1 Problem Formulation

In the majority of studies covering video anomaly detection, the algorithms were benchmarked on public datasets featuring surveillance pedestrian videos. The training videos feature people performing normal behaviors, primarily walking in public pathways. In the testing videos, there are normal samples and abnormal samples that feature pedestrians conducting unknown activities such as running, skating, riding bicycles or even the appearance of new objects like small carts (UCSD ped1 and UCSD ped2).

In this dissertation, the goal is to detect abnormal activities inside vehicles, not on pedestrian surveillance datasets. Notwithstanding, the techniques and algorithms involved are similar and have to be adapted to handle different inputs.

In section 3.6, the most relevant studies regarding video anomaly detection were covered in detail. All of these algorithms have different deep learning frameworks that are employed to detect anomalies in pedestrian surveillance data. The S^2-VAE proposed in Wang et al. (2019) and the OF-ConvAE-LSTM proposed in Duman and Erdem (2019) showed the best results for the UCSD ped1, UCSD ped2 and CUHK Avenue datasets. However, these works are quite recent and did not yet receive enough attention from the scientific community. For that reason, a simpler yet similarly effective alternative was chosen - the Spatiotemporal Autoencoder of Chong and Tay (2017). The framework has appeared in several other papers from which one is mentioned in the previous chapter - (Duman and Erdem, 2019). It is an end-to-end architecture that learns regular temporal patterns with the help of a spatial and temporal encoder/decoder structure.

4.2 Replication of the Spatiotemporal autoencoder framework

As stated in section 3.5, the approach revolves around the reconstruction error. An autoencoder is used to learn regularity in video sequences so that the reconstruction of regular video sequences results in a low reconstruction error and the reconstruction of irregular video sequences results in a high reconstruction error. The framework has a sequence of three main steps - Preprocessing, Feature Learning and Anomaly Detection.

4.2.1 Preprocessing

This stage converts raw data into the aligned and acceptable input for the model. The frames of the input video are resized to a 224x224 resolution and the pixels of each frame are scaled between 0 and 1. All the frames (training and testing) are subtracted from the global mean image and then are normalized to have zero mean and unit variance. Here, mean image refers to the resulting image of averaging the pixel values at each location of every frame in the training set.

The public datasets on which the framework was tested (UCSD ped1, UCSD ped2 and CUHK Avenue) are not particularly large (UCSD ped1 having 34 training videos and 36 testing videos

of only 200 frames each). The model has a little over a million parameters and needs a sizeable amount of training data. To tackle this problem, the authors decided to build video volumes with various skipping strides. In the context of video sequences, this temporal data augmentation technique involves concatenating frames with stride-1, stride-2 and stride-3 before feeding them to the model. For a sequence of frames 1 through 10, the stride-1 sequence is made up of frames 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, whereas the stride-2 sequence contains frame numbers 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, and the stride-3 sequence contains frame numbers 1, 4, 7, 10, 13, 16, 19, 22, 25, 28.

4.2.2 Feature learning

In this stage, the model is trained with the training video volumes with the goal of minimizing the reconstruction error between the input video volume and the output video volume reconstructed by the model.

The model itself has two parts:

- **A Spatial autoencoder** is used for learning spatial structures of each video frame. The encoder and the decoder both have two convolutional and deconvolutional layers responsible for extracting features from the input frames and, based on these features, reconstruct them. As previously stated, convolutional layers are preferred over fully connected layers for the purpose of feature extraction from images due to their preservation of the spatial relationship between pixels, (Chong and Tay, 2017) and (Simonyan and Zisserman, 2014).

- A **Temporal autoencoder** is employed for learning temporal patterns of the encoded spatial structures. The temporal encoder/decoder is a convolutional LSTM. The LSTM itself is known for sequence learning and time-series modeling having proved its performance in many applications, (Chong and Tay, 2017). As the spatial module has a convolutional architecture, to perfectly tie the spatial to the temporal autoencoder, the temporal module is a **convolutional** LSTM.

Viewing the architecture from a functional perspective, the model takes a sequence of T frames as input and outputs a reconstructed sequence of the same T frames (in the same order). The spatial encoder processes each frame at a time so that after the T frames have been processed, their encoded features can be concatenated and fed into the temporal encoder for motion encoding.

4.2.3 Anomaly detection

The reconstruction error of all the pixel values in frame t of a video sequence is taken as the Euclidean distance between the input frame $x(t)$ and the reconstructed frame $f_W(x(t))$:

$$e(t) = ||x(t) - f_W(x(t))||_2 \tag{4.1}$$

For a clearer representation of the anomaly score, normalized according to the events of the video, the regularity score is calculated:

$$s_a(t) = \frac{e(t) - e(t)_{min}}{e(t)_{max}} \tag{4.2}$$

$$s_r(t) = 1 - s_a(t) \tag{4.3}$$

The true positive and false positive rates (TPR and FPR) are obtained by setting different error thresholds in order to calculate the area under the receiver operating characteristic (ROC) curve (AUC). The equal error rate (EER) is also obtained, its calculation was explained in section 2.6.5. The regularity score allows for a better understanding of how regular a frame is according to its context in the testing video. Additionally, to reduce the noisy and unmeaningful minima in the regularity score, the Persistence1D algorithm from Kozlov (2017) is used to group local minima with a fixed temporal window of 50 frames as abnormal events usually last 2-3 seconds (in videos at 25 fps).

4.2.4 Implementation

Some of the framework's implementation is available in an online repository. The whole framework is written in python version 3.6.10 using CUDA version 10.0.130. Model declaration, training and testing is performed using *Tensorflow* version 1.14 with Keras API. Parts of the framework depicted in the paper are missing, namely the data augmentation section where the stride-1, stride-2 and stride-3 sequences were concatenated before training. This functionality was implemented and added to the base framework. Another functionality missing was the application of the Persistence1D algorithm to reduce the noise and unmeaningful minima in the regularity score. As this feature only smoothes the regularity score, and since its original paper was not available at the time of writing, it was replaced by an alternative - the Savitzky Golay filter. This type of low-pass filter was proposed by Savitzky and Golay and smoothes one dimensional data based on local least-squares polynomial approximation. It was shown that fitting a polynomial to a set of input samples and then evaluating the resulting polynomial at a single point within the approximation interval is equivalent to discrete convolution with a fixed impulse response. These filters maintain the shape and height of waveform peaks, preserve the trends and filter out points that have large excursions from the mean. For more information regarding the theory behind the Savitzky-Golay filter, refer to the original paper (Savitzky and Golay, 1964). The filter was implemented using *scipy's* function `savgol_filter()` with a window length of 51 and a polynomial order of 11. This parameter combination was used instead of others because it was the one that closely depicted the original trends while removing noisy data points.

For this replication, only the UCSD ped1 dataset was considered. The dataset's videos were split into frames and stored in directories. There are two large directories, one for the training

frames and another for the testing frames. In the training frames directory there are 34 directories storing the frames of each one of the 34 training videos and in the testing frames directory there are 36 directories storing the frames of each one of the 36 testing videos. The framework launches processes to run the training or testing stages. Besides training and testing, there is also a function to retrieve the model, another to compile the model, another to get the UCSD ped1 ground truth file and another to calculate the overall AUC and EER values. Alongside the classifier utilities file, there is a dataset utilities file that stores the implementation of all the functions related to the preprocessing procedure - global mean frame calculation, global mean frame subtraction, volume building and volume concatenation and storage.

In terms of model training, Chong and Tay (2017) refer that the original model was trained with the standard *Adam* optimizer to allow it taking the role of setting the learning rate automatically based on the model's weight update history. In terms of epochs, each training volume was trained for a maximum of 50 or until the reconstruction loss of validation data stopped decreasing after 10 consecutive epochs (validation data being 15% of the training frames randomly chosen). The rectified linear unit (ReLU) has a useful regularization ability but its resulting activations have no upper bound. For this reason, the activation function utilized for the spatial encoder and decoder was the Hiperbolic tangent to ensure the symmetry of the encoding and decoding function, (Chong and Tay, 2017).

Lower level details regarding the framework implementation are provided in section 5.2.

4.2.4.1 Experiments and Results

The first experiments were conducted using the model training parameters mentioned in the paper, section 4.2.4. However, the AUC metric did not stabilize in a fixed value. Nevertheless, other hyperparameter combinations were tested, most of them utilizing another optimizer - stochastic gradient descent (SGD). To have a faster convergence, the learning rate was also increased and the Nesterov variant was selected to make the optimization more responsive to changes. This combination of hyperparamaters showed the best results. The model was trained for 200 epochs and the training and validation losses were saved during model training. In terms of testing, for each resulting model stored during training, the frame reconstruction errors for all the 36 UCSD ped1 testing videos were determined and concatenated into a single array to be compared with the testing set's complete ground truth array. This allowed to calculate the AUC metric for the resulting models on UCSD ped1's test set after each epoch. The evolution of the AUC metric with the increasing number of epochs was performed, showing a continuous evolution of the AUC curve and achieving the highest score of 76.7% for epoch number 105, see figure 4.1.

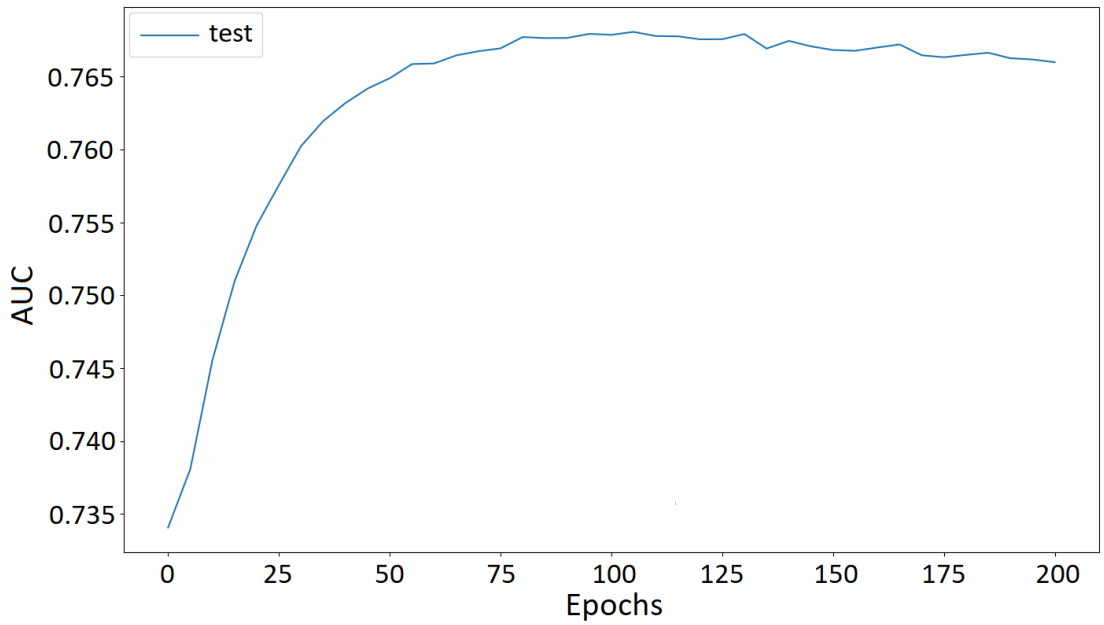


Figure 4.1: AUC evolution with the increase of training epochs for UCSD pedestrian 1 dataset. Maximum score of 76.7% reached on epoch 105.

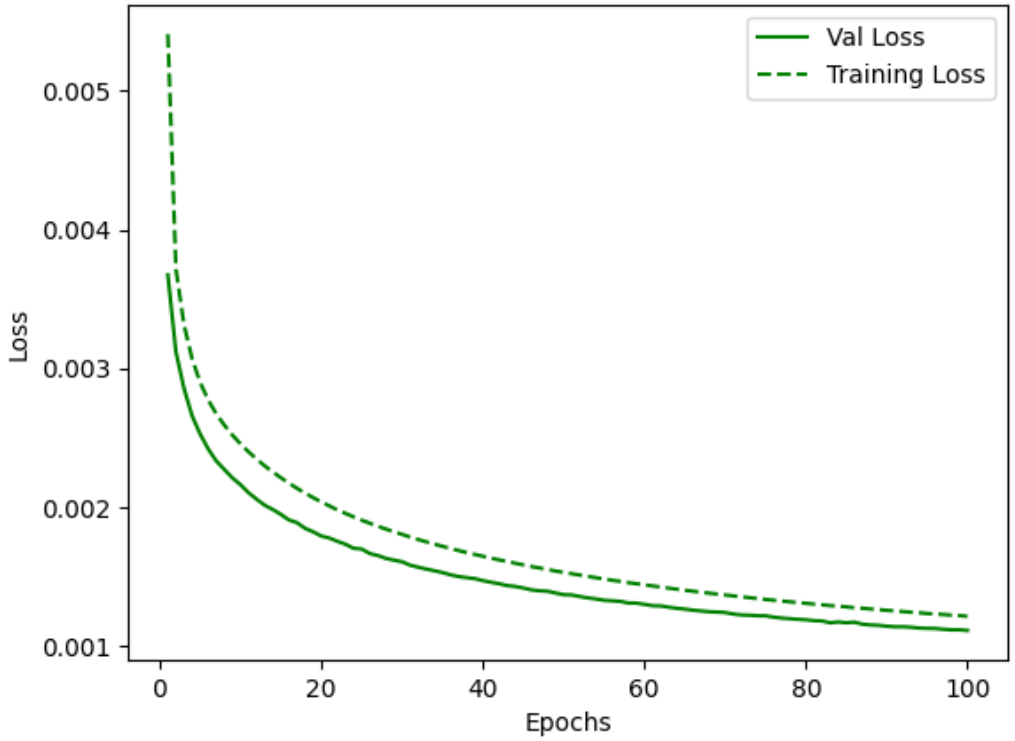
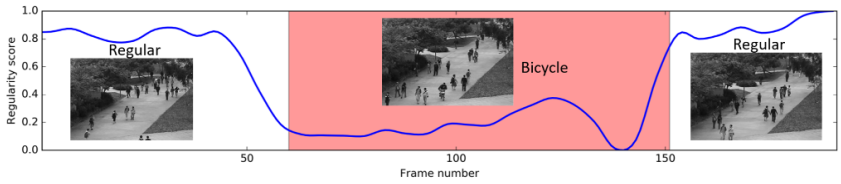
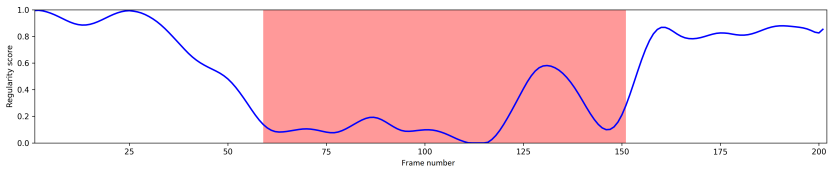


Figure 4.2: Training and validation loss with for each training epoch.

In terms of regularity scores, in figures 4.3 through 4.6 the results from the original paper and this replication on four videos from UCSD ped1's testing set are compared. The regularity scores are superimposed on the testing set's ground truth data where abnormal frames are displayed in red and the normal frames in white. The trends in the scores are all similar, video 1 displaying the largest discrepancies when compared with the original results.

