

**UNIVERSIDAD DE INVESTIGACIÓN DE TECNOLOGÍA
EXPERIMENTAL YACHAY**

Escuela de Ciencias Matemáticas y Computacionales

**TÍTULO: Anomaly Detection System in Video Surveillance
using Deep Learning Techniques**

Trabajo de integración curricular presentado como requisito
para la obtención del título de Ingeniero en Tecnologías de la
Información

Autor:

Aguilera Jaramillo Mauricio Andres
Armijos Bustamante Daniel Alejandro

Tutor:

Lorena de los Ángeles Guachi, Ph.D.

Urcuquí, julio del 2020.

Urcuquí, 20 de marzo de 2020

SECRETARÍA GENERAL
(Vicerrectorado Académico/Cancillería)
ESCUELA DE CIENCIAS MATEMÁTICAS Y COMPUTACIONALES
CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN
ACTA DE DEFENSA No. UITEY-ITE-2020-00015-AD

A los 20 días del mes de marzo de 2020, a las 12:30 horas, de manera virtual mediante videoconferencia, y ante el Tribunal Calificador, integrado por los docentes:

Presidente Tribunal de Defensa	Dr. CUENCA LUCERO, FREDY ENRIQUE , Ph.D.
Miembro No Tutor	Dr. ARMAS ARGINIEGA, JULIO JOAQUIN , Ph.D.
Tutor	Dra. GUACHI GUACHI, LORENA DE LOS ANGELES , Ph.D.

El(la) señor(ita) estudiante **ARMIJOS BUSTAMANTE, DANIEL ALEJANDRO**, con cédula de identidad No. 1105801466, de la **ESCUELA DE CIENCIAS MATEMÁTICAS Y COMPUTACIONALES**, de la Carrera de **TECNOLOGÍAS DE LA INFORMACIÓN**, aprobada por el Consejo de Educación Superior (CES), mediante Resolución **RPC-SO-43-No.496-2014**, realiza a través de videoconferencia, la sustentación de su trabajo de titulación denominado: **ANOMALY DETECTION SYSTEM IN VIDEO SURVEILLANCE USING DEEP LEARNING TECHNIQUES.**, previa a la obtención del título de **INGENIERO/A EN TECNOLOGÍAS DE LA INFORMACIÓN**.

El citado trabajo de titulación, fue debidamente aprobado por el(los) docente(s):

Tutor	Dra. GUACHI GUACHI, LORENA DE LOS ANGELES , Ph.D.
--------------	---

Y recibió las observaciones de los otros miembros del Tribunal Calificador, las mismas que han sido incorporadas por el(la) estudiante.

Previamente cumplidos los requisitos legales y reglamentarios, el trabajo de titulación fue sustentado por el(la) estudiante y examinado por los miembros del Tribunal Calificador. Escuchada la sustentación del trabajo de titulación a través de videoconferencia, que integró la exposición de el(la) estudiante sobre el contenido de la misma y las preguntas formuladas por los miembros del Tribunal, se califica la sustentación del trabajo de titulación con las siguientes calificaciones:

Tipo	Docente	Calificación
Tutor	Dra. GUACHI GUACHI, LORENA DE LOS ANGELES , Ph.D.	10,0
Miembro Tribunal De Defensa	Dr. ARMAS ARGINIEGA, JULIO JOAQUIN , Ph.D.	9,5
Presidente Tribunal De Defensa	Dr. CUENCA LUCERO, FREDY ENRIQUE , Ph.D.	9,0

Lo que da un promedio de: **9.5 (Nueve punto Cinco)**, sobre 10 (diez), equivalente a: **APROBADO**

Para constancia de lo actuado, firman los miembros del Tribunal Calificador, el/la estudiante y el/la secretario ad-hoc.