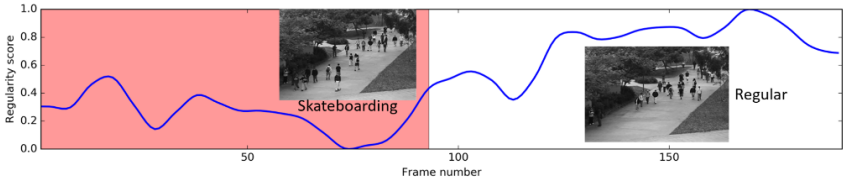


(a) Original

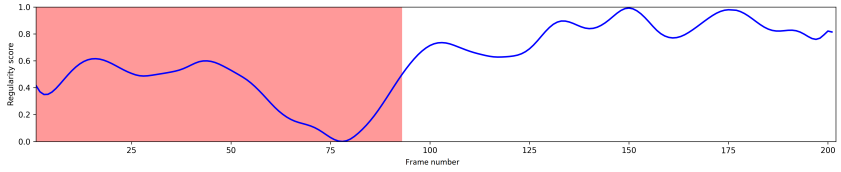


(b) Replication

Figure 4.3: Regularity score (from Chong and Tay (2017)) (a) and regularity score obtained by the replicated framework (b) for testing video 1 of UCSD pedestrian 1.

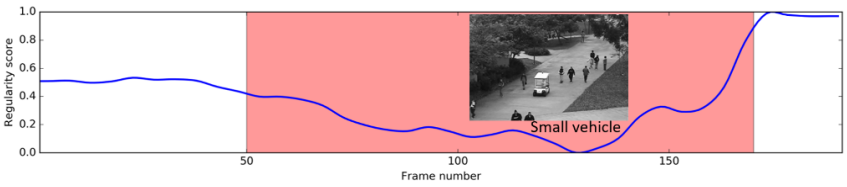


(a) Original

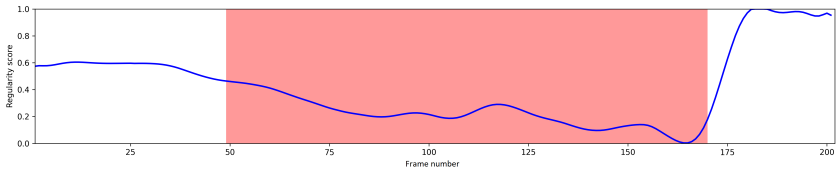


(b) Replication

Figure 4.4: Regularity score (from Chong and Tay (2017)) (a) and regularity score obtained by the replicated framework (b) for testing video 8 of UCSD pedestrian 1.

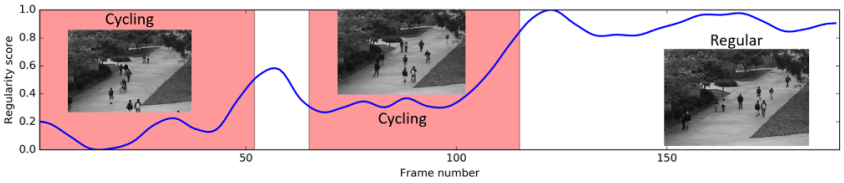


(a) Original

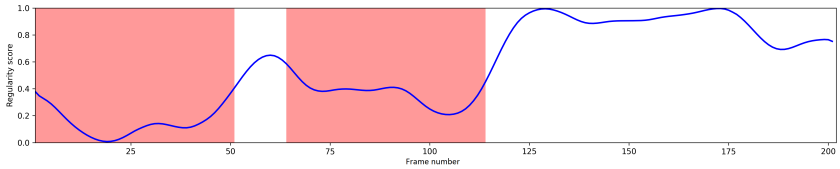


(b) Replication

Figure 4.5: Regularity score (from Chong and Tay (2017)) (a) and regularity score obtained by the replicated framework (b) for testing video 24 of UCSD pedestrian 1.



(a) Original



(b) Replication

Figure 4.6: Regularity score (from Chong and Tay (2017)) (a) and regularity score obtained by the replicated framework (b) for testing video 32 of UCSD pedestrian 1.

The 76.7% AUC score obtained with this experiment is lower than the 89.9% presented in Chong and Tay (2017) for UCSD ped1. However, looking at the regularity scores comparison, it is clear that the replicated framework holds an overall discrimination ability similar to the original. For this reason, this (replicated) framework was selected for addressing the in-vehicle anomaly detection problem. The next chapter touches on implementation details not discussed in section 4.2.4 and presents the main results for the anomaly detection inside vehicles.

Chapter 5

In-Vehicle Anomaly Detection

This chapter covers the adaptation made to the framework described in chapter 4 to detect abnormal human interactions inside vehicles. Details are provided on the Anomaly detection dataset used for training, validation and testing. The implementation of the framework and results are also presented. The chapter is divided in the following sections:

- **Bosch Anomaly detection dataset -** statistics of the dataset in terms of classes, different splits, imbalances and presentation of the data handling scripts, section 5.1.
- **IVS anomaly detection framework implementation -** implementation details of the framework and additional scripts for data processing and gathering, section 5.2.
- **Training and Testing -** explanation of the training and testing procedures as well as imbalanced data counter-measures, section 5.3.
- **Performance -** performance measure of the testing procedure, section 5.4.
- **Results -** presentation of the results obtained for all the different anomaly detection data splits and comparison of the framework with an action recognition model, section 5.5.

5.1 Bosch Anomaly detection dataset

As already mentioned in the previous chapter, the spatiotemporal autoencoder architecture, in this case its replication from section 4.2.4 was adapted to detect anomalies inside vehicles. This framework will be regarded from this point forward as the **IVS anomaly detection framework**.

The dataset used for training, validation and testing was provided by Bosch Car Multimedia and is a subset of a larger dataset. From this point forward, it will be referred as the **In-Vehicle Sensing (IVS) dataset** and contains videos of 9 different actor pairs, performing various activities in the backseat of a vehicle. There are no videos featuring a single actor or more than two actors and for that reason, the anomalies are related to human interactions only.

5.1.1 Labeling

Each video of the **IVS dataset** is labeled in its respective .xml file with timestamps of all the activities elapsed in the video. There are various different actions and all of them can be split into the Non-violent and Violent categories. The actions and the categories they belong to are in table 5.1. Looking at the table, it is clear that there is a fair amount of different interactions. Some actions are related to others - Talk_Right, Talk_Left and Arguing are variations on the Talking class. Aside from these actions, the other actions are self-explanatory besides the Unlabeled class. The Unlabeled class represents activities that do not belong to any of the other classes and are considered Non-Violent because they represent common behaviors of people inside vehicles, such as fastening seat belts, interacting with smartphones, opening/closing windows and many others.

Table 5.1: Actions in the **IVS dataset** split into the Violent and Non-violent categories.

Action	Violent	Non-violent
Touching		✓
Arguing		✓
Talking		✓
Talk_Right		✓
Talk_Left		✓
Handshaking		✓
Hugging		✓
Unlabeled		✓
Slapping	✓	
Fighting	✓	
Punching	✓	
Kicking	✓	
Grabbing	✓	
Pushing	✓	
Elbowing	✓	

To handle the .xml files, python scripts had to be developed. The most important one and developed first was the script to split the videos into smaller clips containing only the labeled actions, figure 5.1. As the resulting clips are stored with the corresponding action's denomination (as the file name), they can be later selected for training (or validation) based only on the action's name without having to deal with any additional files.

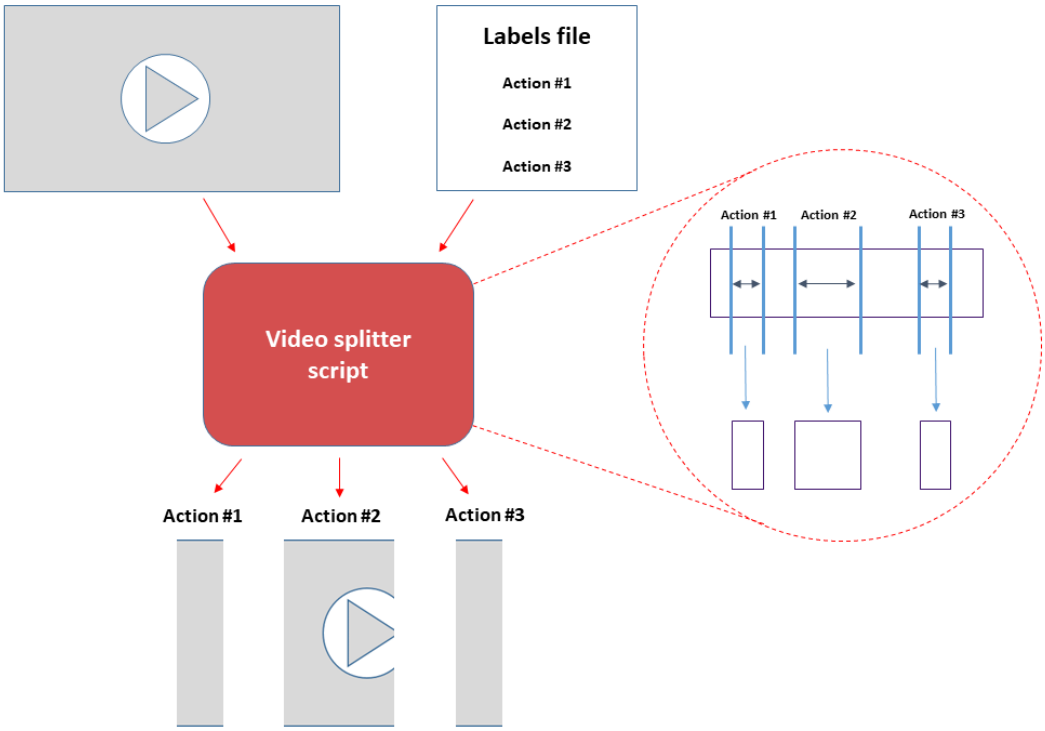


Figure 5.1: Video splitter script - it takes videos and their respective .xml label files and outputs the action clips found in the videos.

5.1.2 Defining the anomalies

Before deciding on what actions should be considered normal or abnormal, the videos from 1 of the 9 actor pairs were selected for testing, comprising the **TEST_SET**, and the videos from the remaining 8 other actor pairs constitute the **TRAIN_VAL_SET**. Besides the Talk_Left, Talk_Right, Handshaking and Elbowing actions, all the other classes featured in table 5.1 are performed in the **TEST_SET** at least once. Table 5.2 reveals the number of occurrences of each class in both datasets.

Table 5.2: Distribution of Actions in the **TRAIN_VAL_SET** and the **TEST_SET**.

Action	**TRAIN_VAL_SET**	**TEST_SET**
Touching	48	74
Arguing	12	8
Talking	318	191
Talk_Right	92	0
Talk_Left	33	0
Handshaking	11	0
Hugging	268	45
Unlabeled	32	1
Slapping	578	107
Fighting	290	28
Punching	212	21
Kicking	166	23
Grabbing	445	43
Pushing	426	232
Elbowing	4	0

To complement the video splitter script mentioned in section 5.1.1 a split generator script was developed. This script creates a train and validation data split according to the actions defined by the user as normal. For each video with a normal action denomination from the **TRAIN_VAL_SET**, it splits it into frames and stores them in the training folder or validation folder of the new dataset.

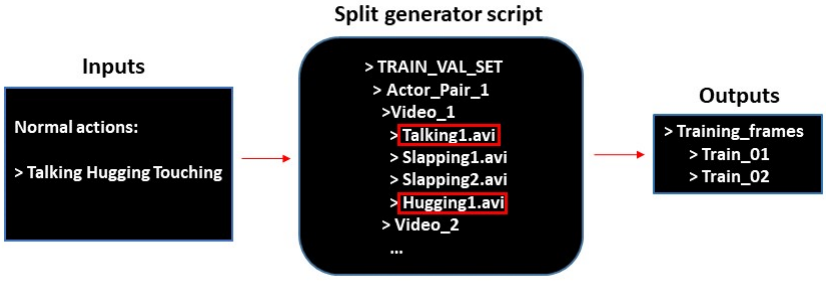


Figure 5.2: Split generator script - it takes the normal classes, searches for their occurrences in **TRAIN_VAL_SET** and stores the frames of each occurrence in the training frames folder.

Making use of the utility scripts mentioned, three variants for Training, Validation and Testing splits were considered for this work:

- **IVS Simple -** This split used a simple data distribution, gathering only the Talking, Talk_Left and Talk_Right as normal actions for an 80/20% training/validation split. Any physical interaction besides these low movement activities was considered abnormal. This dataset is comprised of only one actor pair (the same later used for the **TEST_SET**). As such, only 5 videos were selected for testing, displaying all the classes from the **TEST_SET** except the

Talk_Left, Talk_Right and Hugging actions. This simplification of the anomaly detection problem allowed to validate the proof of concept as it will be later confirmed.

- **IVS Normal -** The goal of this split was to expand on **IVS Simple** by having a varied number of actors for training and validation. Testing was also performed with the same actor pair from the **TEST_SET** which allowed to evaluate the framework on its ability of generalization. As in **IVS Simple**, only the 'Talking', 'Talk_Left' and 'Talk_Right' action clips from the **TRAIN_VAL_SET** were considered for training and validation with a split of 86.5% for training and 13.5% for validation. This selection implied that 3 actor pairs from the **TRAIN_VAL_SET** could now be used for training and validation. Regarding testing, 34 from the 35 videos from the **TEST_SET** were now used except the one that contained the Unlabeled class since the model was not trained with any Unlabeled action clip.

- **IVS Complex -** This split was conceived with the goal of evaluating the framework in a difficult use case, in this case **violence detection**. All of the non-violent action clips from the **TRAIN_VAL_SET** were considered for training and validation with a split of 85% for training and 15% for validation. This selection implied that the normal activities selected for training and validation were no longer mostly static activities but also dynamic non-violent interactions such as the complex Hugging activity and many Unlabeled behaviors. In this case, only the violent classes were considered as anomalies and all of the 35 videos from the **TEST_SET** were considered for testing.

Table 5.3: **IVS anomaly detection framework** data splits statistics.

Split	# Training frames	# Validation frames	% T/V	# Testing frames	# Normal testing frames	# Abnormal testing frames
TRAIN_VAL_SET	1850860	-	-	-	-	-
TEST_SET	-	-	-	220956	-	-
IVS Simple	32366	8300	79.6/20.4	36871	27012	9859
IVS Normal	323284	50574	86.5/13.5	207734	123561	84173
IVS Complex	557648	97986	85/15	220956	160252	60704

Table 5.3 shows statistics on the **TRAIN_VAL_SET**, **TEST_SET**, **IVS Simple**, **IVS Normal** and **IVS Complex** datasets. The **TRAIN_VAL_SET** split corresponds to the videos from 8 of the 9 actor pairs available and has close to 2 million frames. The training sets (Training frames) from **IVS Normal** and **IVS Complex** are different samples from the **TRAIN_VAL_SET** according to their specific use cases and **IVS Simple**'s training set is composed of a small sample of the **TEST_SET**. All of the testing sets (Testing frames) from **IVS Simple**, **IVS Normal** and **IVS Complex** are samples from the **TEST_SET**, **IVS Complex**'s corresponding exactly to the entire **TEST_SET**. The 80/20 train/validation percentages were mostly respected with a few exceptions due to the nature of the validation videos selection method mentioned in section 5.2. The table also shows that the number of normal and abnormal testing frames is different for every dataset. To obtain the skewness of the test sets of **IVS Simple**, **IVS Normal** and **IVS Complex**, the anomalies

were considered the positive samples and the normal frames were considered the negative samples. The skewness of the splits, as defined in section 2.6.6, are **2.74** for **IVS Simple**, **1.47** for **IVS Normal** and **2.64** for **IVS Complex**. Ideally, the skewness of all the testing sets would be close to 1. In section 5.5, the results of the **IVS anomaly detection framework** on these 3 testing splits are presented alongside their balanced versions.

5.2 IVS anomaly detection framework implementation

The major changes when compared with the replicated framework from section 4.2.4 lies on the preprocessing and training phases. As with section 4.2.4, the preprocessing stage first calculates the global mean frame from all the training frames and then all the training, validation and testing frames are converted to grayscale, normalized to have zero mean and unit variance (on a pixel level), resized to 224x224 and subtracted by the global mean frame. Then, for the frames of each training, validation and testing video, 10 frame sequences, or volumes, are constructed and stored separately for later use.

Data augmentation procedures are essential to increase dataset size and mitigate the overfitting phenomenon during training. However, and as mentioned in section 5.3, the three datasets built for training are substantially large and the possible gain in performance obtained by employing a data augmentation technique would not outweigh the extra computational cost. Bearing this in mind, the data augmentation module from section 4.2.4 had to be removed. For a more detailed look at the preprocessing pipeline, see figure 5.3.

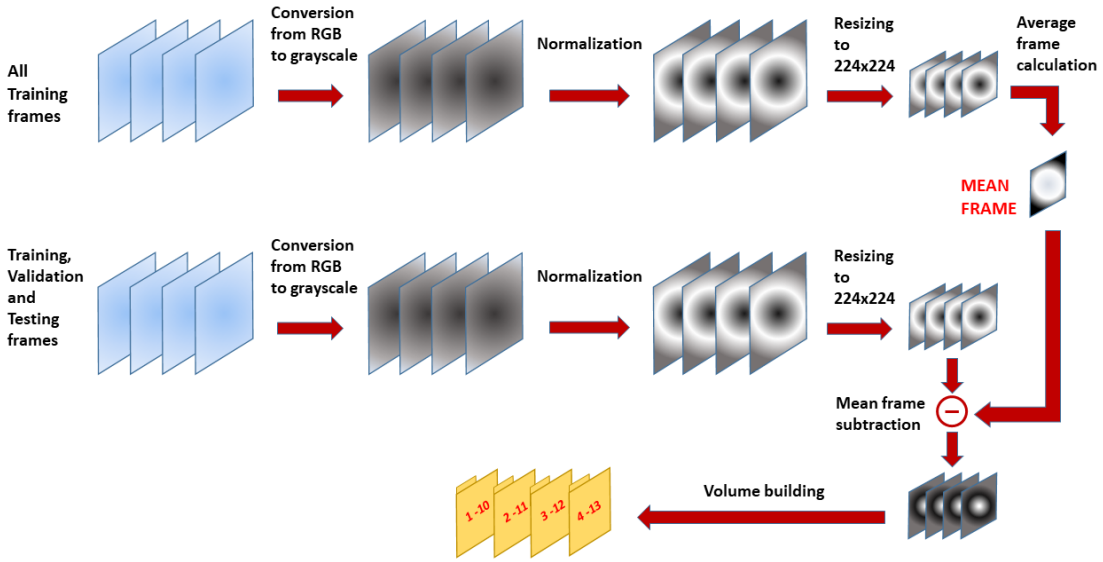


Figure 5.3: Pre processing stage of the IVS anomaly detection framework.

In section 4.2.4, the validation data was randomly selected totalling 15% of the validation and remaining training frames combined. In the **IVS anomaly detection framework**, to further

optimize the training procedure, **generators** were implemented to store the training and validation data sequentially on RAM instead of simply storing it all in its entirety which would eventually lead to memory problems. With this change, the validation data could not be selected randomly. This presented itself more of an opportunity than an issue as it allowed for the careful selection of validation data. Validation data was now picked manually according to the amount of each normal action in the training set, while also aiming at a final 15%/85% validation/training split. After the preprocessing stage, the resulting training and validation frames were fed to each individual thread-safe generator so that during training, the network was trained with small batches in a sequential manner and validated at the end of each epoch.

The type of ConvLSTM-Autoencoder architecture used here uses convolutional layers for spatial encoding/decoding, and ConvLSTM layers for temporal encoding/decoding. In particular, the spatial encoder consists of two 2D convolution layers:

- **Input convolutional layer -** this layer is responsible for capturing the low-level features from the frames. As such, it has a larger number of filters than the second layer - 128 filters of size 11 x 11 with stride 4 x 4 and full padding.
- **Second convolutional layer -** This layer's goal is to extract the high-level features and has half of the previous layer's filters (64), which are also smaller, with a size of 5 x 5, a stride of 2 x 2 and full padding.

As stated in section 4.2.4, these layers are activated by the hiperbolic tangent function. They are implemented using *Keras* `Conv2D()` layer and wrapped with the `TimeDistributed()` wrapper class. This wrapper allows to apply the `Conv2D()` layer to each one of the 10 timesteps, independently. Between these two layers, batch normalization is performed in order to increase the stability of the network. This was implemented with *Keras*' `BatchNormalization()` layer, also wrapped with the `TimeDistributed()` wrapper class.

The temporal encoder/decoder structure is made up of three layers:

- **First convolutional LSTM layer -** This cell is connected with the output of the second convolutional layer and it receives the compressed spatial features of each frame, one at a time. It has the same number of filters, 64, of size 3 x 3 and full padding.
- **Second convolutional LSTM layer -** This layer lies between all the previous encoding layers and all the following decoding layers. It has 32 filters of size 3 x 3 and full padding.
- **Third convolutional LSTM layer -** This is the first of the decoding layers and it is exactly the same as the first convolutional LSTM layer.

These 3 convolutional LSTM cells are implemented using *Keras* dedicated `ConvLSTM2D()` layer. These layers are not wrapped with the `TimeDistributed()` wrapper class because they process each frame sequentially after the spatial encoding of all the 10 contiguous frames.

The spatial decoder is similar to the spatial encoder:

- **First deconvolutional layer -** This layer has the same parameters as the second convolutional layer from the spatial encoder except the number of filters which is equal to the number of filters of the spatial encoder's input convolutional layer (128).
- **Second deconvolutional layer -** This layer has the same parameters of the spatial encoder's input layer besides the number of filters which is 1 instead of 128.

These are implemented with Keras's `Conv2D()` complementary layer - `Conv2DTranspose()` and are also wrapped with the `TimeDistributed()` wrapper class. Between the two layers, a `BatchNormalization()` layer is also added for network stability and symmetry. For a schematic of the network architecture, see figure 3.4.

As previously stated, the network's objective is to reconstruct spatiotemporal volumes. These are 10 frame sequences resulting from the application of a sliding window with stride $S = 1$ and window length $T = 10$ across all the frames of the video, where S stands for the amount of shift by frames in subsequent windows. Input volumes are subtracted by their reconstructed counterparts, resulting in arrays of 10 elements in which each element contains a 224x224 matrix comprising the pixel reconstruction errors of each frame. Then, the euclidean norm is applied on this reconstruction error array resulting in a per volume reconstruction error. For instance, if a video has $V = 200$ frames, the number of volumes and their reconstruction errors resulting from applying the aforementioned sliding window technique is $N = 191$ (equation 5.1).

$$N = \lfloor \frac{V - T}{S} \rfloor + 1 \tag{5.1}$$

Ultimately, the framework's goal is to provide a reconstruction error for each frame and not for each volume. To convert the volume reconstruction errors into frame reconstruction errors, *bilinear* interpolation is applied via *scipy*'s `imresize()` function. *Bicubic* interpolation could also have been used but its longer time of computation outweighed the benefits it provided. It should also be noted that *Bilinear* and *Bicubic* interpolation methods are normally used for 2-dimensional arrays, but can also be applied to 1-dimensional arrays (linear and cubic interpolation). For more details on the frame reconstruction error calculation pipeline, see figure 5.4.

After interpolation, the reconstruction error of each frame determines whether it should be considered abnormal or not according to the defined threshold. The true positive and false positive rates are obtained by setting different frame reconstruction error thresholds in order to calculate the area under the ROC curve. Additionally, an equally important metric of anomaly detection derived from the frame reconstruction errors are the regularity scores. As explained in section 4.2.3, the regularity scores of each video are calculated based on the respective minimum and maximum frame reconstruction errors. The result is a 0 to 1 score for each frame that is more easily understandable than the non-scaled original reconstruction error.

In section 5.5, a slight variation on this architecture is tested. To extract more features from the frames in order to improve pattern recognition, the number of filters of each convolutional layer was doubled. All the other parameters of the network were maintained. This improvement of the

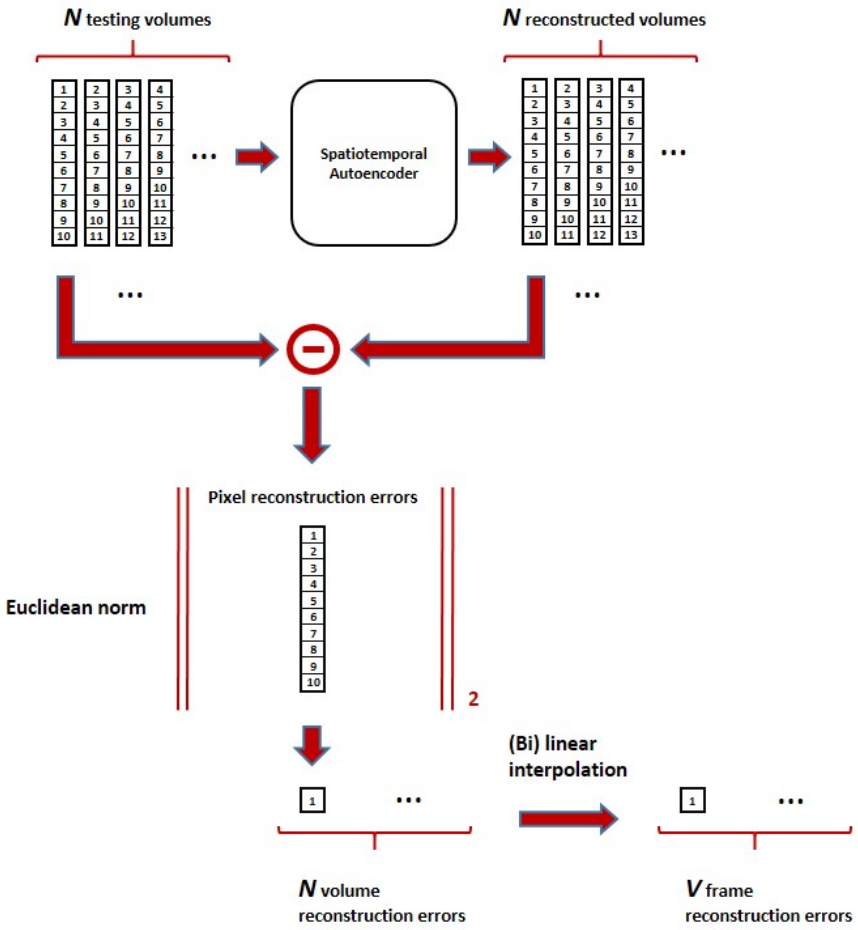


Figure 5.4: Pipeline of the calculation of the frame reconstruction errors.

model increased the training time but the results after testing were also better than the results from the standard model.

In section 4.2, the UCSD ped1 dataset is used to measure the performance of the replicated framework and the ground truth files from its test set are data files, one for each video, that store abnormal event information. Each event is represented in a line of the file as two integer numbers - the initial and final frame number separated by the *whitespace* character. As an example, if an abnormal event occurs from frame 34 to frame 78 of a testing video, in its ground truth file there is a line with the content **34 78**.

To generate the ground truth files for the data splits mentioned in section 5.1.2, a script was developed. The script takes as input the classes to be considered as anomalies and loops through the testing videos of the data split in question (**IVS Simple**, **IVS Normal** or **IVS Complex**) and their respective label files. If a testing video has none of the anomalies, an empty ground truth file is generated but if it contains any of the selected abnormal actions, the resulting ground truth file contains the abnormal events information as mentioned above.

In the **replicated framework**, the regularity score graph of each testing video is outputted with the abnormal frames marked in red according to the corresponding ground truth data. The

same feature is implemented here and to visualize the regularity score in real-time alongside the video footage, a **visualization tool** was developed. The script takes the testing video, its ground truth file and regularity score and produces a new video with all the additional information visible for each frame. A binary ground truth indicator displays the green color if the current frame being presented is normal or the red color if it is abnormal and a gauge with the framework's prediction of how regular the frame is. This tool was essential in validating the performance of the algorithm and perceive how it responded according to the actors' movements. For example outputs, see figures A.5(a), A.5(b) and A.5(c).

5.3 Training and Testing

As previously mentioned, the training process of a deep learning model can be a lengthy one, specially when the the model is fairly complex with more than a million trainable parameters. As explained in section 4.2.2, the model extracts spatial information of ten frames first and then each processed frame is passed sequentially to a convolutional LSTM structure to learn temporal information so that the sequence can be reconstructed. This LSTM-based temporal encoder is the bottleneck of the model in terms of training because it requires sequential processing, disregarding most of the highly parallel structure of the GPU hardware it is trained on. Adding up to the model complexity, having training data of more than 1 Tbyte of information also increases the training time substantially. Bearing this in mind, the model was trained in different hyperparameter configurations in parallel, with the goal of minimizing the mean squared error (MSE) between the input volumes and reconstructed volumes.

The **IVS anomaly detection framework** outputs the AUC and EER metrics for each test set but only the **AUC score** was chosen to measure the performance of the framework. According to the literature, section 2.6.5, the area under the ROC curve is insensitive to class distribution but may mask poor performance in some cases. Nonetheless, this metric was used to have a basis of comparison with state of the art video anomaly detection algorithms. In order to research the effects of test set imbalance for this particular use case, modifications were made on **IVS Simple** and **IVS Complex** testing sets. Only these two splits were altered because **IVS Normal** has a skewness of 1.47 being arguably balanced since UCSD ped1 has a similar skewness, of 1.3. The same cannot be said about **IVS Simple** and **IVS Complex** with skewness levels of 2.74 and 2.64, respectively.

In order to balance the testing sets of the two splits, the videos containing mostly the Talking action and the least number of abnormal frames were removed. There are many other normal actions in the testing set but as the goal was to decrease the number of normal frames while trying to maintain the variety of normal and abnormal classes, only videos with the most abundant normal class (Talking), were removed:

- **IVS Simple -** To balance the testing set from this split, 1 of its videos was removed. The video had 12357 normal frames and 910 abnormal frames. Its removal led to a new skewness level of 1.74 which translates to an improvement of 34% in class distribution. For

further analysis and comparison, this version of **IVS Simple**'s test set will be mentioned as **Balanced IVS Simple**.

- **IVS Complex -** This is a much larger dataset and therefore 6 videos from its testing set had to be removed. The videos totalled 40994 normal frames and 44 abnormal frames. Their removal led to a new skewness level of 1.97 which translates to an improvement of 26% in class distribution. For further analysis and comparison, this version of **IVS Complex**'s test split will be mentioned as **Balanced IVS Complex**.

These changes are summarized in in table 5.4.

Table 5.4: **IVS Simple** and **IVS Complex** test splits and their balanced versions.

Split	# Testing frames	# Normal frames	# Abnormal frames	# Skewness
IVS Simple	36871	27012	9859	2.74
Balanced IVS Simple	24514	15565	8949	1.74
IVS Complex	220956	160252	60704	2.64
Balanced IVS Complex	179918	119258	60660	1.97

5.4 Performance

As previously mentioned, all the frames from the Training, Validation and Testing sets are preprocessed before initiating any of the different stages. It involves calculating the global mean frame from the training data and subtracting it from the training, validation and testing data. Then, for all the sets (training, validation and testing), spatiotemporal volumes (10 frame sequences) are computed and stored for future use. In terms of testing, the main output of the framework are the regularity scores, calculated on a per video basis, and stored in a data file. With the scores of each frame from the data file and the video's ground truth data, the complete video regularity score graph is generated. Having this in mind, the run-time analysis of the **IVS anomaly detection framework** running on a Xeon Gold 6150 CPU and a Tesla V100-SXM2 32GB GPU is in table 5.5. The study can be split into three parts: mean frame subtraction, volume building and prediction (which corresponds to the computation of the regularity score data file). The stage that takes on average the less time to complete is the mean frame subtraction and the one that takes the most time to complete is the prediction stage. Nonetheless, the total time of computation per frame is less than 50 ms. As the system stands now, it cannot work in a real-time scenario because the preprocessing stages are conducted for all the testing videos before the testing stage may begin, but with real-time preprocessing, proper network compression and optimization, it is a realistic scenario in future iterations of the algorithm.

Table 5.5: Run-time during preprocessing and testing (milliseconds/frame).

Statistics	Mean frame subtraction	Volume building	Prediction	Total
Average	9.621	16.75	20.94	**47.3**
Standard deviation	1.01	4.15	1.24	-

5.5 Results

The results displayed here revolve around the three major splits presented in section 5.1.2 and the balanced versions of **IVS Simple** and **IVS Complex** test sets - **Balanced IVS Simple** and **Balanced IVS Complex**.

5.5.1 IVS Simple

As mentioned in section 5.1.2, this first split served mostly to perceive if the Spatiotemporal autoencoder framework, besides detecting anomalies in crowded scenes, would also detect anomalies inside vehicles. To do so, the model was trained on a very restricted sample of videos from the **TEST_SET** that featured the actor pair always dressed in the same clothes and always seated in the same seats. The validation and testing stages were performed on other videos from the **TEST_SET** but with the same conditions of the training stage. It is important to note that the results from this experiment cannot be compared with the results from the following splits because they are a product of an oversimplification of the use case for this application.

5.5.1.1 Discussion

Stemming from what was learned by replicating the framework in section 4.2.4, the model was first trained with the SGD optimizer allied with *Nesterov momentum*. The progression of the AUC score with each training epoch is shown in figure 5.5. It is clear that this hyperparameter combination is not ideal for this dataset. The AUC value starts at 53.88% and decreases in a logarithmical form. Training is automatically interrupted via a *callback* that stops it after no significant improvement. To further investigate if it was simply the case of slowness of the SGD algorithm, a test was made for 100 uninterrupted epochs but the AUC value kept decreasing. In figure 5.6, by looking at the initial progression of the validation loss, it seems likely that the SGD algorithm, aided by the *Nesterov* variant of *momentum*, failed at finding the ideal local minima.

The experiment showed that traditional approaches of optimization seem to struggle with finding local minima during training on this dataset. Alternatively, optimizers with *momentum* and adaptive learning rates should perform better since the automatic adjustments in their parameters' learning rates allow to find local minima faster. Having this in mind, *Adam* was tested next.

Figure 5.7 clearly shows that employing the *Adam* optimizer substantially changed the performance of the anomaly detection framework. Aside the dip on epoch 2, due to the increase in

validation loss, the AUC score grew in a logarithmic type fashion showing no significant increase after epoch 12, the highest being 79.55% at epoch 22.

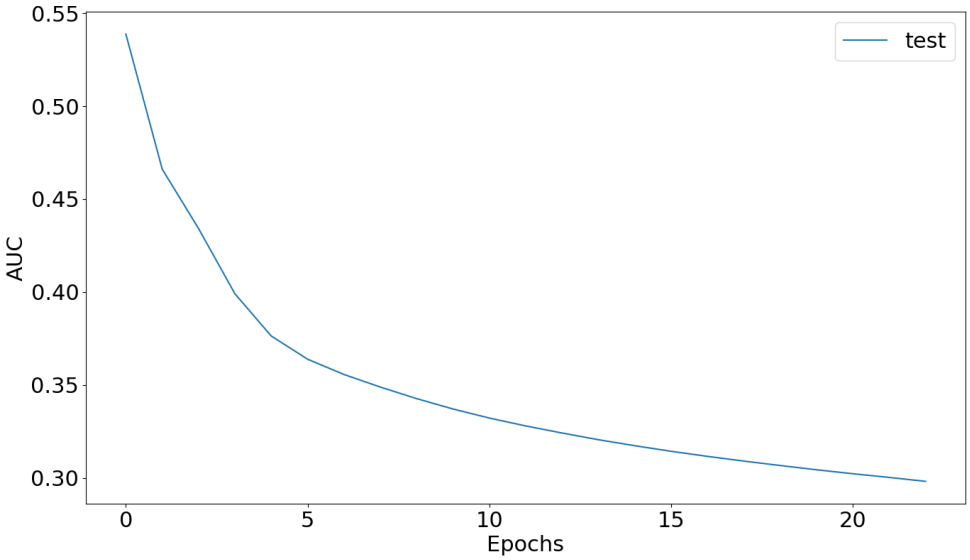


Figure 5.5: AUC score progression for each training epoch on **IVS Simple**, optimized by the SGD algorithm. Maximum score of 53.88% reached on epoch 1.

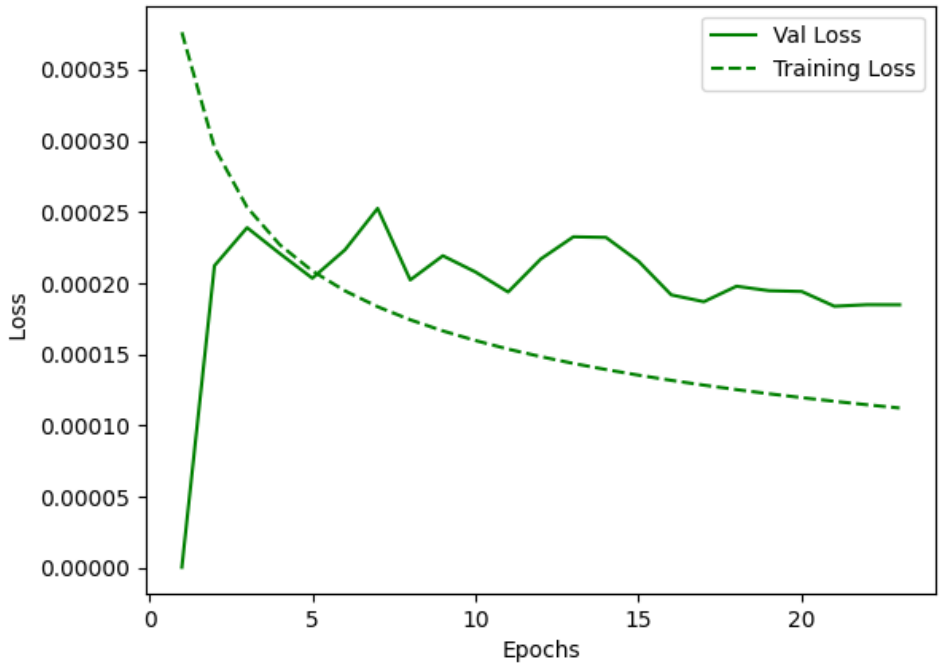


Figure 5.6: Training and validation loss progression for each epoch on **IVS Simple** with SGD optimization.

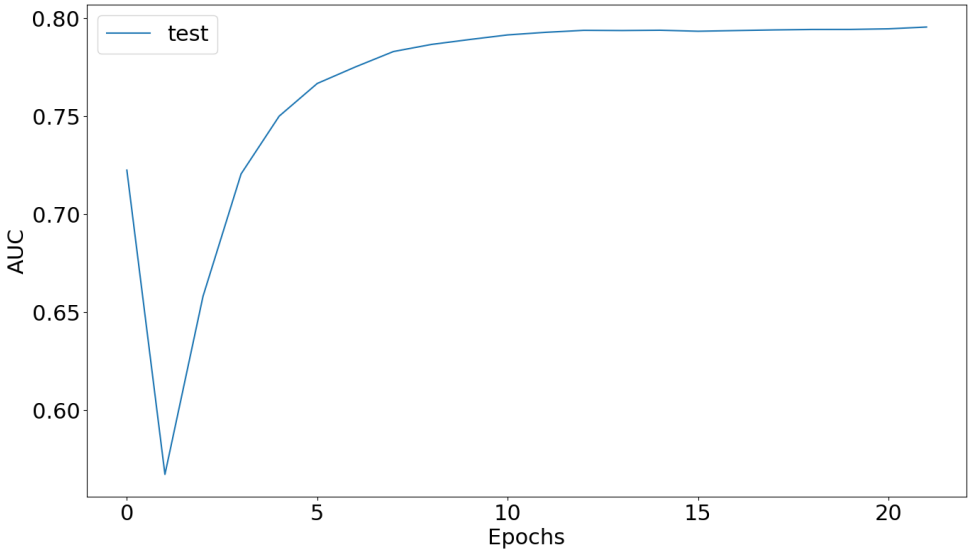


Figure 5.7: AUC score progression for each training epoch on **IVS Simple**, optimized by the Adam algorithm. Maximum score of 79.55% reached on epoch 22.

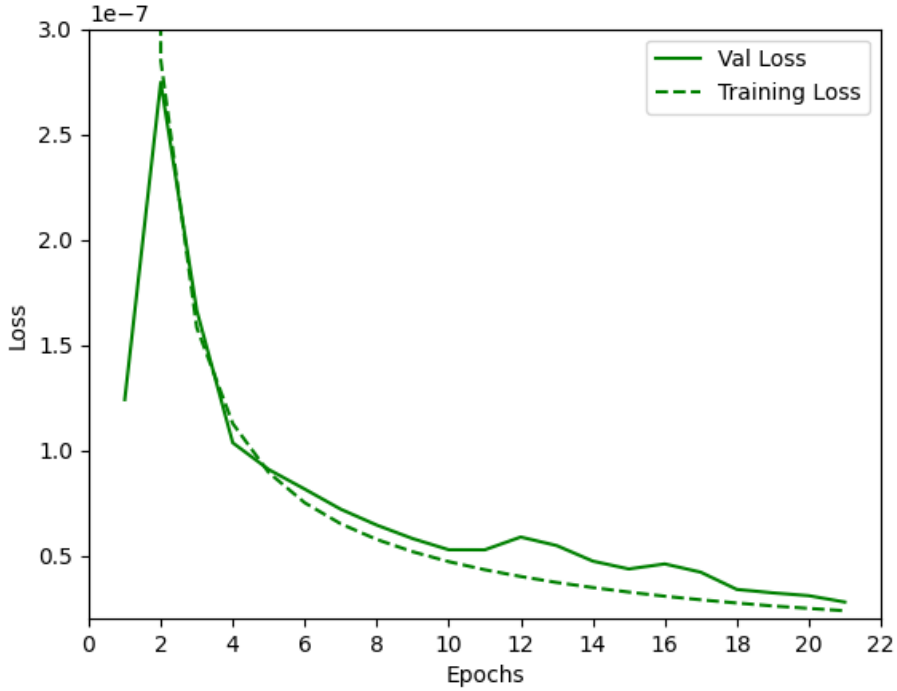


Figure 5.8: Training and validation loss progression for each epoch on **IVS Simple** with Adam optimization.

Another optimizer was also tested - the *Nadam* optimizer which is a variant of *Adam* in the sense that it is accelerated by *Nesterov*'s *momentum* instead of the regular *momentum*. As expected, the performance is similar to the one presented by *Adam* but the AUC progression during training shows some irregularities resulting in a highest AUC value of 80.53% for epoch 11, see figure 5.9. Despite showing a maximum AUC value 1% higher than the highest score obtained by the *Adam* optimizer, the model trained with the *Adam* optimizer presented a better progression during training. For this reason, all of the other splits were trained with the *Adam* optimizer.

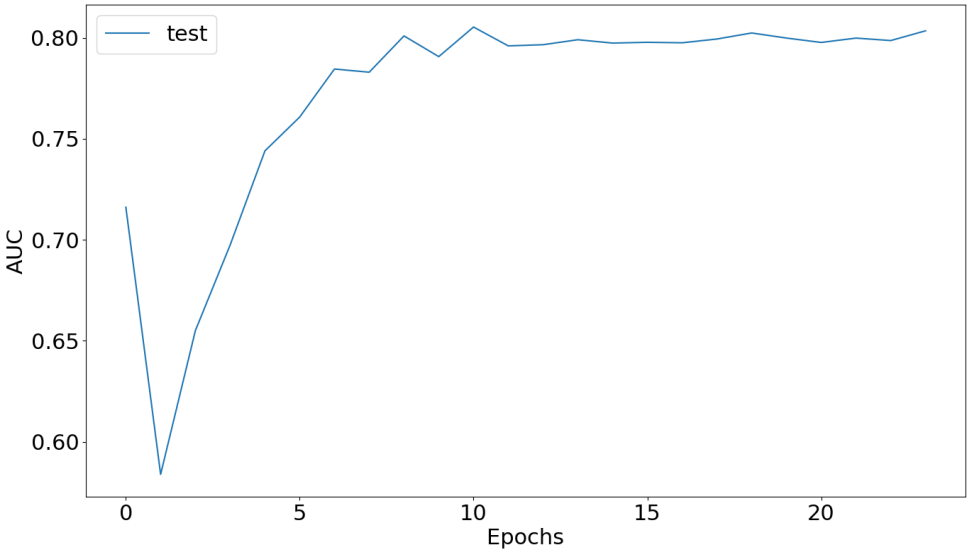


Figure 5.9: AUC score progression for each training epoch on **IVS Simple**, optimized by the Nadam algorithm. Maximum score of 80.53% reached on epoch 11.

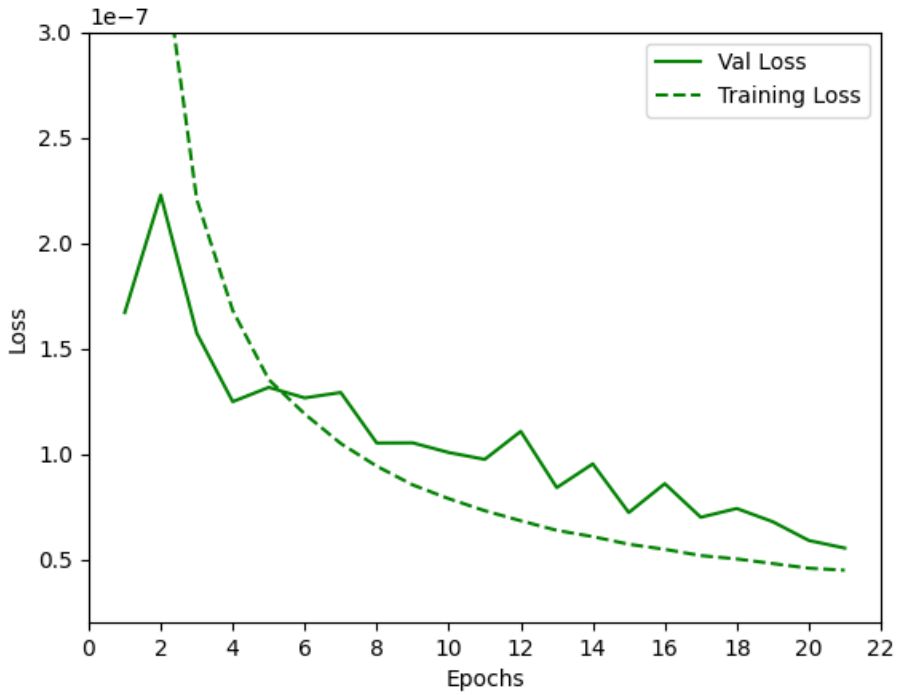


Figure 5.10: Training and validation loss progression for each epoch on **IVS Simple** with Nadam optimization.

5.5.1.2 Impact of imbalanced data

As previously mentioned, a balanced version of **IVS Simple**'s testing split was devised - **Balanced IVS Simple**. In figure 5.11, the performance of the framework on this testing split is compared with the performance of the framework on the original version of **IVS Simple**'s testing split. It is clear that the model displays a slightly worse performance on **Balanced IVS Simple** but with a very similar AUC score evolution trend. On the balanced testing split, the highest AUC score occurred also at epoch 22 but with a value of 77.16% which reflects a 2.39% performance difference when compared with the AUC score obtained for the same epoch on **IVS Simple**'s testing split. The next section covers **IVS Normal** which has a reasonably balanced testing set. **IVS Complex**'s section 5.5.3.2 covers this problematic again and as it represents the use case in a more realistic way, using separate actor pairs for training and testing as well as a larger group of classes, proper conclusions regarding this topic are made.

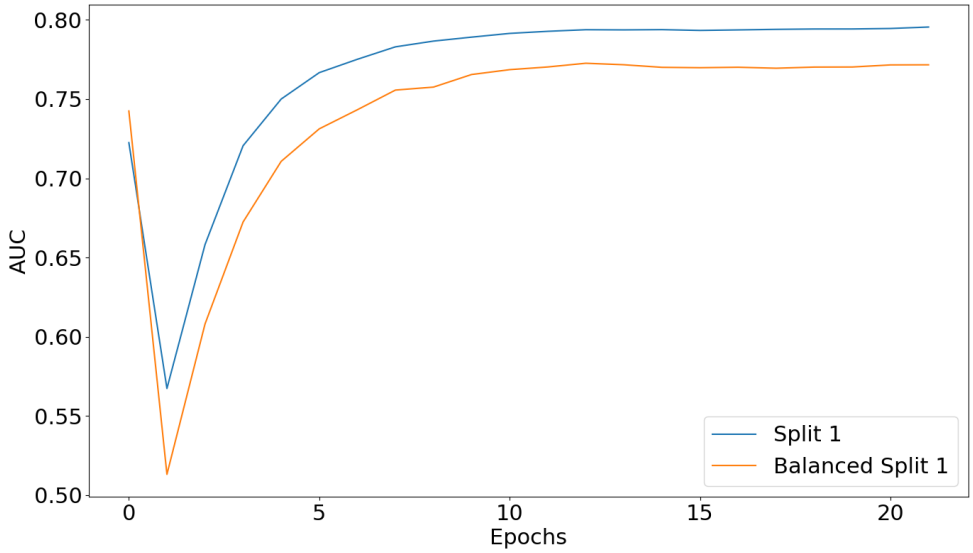


Figure 5.11: AUC score comparison for **IVS Simple** and **Balanced IVS Simple**. Maximum scores of 79.55% and 77.16%, both on epoch 22, for **IVS Simple** and **Balanced IVS Simple**, respectively.

5.5.2 IVS Normal

This split considers any action from classes outside Talking, Talk_Left and Talk_Right as anomalies. This is a simplified approach to the anomaly detection problem inside the vehicle but contrary to **IVS Simple**, it covers a more realistic use case since the system is trained and tested on videos from different actor pairs.

5.5.2.1 Discussion

As discussed in the results from **IVS Simple**, this split and **IVS Complex** were only trained and optimized by the *Adam* optimizer as it showed the best overall performance. However, when the framework was trained allied by the *Adam* optimizer with its default learning rate, the AUC score did not progress as well as it did during the testing of **IVS Simple**, in the same conditions. The AUC score stabilized below 60% which did not allow for acceptable discrimination between normal and abnormal behavior. This was to be expected as the training set from **IVS Normal** is very different from the training set from **IVS Simple**. To allow for slower training, the learning rate of the *Adam* optimizer was reduced and the model was trained and tested again. Figure 5.12 shows a far better AUC score progression with the highest value of 74.88% for epoch 31. As in the previous training schemes, training was interrupted after 10 consecutive epochs of no significant improvement but since the learning rate was reduced, it took more epochs to get a stable AUC score.

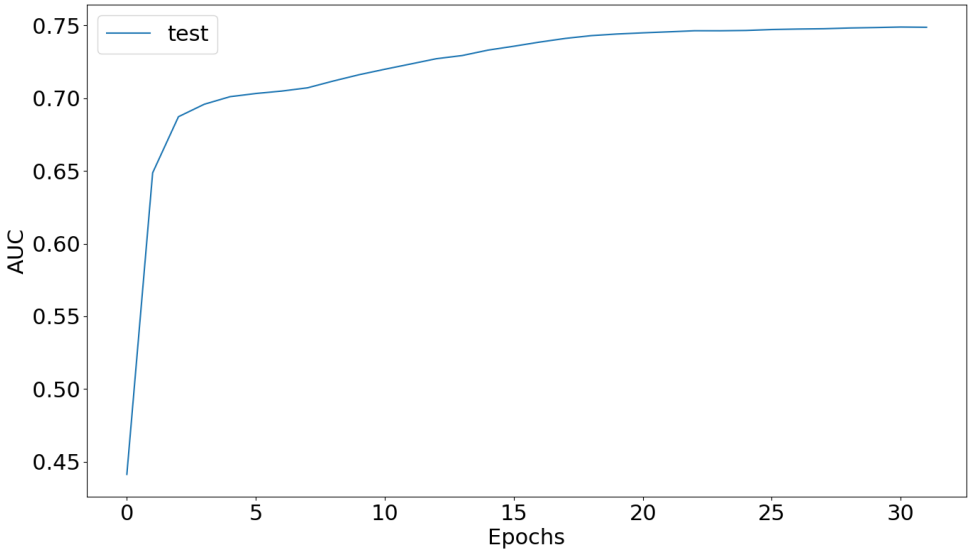


Figure 5.12: AUC score progression for each training epoch on **IVS Normal**, optimized by the Adam algorithm with a reduced learning rate. Maximum score of 74.88% reached on epoch 31.

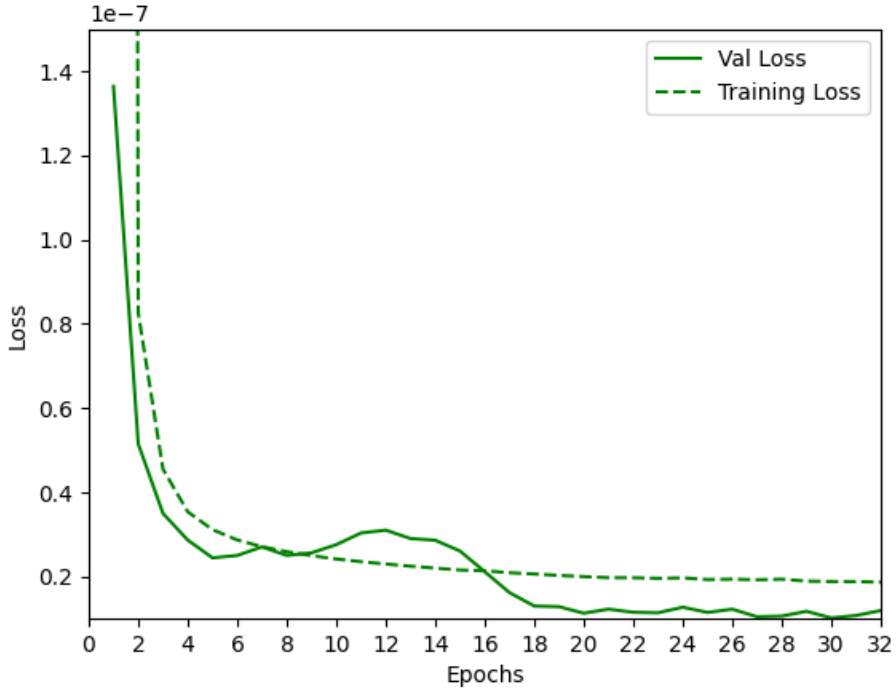


Figure 5.13: Training and validation loss progression for each epoch on **IVS Normal** with Adam optimization with reduced learning rate.

This is a functioning video anomaly detection algorithm whose goal is to alert for any action that deviates too much from casual conversations between passengers in the backseat of vehicles. The regularity scores outputted by the framework on this dataset are much more noisy than the ones obtained when testing on UCSD ped1. The fluctuations on the score may be due to the fast hand gestures the actors make from time to time. In figure 5.14, regularity scores of 4 testing videos from **IVS Normal** are displayed. To aid in the analysis of the results, the 0.5 and 0.7 thresholds were also added in red and green, respectively.

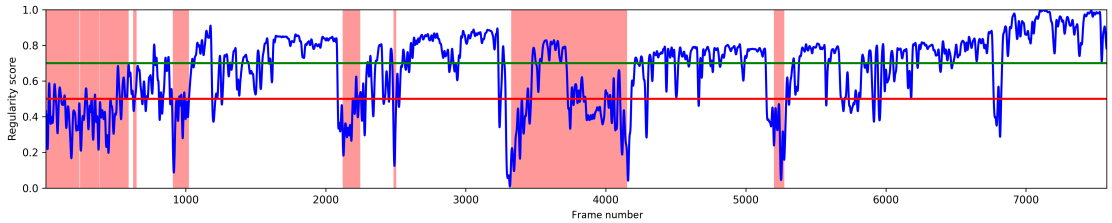
- **Video 9 -** In this video, the abnormal actions performed between the two actors are Fighting, Grabbing, Slapping and Touching. Considering a threshold of 0.5, several abnormal activities would be correctly detected, particularly the Slapping actions starting at frame numbers 901 and 2450. The same cannot be said for the Grabbing action starting at frame 3277 that would be detected correctly initially as one of the actor quickly grabs the other's arm, but as the regularity score increases when the actors stop waving their arms, a significant portion of the interaction would be incorrectly classified. With a 0.7 threshold, the abnormal activities classification would improve but the number of false positives would increase substantially as more normal frames would be classified as abnormal.

- **Video 19 -** The abnormal actions performed between the two actors in this video are Kicking, Pushing, Slapping and Touching. Here, the abnormal actions were differentiated from the normal ones, but the regularity score did not decrease as much as it should for a clear discrimination. As such, choosing a 0.7 instead of a 0.5 threshold would greatly increase the true positive rate with a very small increase in the false positive rate.

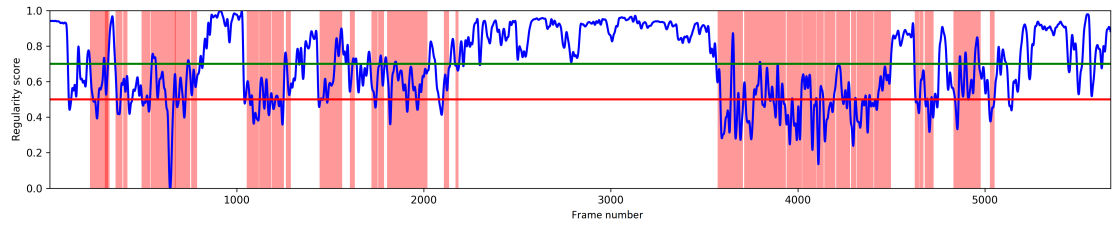
- **Video 22 -** Here, Punching is the abnormal action performed between the two actors. In this particular video, the framework showed the best discrimination from all the testing set decisively detecting the Punching action performed repeatedly for the first 465 frames of the video for either a 0.5 or 0.7 threshold.

- **Video 25 -** Besides some stationary moments, this video only includes Fighting sequences between the two actors. The framework showed good discrimination on the first two fighting sequences but the last sequence showed a slightly higher average regularity score. As with the test video 22, a threshold of 0.7 would lead to a greater number of frames being correctly classified as abnormal, with a residual increase of the false positive rate.

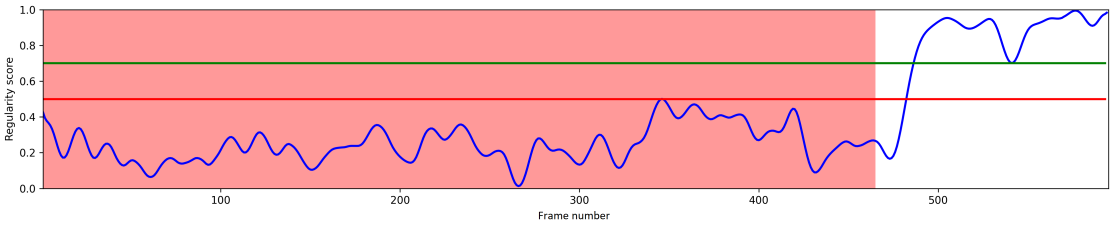
For a clearer picture of the AUC curve for epoch 31, see figure A.1 in appendix A.



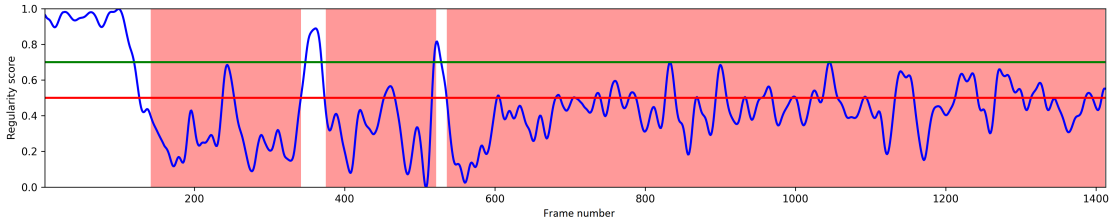
(a) Video 9



(b) Video 19



(c) Video 22



(d) Video 25

Figure 5.14: Regularity scores obtained by the IVS anomaly detection framework for testing videos 9, 19, 22 and 25 of **IVS Normal**.

5.5.3 IVS Complex

This is the split that depicts the anomaly detection problem in a more broader sense with violent actions marked as anomalies. As so, it can be considered as the dataset that more closely represents the intended anomaly detection scenario.

The model was trained with all the non-violent class occurrences from the **TRAIN_VAL_SET** which took inevitably more time to train. The testing videos have less anomalies when compared with **IVS Normal** because then, the majority of classes were deemed abnormal. The results on

IVS Complex are discussed first and then, a comparison is made between the difference in AUC scores from **IVS Complex**'s testing split and its balanced version - **Balanced IVS Complex**.

5.5.3.1 Discussion

Looking at figure 5.15, it is clear that the framework performed better on this testing split than on **IVS Normal**'s. Where **IVS Normal** topped at a score of 74.88% for epoch 31, this time around, the highest score for **IVS Complex**'s test split is of 78.89% on epoch 14. Having a larger variety of actions on the training set seems to allow the network to learn that certain sudden movements can be considered normal, and to discriminate according to the actions themselves instead of how static the scene may be. In testing video 13, figure 5.17, the Grabbing, Pushing and Slapping abnormal sequences occur. The initial 2000 frames correspond to the normal actions of Touching, Talking and the unlabeled action of unlocking the seat belt. The regularity score average value maintained itself above 0.5 in response to these actions but as with the results from **IVS Normal**, the 0.7 threshold should display a better discrimination ability, in this case a higher TPR rate with an increase in the FPR, when compared with the 0.5 threshold.

Using the **visualization tool** referenced in section 5.2, it can be seen in figures , A.5(b) and A.5(c) that the framework correctly detected the abnormal actions of Slapping and Punching, respectively.

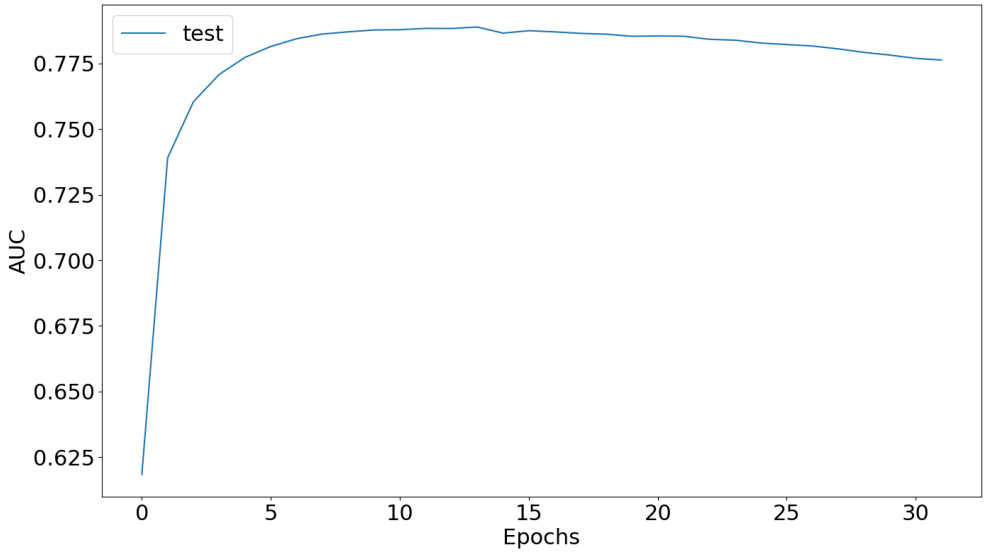


Figure 5.15: AUC score progression for each training epoch on **IVS Complex**, optimized by the Adam algorithm with a reduced learning rate. Maximum score of 78.89% reached on epoch 14.

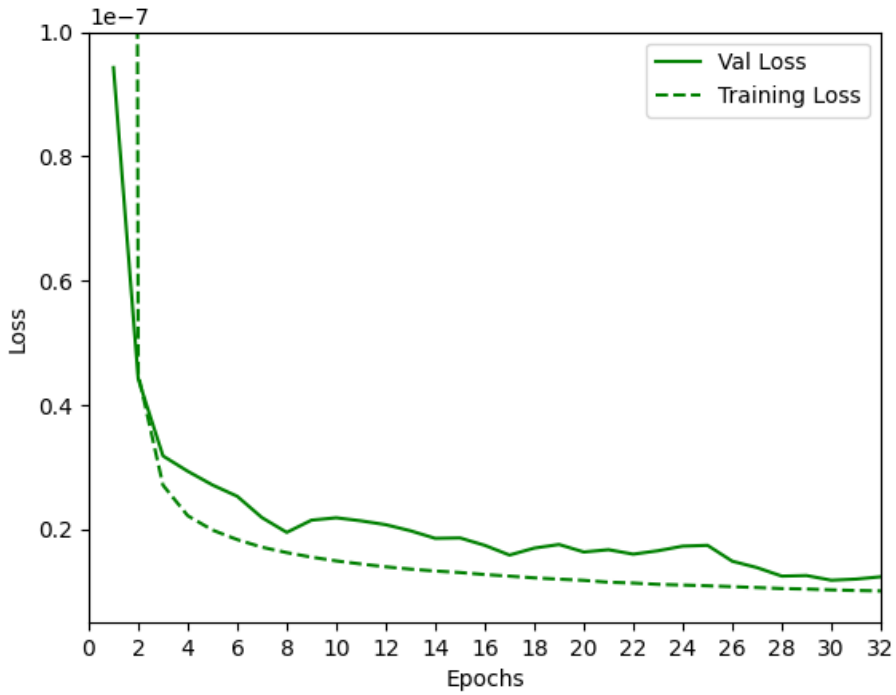


Figure 5.16: Training and validation loss progression for each epoch on **IVS Complex** with Adam optimization and a reduced learning rate.

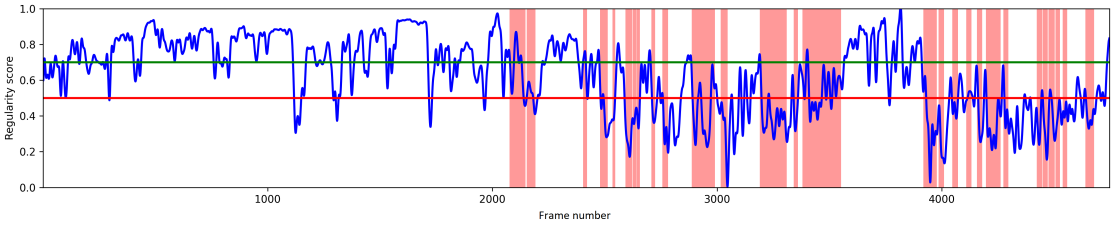


Figure 5.17: Regularity score obtained by the IVS anomaly detection framework for testing video 13 of **IVS Complex**.

5.5.3.2 Impact of imbalanced data

As previously mentioned, a balanced version of this testing split was devised - **Balanced IVS Complex**. In figure 5.18, the performance of the framework on this split is compared with the performance of the framework on the original version of **IVS Complex**'s testing split. As with **IVS Simple**, the model displayed a slightly worse performance on **Balanced IVS Complex** than on **IVS Complex** but the score evolution trend is very similar. On the balanced test split, the highest AUC score occurred at epoch 12 with a value of 77.59% which is a 1.3% performance difference when compared with the AUC score of 78.89% obtained for epoch 14 on **IVS Complex**. The impact of class imbalance on the AUC score of **IVS Simple** was higher (2.39%) because the improvement on class distribution was also higher (34% vs 26% for **IVS Complex**). Either way,

the reduction in AUC score was minimal for the two splits which is why it can be concluded that the AUC metric, for this particular scenario, is not particularly sensitive to class imbalance.

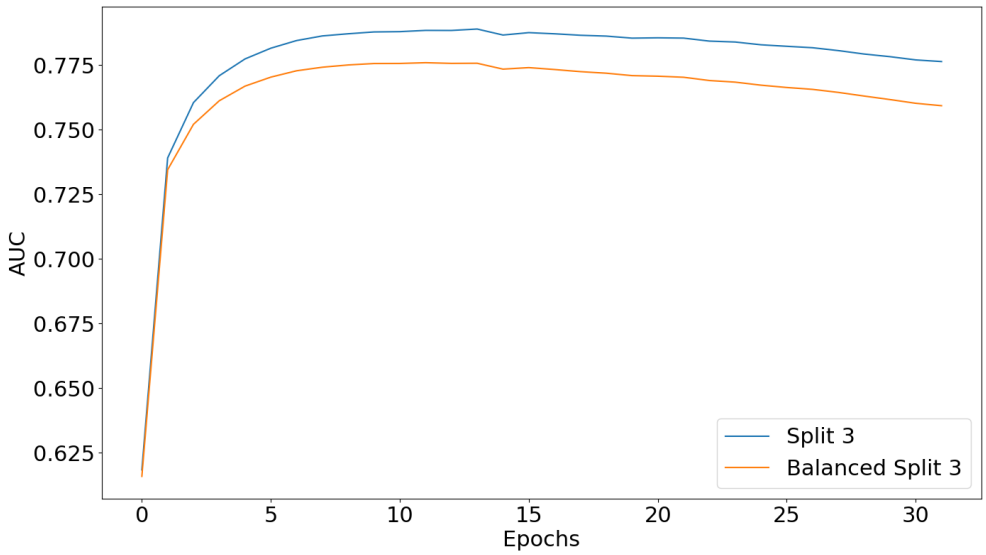


Figure 5.18: AUC score comparison for **IVS Complex** and **Balanced IVS Complex**. Maximum scores of 78.89% on epoch 14 and 77.59% on epoch 12, for **IVS Complex** and **Balanced IVS Complex**, respectively.

The AUC curves for epoch 14 of **IVS Complex** and epoch 12 of **Balanced IVS Complex**, can be seen on figures A.2 and A.3 in appendix A.

5.5.3.3 IVS anomaly detection framework 2.0

As mentioned in section 5.2, to measure the impact of the number of convolutional filters in feature extraction and pattern recognition, a variation of the spatiotemporal autoencoder architecture was also tested. In figure 5.19, the AUC score obtained by the altered model after each training epoch on **IVS Complex** is compared with the AUC score obtained by the original framework. The network alterations did not improve the final AUC score significantly, 79.1% as the new maximum value, but reached it in fewer epochs than the original network - epoch 9 instead of epoch 14 on the original network at 78.89%. As this improvement is not significant, this model was not used in the next sections.

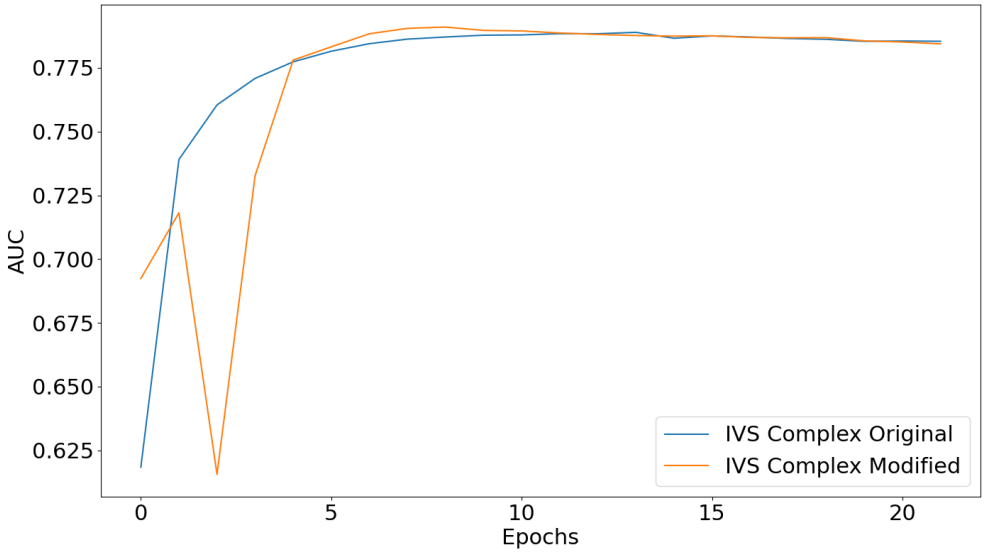


Figure 5.19: AUC score comparison for the standard IVS anomaly detection framework and the modified IVS anomaly detection framework on **IVS Complex**. Maximum scores of 78.89% on epoch 14 and 79.1% on epoch 9 for the standard and modified frameworks on **IVS Complex**.

5.5.3.4 Comparison with an action recognition framework

To further validate the **IVS anomaly detection framework**, its results on **IVS Complex**, section 5.5.3.1, were compared with the results from an action recognition model from Bosch. The objective of this particular action recognition system is to detect violent (abnormal) actions. However, the action recognition model is trained on all the labeled actions from the **TRAIN_VAL_SET** unlike the **IVS anomaly detection framework**. The action recognition model outputs a 0 to 1 score in which lower values represent predictions for violent frames and higher values represent predictions for non-violent frames. The regularity score, in this particular use case, represents the same as the non-violence score which allows for a proper comparison of the two solutions. Due to the focus of the action recognition solution on the performance on each specific action, an earlier comparison between the two approaches was made based on how well each classified an action from the test set as violent (abnormal) or not. To do so, the best performing anomaly detection model on **IVS Complex**'s test set, with an AUC score of 78.89%, was tested and the error rate (ERR = 1 - ACC) of the model was measured for each specific action, i.e., a concatenation of the regularity scores of all the occurrences of each action was compared with an equally sized ground truth array. Note that the accuracy metric (ACC), in this case the function `accuracy_score()` from the `sklearn.metrics` package, can be safely used here because since it is being employed on each class individually, there are not any related problems with class distribution. To measure the error rate of the frameworks on all classes, the regularity and non-violence scores were converted to 0 or 1 values depending on the imposed threshold. Having an extremely low threshold results in high error rates for the violent actions and low error rates for the non-violent actions.

This happens because any regularity score above the threshold is converted to 1, which means that the majority of abnormal actions are classified as being non-violent, and almost all non-violent actions are correctly classified. Similarly, for high thresholds, most of the non-violent actions with regularity scores below the threshold are misclassified as abnormal actions and violent actions end up being correctly classified as such. For this reason, it is important to set a threshold that ensures the least error rate across all the activities. This phenomenon was clearly observed for the anomaly detection framework but not as much on the action recognition framework that even with high thresholds correctly classified some of the non-violent actions. The lower range of thresholds provide the best performance for the detection of normal actions but as these systems are primarily being used as violence detectors, it is arguably more important to encounter less false negatives than false positives. The system may lead to more false positives but at least the majority of violent activities is correctly detected. Taking this into account, and as predicted by the observations made on sections 5.5.2.1 and 5.5.3.1, the **0.7 threshold** showed the least amount of error rate for the violent actions while presenting fairly low error rates for the detection of non-violent actions. To measure the classification ability of the anomaly detection framework against the action recognition model, error rate ratios were calculated according to equation 5.2,

$$\text{ERR ratio}_{\,a} = \frac{\text{ERR}_{\,anom,\,a} - \text{ERR}_{\,action,\,a}}{\text{ERR}_{\,action,\,a}} \tag{5.2}$$

where *anom* stands for anomaly detection framework, *action* for action recognition model and a represents an action from the **IVS dataset**.

In the chart on figure 5.20, the two algorithms are compared according to the Error rate ratio. It is clear that the approach of anomaly detection has a lower error rate when compared with the action recognition approach on all the abnormal/violent actions except for the Punching action, but presents a much higher error rate for the Arguing and Talking activities. Hugging and Touching were the non-violent classes that also presented a lower error rate when compared with the results from the action recognition approach. Without a doubt, the Arguing action is where the anomaly detection framework struggles the most. Despite sharing similarities with the Talking activity, Arguing can be sometimes considered violent, depending on the context. Adding the audio data captured from the scenes would help to discern between Arguing and all the other activities, reducing the number of misclassified occurrences.

Looking at the two algorithms' confusion matrices, tables 5.6 and 5.7 and respective true positive, false negative, true negative and false positive rates, table 5.8, the differences between the two algorithms performances is apparent. Due to the higher error rates when classifying the Arguing and Talking actions, the anomaly detection algorithm presents a much higher false positive rate than the action recognition algorithm. The anomaly detection algorithm has an overall better performance in correctly detecting the violent classes, higher TPR and lower FNR, but struggles at correctly classifying two non-violent activities from which one, Talking, has many occurrences on **IVS Complex**'s testing set. Bearing this in mind, it can be concluded that the anomaly detection

approach is a viable alternative to action recognition algorithms if a slightly high false positive rate is acceptable. If for a particular application, the false positive rate must be very low and there is no problem in a slightly lower true positive rate, then action recognition algorithms are a better approach to detect violent interactions inside vehicles. On the other hand, as there is an overwhelming variety of human behaviors, some cannot be represented in a reasonable number of classes. Moreover, some interactions are so rare that there is not enough data available for training. Anomaly detection algorithms could support action recognition systems to detect these sparse actions and movements that could be later featured in new classes. A possible system architecture featuring a cascade of the two alternatives would be the one on figure 5.21. The action recognition model would discriminate the video sequence first as being non-violent or violent. Then, the anomaly detector, with its knowledge of only non-violent activities, would classify the video sequences marked as non-violent by the action recognition model as non-violent or violent. In theory, this cascade of the two approaches would give more robust results than each system on its own.

An interesting comparison that could not be conducted was the performance of the two algorithms on a test set including edge cases. Having a different number of actors or have them sit in awkward positions are edge cases that were not present in the **TEST_SET**. It would be interesting to test the two algorithms on these unusual circumstances to assess if these would classify them as non-violent activities as they should be.

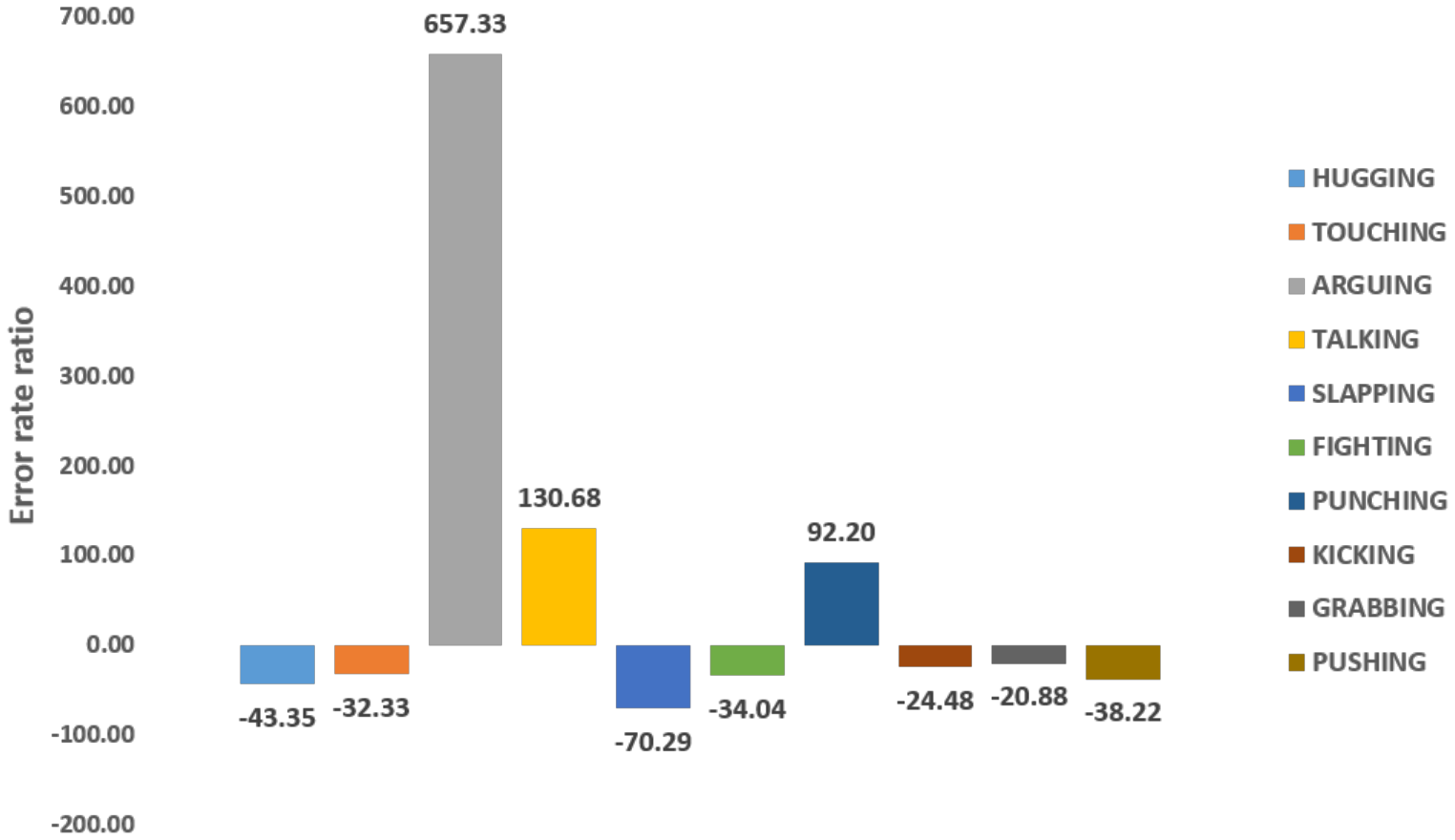


Figure 5.20: Error rate ratio between the anomaly detection and action recognition methodologies.

Table 5.6: Anomaly detector algorithm confusion matrix on **IVS Complex**'s test set.

		Actual Class: V	Actual Class: NV
Predicted Class	V	True Positive 47671	False Positive 52872
	NV	False Negative 13033	True Negative 107380

Table 5.7: Action recognition algorithm confusion matrix on **IVS Complex**'s test set.

		Actual Class: V	Actual Class: NV
Predicted Class	V	True Positive 41717	False Positive 26239
	NV	False Negative 18987	True Negative 134013

Table 5.8: True positive rate (TPR), False negative rate (FNR), True negative rate (TNR) and False positive rate (FPR) of the anomaly detection and action recognition algorithms.

Algorithm	TPR	FNR	TNR	FPR
Anomaly detection	0.79	0.21	0.67	0.33
Action recognition	0.69	0.31	0.84	0.16

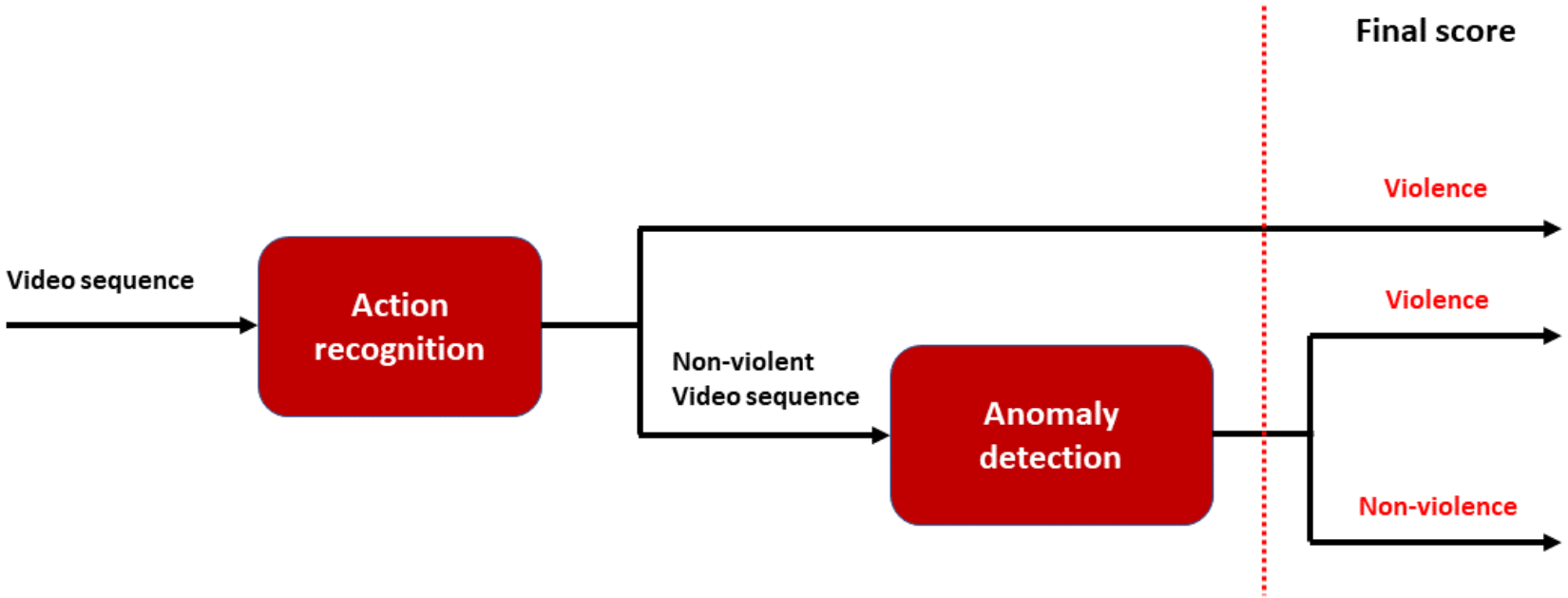


Figure 5.21: Action recognition and anomaly detection cascade architecture.

Chapter 6

Conclusions and Future Work

The emergence of SAVs have led to major interest in the scientific community in developing the necessary systems that compensate the absence of the driver. Besides engine control and steering, interior perception is of utmost importance to ensure the safety of the passengers. This dissertation focused on human interactions inside the vehicles, more specifically the detection of abnormal behaviors.

In chapter 2 the fundamental concepts of anomaly detection were studied. This was an important step in understanding definitions, metrics and the different families of algorithms currently employed to solve the anomaly detection problem.

Chapter 3 delved deeper into the thesis thematic by addressing anomaly detection in video data. The best results found in the literature on the public anomaly detection datasets of UCSD ped1, UCSD ped2 and Avenue hailed from deep learning techniques, particularly autoencoder based frameworks. These semi-supervised deep learning models perform one-class classification - learn how to reconstruct normal frame sequences so that during testing, poor reconstruction leads to the detection of unknown (abnormal) frame sequences. Having this in mind, a thorough analysis was performed on the best performing variants of autoencoder architectures with the goal of selecting the one that could best be adapted to in-vehicle anomaly detection. The Spatiotemporal autoencoder from Chong and Tay (2017) was selected and validated on chapter 4. The spatiotemporal framework had to be trained with a different optimizer and for more epochs, resulting in an inferior AUC score than the one reported in Chong and Tay (2017). Nevertheless, the framework stood out due to its robustness that granted the necessary flexibility and adaptability properties needed for its use in different datasets.

Chapter 5 covered the adaptations that had to be made to the framework and presented details on its implementation. Three datasets with increasing complexity were devised from one of Bosch's internal in-vehicle datasets to understand how well the anomaly detection framework performed. The first dataset, **IVS Simple**, only featured the same actor pair during training, validation and testing. Moreover, as the abnormal classes were defined as being any action different than conversation activities, the framework displayed a higher AUC score than any of the other two datasets, as expected. To increase the difficulty of the anomaly detection task, on **IVS Nor-**

mal, the same normal/abnormal action split was considered but with a larger variety of actor pairs on the training and validation phases. The performance decreased slightly but this test proved the efficiency of the detector on learning spatiotemporal features from a varied group of individuals. Last but not least, **IVS Complex** was constructed to test the system on a violence detection scenario. The system was trained on all the non-violent activities and had the task of detecting violent actions during testing. As observed in section 5.5.3.1, the AUC score obtained on this data split reflects the good discrimination power of the anomaly detection framework on even this more complicated use case. It was also verified that imbalanced test data did not alter the AUC scores significantly. The results of the framework on **IVS Complex** were compared with the results of an action recognition model from Bosch, also working as a violent detector, and it was verified that the anomaly detector presented a reduced error rate on the majority of actions on the **IVS Complex** dataset, but presented a higher false positive rate than the action recognition model. It was concluded that the anomaly detection approach is a viable alternative to action recognition algorithms if a slightly high false positive rate is acceptable. It can also serve to support state of the art action recognition algorithms to detect unknown actions and movements. Still, probably the greatest advantage of adopting an anomaly detection system, particularly a deep learning one, is its semi-supervised nature which allows the definition of only the expected normal data, the use of datasets with reduced labeling, and faster deployment since less time is spent defining hand-crafted features.

6.1 Future Work

In terms of future work, the first task would be to optimize the algorithm even further to reduce the false positive rate on **IVS Complex**. One interesting study that could also be made is to investigate how the sliding window length affects the framework's discrimination ability. Moreover, a less computational expensive data augmentation procedure than the one presented in section 4.2.4 could be pursued to augment the datasets. A solution that would most likely improve results would be the use of a much heavier preprocessing stage where human body pose estimation information would be extracted from the raw images. The new frames could have a dark or white background with the human body pose skeletons of the human subjects on top. This approach would severely reduce the variability in human physiognomy and body composition and help the network to simply focus on the movements. Despite this, the removal of hand and finger information would probably degrade the discrimination ability between all the activities involving them. Alternatively, the input data could be changed to optical flow volumes instead, which according to the work of Duman and Erdem (2019) improved the AUC score by almost 3% when compared with the work of Chong and Tay (2017). This solution could improve the results for parked vehicles by emphasizing motion patterns but would certainly struggle with moving vehicles due to variations on light conditions and constant change of outdoor elements.

Besides the inputs, more drastic changes could be made to the network's architecture than the ones conducted on section 5.5.3.3. One of them would be the addition of more convolutional layers

to the spatial encoder and decoder structures and/or adopt a fusion scheme where the raw frame data and human body pose information could be combined to improve the overall discrimination ability. Audio data could also add another important dimension to the framework and aid these two streams of information.

Appendix A

Appendix

Here lie some figures that are referenced throughout the text.

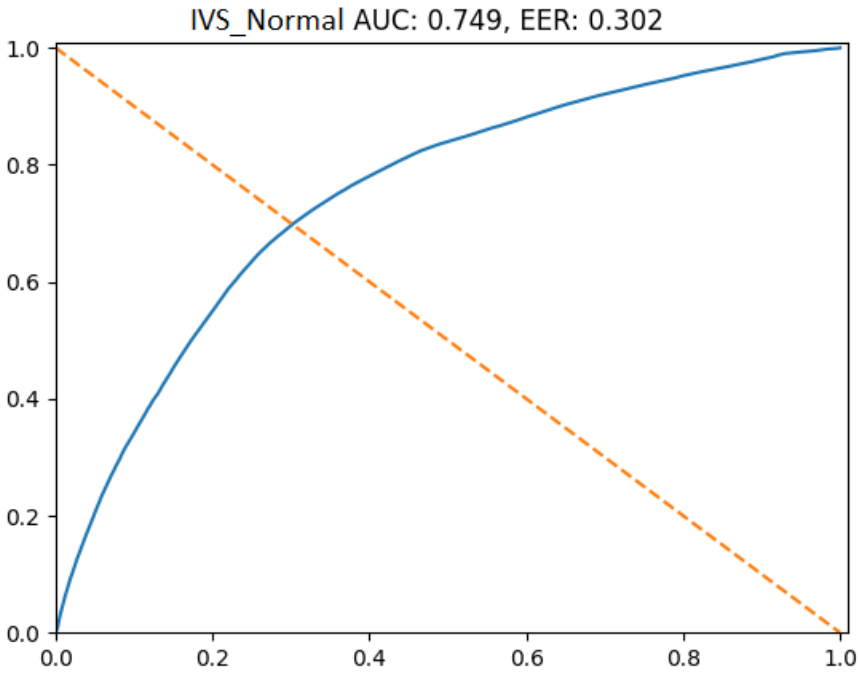


Figure A.1: AUC score for epoch 31 of **IVS Normal**.

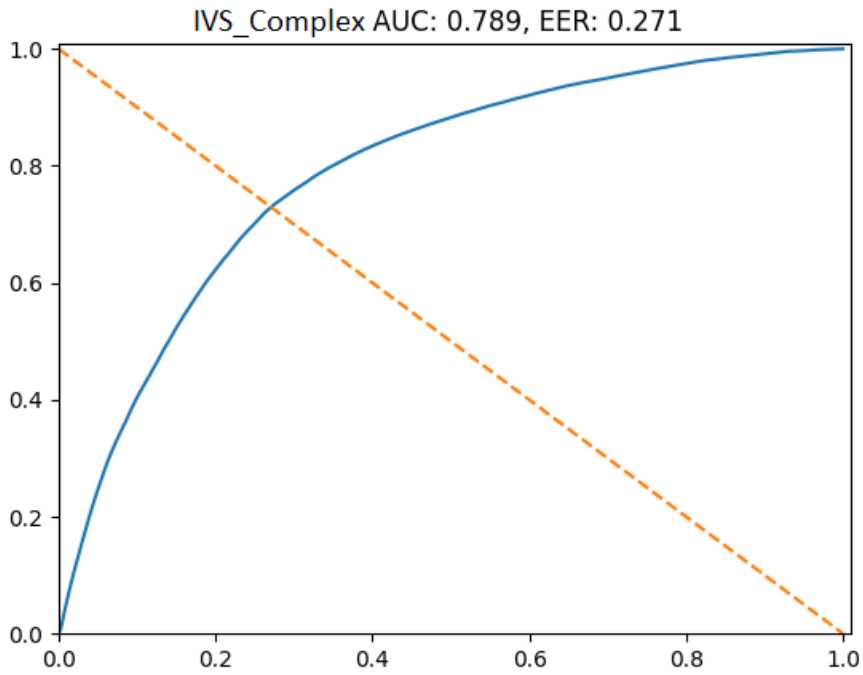


Figure A.2: AUC score for epoch 14 of **IVS Complex**.

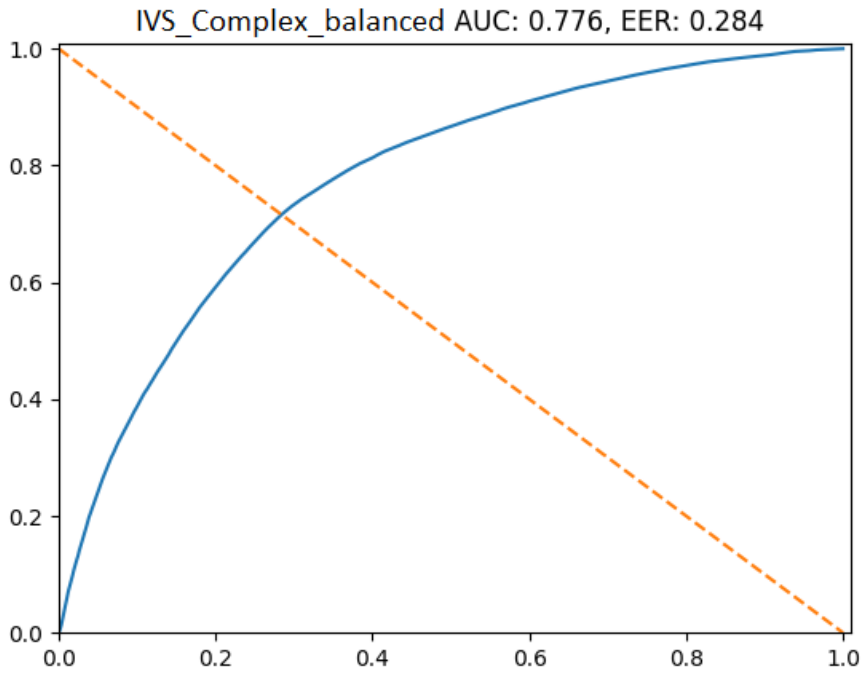


Figure A.3: AUC score for epoch 12 of **Balanced IVS Complex**.

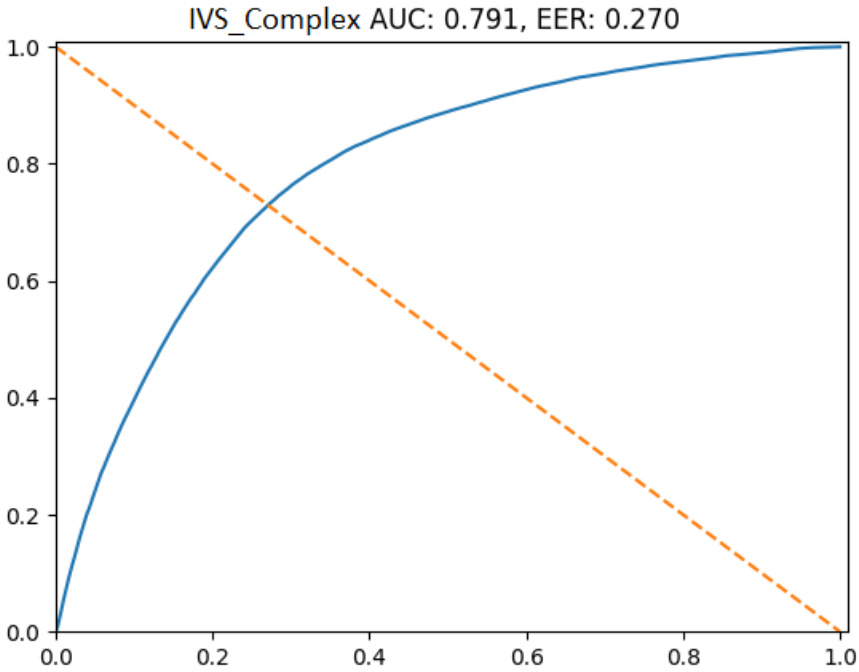
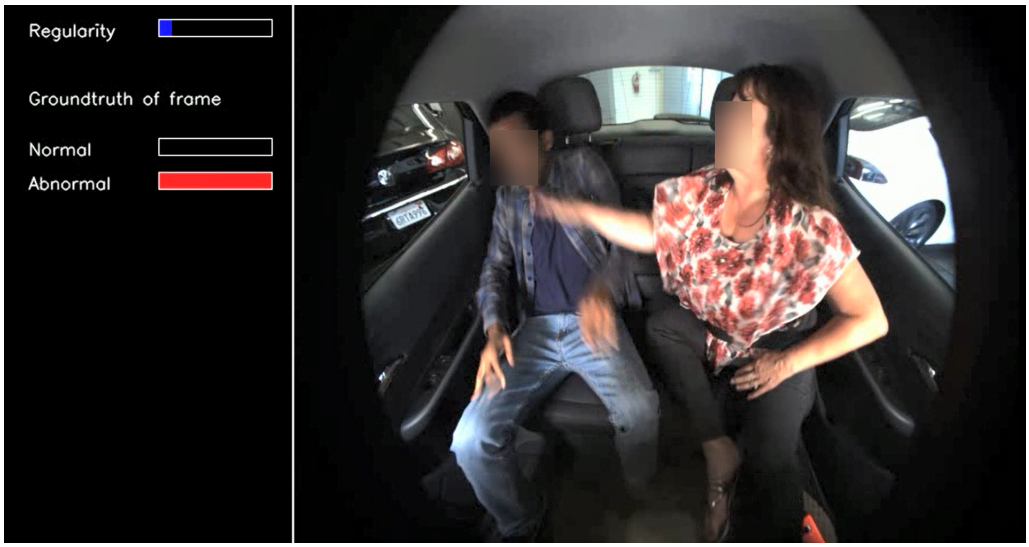
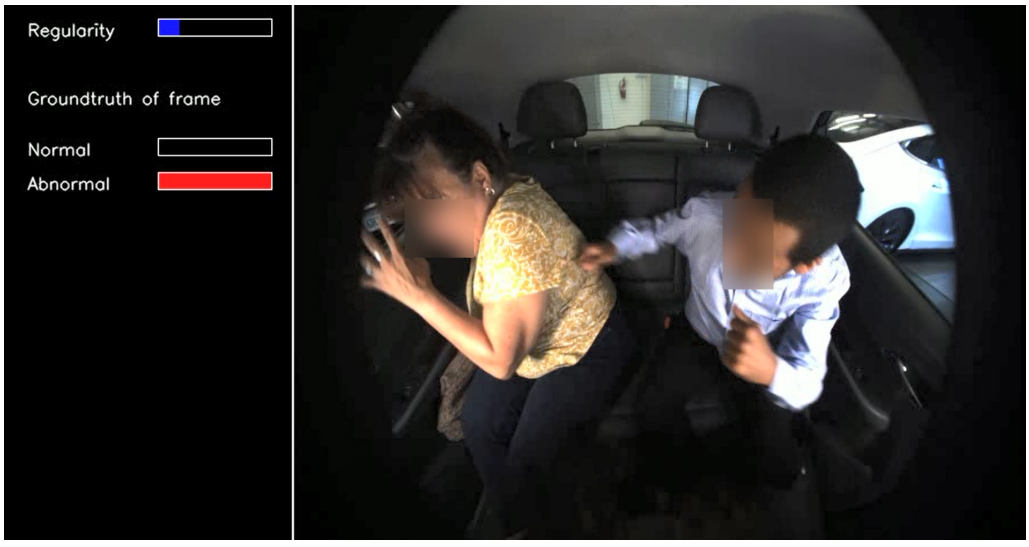


Figure A.4: AUC score for epoch 9 of the modified network on **IVS Complex**.

(a) 'Talking' action from testing video 25 of **IVS Complex**.



(b) 'Slapping' action from testing video 9 of **IVS Complex**.



(c) 'Punching' action from testing video 22 of **IVS Complex**.

Figure A.5: Talking (a), Slapping (b) and Punching (c) actions captured by the visualization tool.

References

Oren Boiman and Michal Irani. Detecting irregularities in images and in video. *International Journal of Computer Vision*, 74(1):17–31, 2007. ISSN 09205691. doi: 10.1007/s11263-006-0009-9.

Varun Chandola, ARINDAM BANERJEE, and VIPIN KUMAR. Survey of Anomaly Detection. *ACM Computing Survey (CSUR)*, 41(3):1–72, 2009.

Sarita Chaudhary, Mohd Aamir Khan, and Charul Bhatnagar. Multiple Anomalous Activity Detection in Videos. *Procedia Computer Science*, 125:336–345, 2018.

Chun Yu Chen and Yu Shao. Crowd escape behavior detection and localization based on divergent centers. *IEEE Sensors Journal*, 15(4):2431–2439, 4 2015. ISSN 1530437X. doi: 10.1109/JSEN.2014.2381260.

Yong Shean Chong and Yong Haur Tay. Abnormal Event Detection in Videos using Spatiotemporal Autoencoder. 1 2017.

Elvan Duman and Osman Ayhan Erdem. Anomaly Detection in Videos Using Optical Flow and Convolutional Autoencoder. *IEEE Access*, 7:183914–183923, 2019. ISSN 21693536. doi: 10.1109/ACCESS.2019.2960654.

Tamara L. Berg Alexander C. Berg Eunbyung Park, Xufeng Han. Combining Multiple Sources of Knowledge in Deep CNNs for Action Recognition. 2016.

Markus Goldstein and Seiichi Uchida. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PLoS ONE*, 11(4):1–31, 2016. ISSN 19326203. doi: 10.1371/journal.pone.0152173.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press.

Mahmudul Hasan, Jonghyun Choi, Jan Neumann, Amit K. Roy-Chowdhury, and Larry S. Davis. Learning Temporal Regularity in Video Sequences. 4 2016.

László A. Jeni, Jeffrey F. Cohn, and Fernando De La Torre. Facing imbalanced data - Recommendations for the use of performance metrics. *Proceedings - 2013 Humaine Association Conference on Affective Computing and Intelligent Interaction, ACII 2013*, pages 245–251, 2013. doi: 10.1109/ACII.2013.47.

B Ravi Kiran, Dilip Mathew Thomas, and Ranjith Parakkal. An overview of deep learning based methods for unsupervised and semi-supervised anomaly detection in videos. 1 2018.

Weinkauf T. Kozlov, Y. Extracting and filtering minima and maxima of 1d functions. 5 2017.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. Technical report.

Hanhe Lin, Jeremiah D. Deng, Brendon J. Woodford, and Ahmad Shahi. Online weighted clustering for real-time abnormal event detection in video surveillance. In *MM 2016 - Proceedings of the 2016 ACM Multimedia Conference*, pages 536–540. Association for Computing Machinery, Inc, 10 2016. ISBN 9781450336031. doi: 10.1145/2964284.2967279.

Weixin Luo, Wen Liu, and Shenghua Gao. Remembering history with convolutional LSTM for anomaly detection. In *2017 IEEE International Conference on Multimedia and Expo(ICME)*, pages 439–444, July 2017.

Amalia Luque, Alejandro Carrasco, Alejandro Martín, and Ana de las Heras. The impact of class imbalance in classification performance metrics based on the binary confusion matrix. *Pattern Recognition*, 91:216–231, 2019.

Manassés Ribeiro, André Eugênio Lazzaretti, and Heitor Silvério Lopes. A study of deep convolutional auto-encoders for anomaly detection in videos. *Pattern Recognition Letters*, 105:13–22, 2018. ISSN 01678655. doi: 10.1016/j.patrec.2017.07.016.

Mehrsan Javan Roshtkhari and Martin D. Levine. An on-line, real-time learning method for detecting anomalies in videos using spatio-temporal compositions. *Computer Vision and Image Understanding*, 117(10):1436–1452, 2013.

Venkatesh Saligrama and Zhu Chen. Video anomaly detection based on local statistical aggregates. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2112–2119, 2012. ISSN 10636919. doi: 10.1109/CVPR.2012.6247917.

Abraham Savitzky and Marcel J.E. Golay. Smoothing and Differentiation of Data by Simplified Least Squares Procedures. 1964.

Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. 9 2014.

David Spulak, Richard Otrebski, and Wilfried Kubinger. Evaluation of PCA, LDA and fisherfaces in appearance-based object detection in thermal infra-red images with incomplete data. In *Procedia Engineering*, 2015. doi: 10.1016/j.proeng.2015.01.480.

Alaa Tharwat. Classification assessment methods. *Applied Computing and Informatics*, 2018.

Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3D convolutional networks. *Proceedings of the IEEE International Conference on Computer Vision*, 2015 Inter:4489–4497, 2015. ISSN 15505499. doi: 10.1109/ICCV.2015.510.

Vikas Tripathi, Durgaprasad Gangodkar, Vivek Latta, and Ankush Mittal. Robust abnormal event recognition via motion and shape analysis at ATM installations. *Journal of Electrical and Computer Engineering*, 2015, 2015. ISSN 20900155. doi: 10.1155/2015/502737.

Emmanu Varghese, Jaison Mulerikkal, and Amitha Mathew. Video Anomaly Detection in Confined Areas. In *Procedia Computer Science*, 2017. doi: 10.1016/j.procs.2017.09.104.

Reza Vosooghi, J. Puchinger, Marija Jankovic, and Anthony Vouillon. Shared autonomous vehicle simulation and service design. *Transportation Research Part C: Emerging Technologies*, 107: 15–33, 2019. ISSN 0968090X. doi: 10.1016/j.trc.2019.08.006.

Tian Wang, Meina Qiao, Zhiwei Lin, Ce Li, Hichem Snoussi, Zhe Liu, and Chang Choi. Generative Neural Networks for Anomaly Detection in Crowded Scenes. *IEEE Transactions on Information Forensics and Security*, 14(5):1390–1399, 2019. ISSN 15566013. doi: 10.1109/TIFS.2018.2878538.

Dan Xu, Yan Yan, Elisa Ricci, and Nicu Sebe. Detecting anomalous events in videos by learning deep representations of appearance and motion. *Computer Vision and Image Understanding*, 156:117–127, 3 2017. ISSN 1090235X. doi: 10.1016/j.cviu.2016.10.010.

Shiyang Yan, Jeremy S. Smith, Wenjin Lu, and Bailing Zhang. Abnormal event detection from videos using a two-stream recurrent variational autoencoder. *IEEE Transactions on Cognitive and Developmental Systems*, 12(1):30–42, 3 2020. ISSN 23798939. doi: 10.1109/TCDS.2018.2883368.