ARMIJOS BUSTAMANTE, DANIEL ALEJANDRO
Estudiante

Dr. CUENCA LUCERO, FREDY ENRIQUE , Ph.D.
Presidente Tribunal de Defensa



Firmado electrónicamente por:
FREDY ENRIQUE
CUENCA LUCERO

Dra. GUACHI GUACHI, LORENA DE LOS ANGELES , Ph.D.
Tutor

Firmado Digitalmente por: **LORENA DE LOS ANGELES GUACHI GUACHI**
 Hora oficial Ecuador: 15/04/2020 14:45

Dr. ARMAS ARGINIEGA, JULIO JOAQUIN , Ph.D.
Miembro No Tutor



Firmado electrónicamente por:
JULIO JOAQUIN
ARMAS
ARCINIEGA

TORRES MONTALVÁN, TATIANA BEATRIZ
Secretario Ad-hoc

Urcuquí, 20 de marzo de 2020

SECRETARÍA GENERAL
(Vicerrectorado Académico/Cancillería)
ESCUELA DE CIENCIAS MATEMÁTICAS Y COMPUTACIONALES
CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN
ACTA DE DEFENSA No. UITEY-ITE-2020-00016-AD

A los 20 días del mes de marzo de 2020, a las 12:30 horas, de manera virtual mediante videoconferencia, y ante el Tribunal Calificador, integrado por los docentes:

Presidente Tribunal de Defensa	Dr. CUENCA LUCERO, FREDY ENRIQUE , Ph.D.
Miembro No Tutor	Dr. ARMAS ARGINIEGA, JULIO JOAQUIN , Ph.D.
Tutor	Dra. GUACHI GUACHI, LORENA DE LOS ANGELES , Ph.D.

El(la) señor(ita) estudiante **AGUILERA JARAMILLO, MAURICIO ANDRES**, con cédula de identidad No. 0704482223, de la **ESCUELA DE CIENCIAS MATEMÁTICAS Y COMPUTACIONALES**, de la Carrera de **TECNOLOGÍAS DE LA INFORMACIÓN**, aprobada por el Consejo de Educación Superior (CES), mediante Resolución **RPC-SO-43-No.496-2014**, realiza a través de videoconferencia, la sustentación de su trabajo de titulación denominado: **ANOMALY DETECTION SYSTEM IN VIDEO SURVEILLANCE USING DEEP LEARNING TECHNIQUES.**, previa a la obtención del título de **INGENIERO/A EN TECNOLOGÍAS DE LA INFORMACIÓN**.

El citado trabajo de titulación, fue debidamente aprobado por el(los) docente(s):

Tutor	Dra. GUACHI GUACHI, LORENA DE LOS ANGELES , Ph.D.
--------------	---

Y recibió las observaciones de los otros miembros del Tribunal Calificador, las mismas que han sido incorporadas por el(la) estudiante.

Previamente cumplidos los requisitos legales y reglamentarios, el trabajo de titulación fue sustentado por el(la) estudiante y examinado por los miembros del Tribunal Calificador. Escuchada la sustentación del trabajo de titulación a través de videoconferencia, que integró la exposición de el(la) estudiante sobre el contenido de la misma y las preguntas formuladas por los miembros del Tribunal, se califica la sustentación del trabajo de titulación con las siguientes calificaciones:

Tipo	Docente	Calificación
Miembro Tribunal De Defensa	Dr. ARMAS ARGINIEGA, JULIO JOAQUIN , Ph.D.	9,5
Presidente Tribunal De Defensa	Dr. CUENCA LUCERO, FREDY ENRIQUE , Ph.D.	9,0
Tutor	Dra. GUACHI GUACHI, LORENA DE LOS ANGELES , Ph.D.	10,0

Lo que da un promedio de: **9.5 (Nueve punto Cinco)**, sobre 10 (diez), equivalente a: **APROBADO**

Para constancia de lo actuado, firman los miembros del Tribunal Calificador, el/la estudiante y el/la secretario ad-hoc.

AGUILERA JARAMILLO, MAURICIO ANDRES
Estudiante



Dr. CUENCA LUCERO, FREDY ENRIQUE , Ph.D.
Presidente Tribunal de Defensa

Firmado Digitalmente por: LORENA DE LOS ANGELES GUACHI GUACHI
 Hora oficial Ecuador: 12/06/2020 16:40

Dra. GUACHI GUACHI, LORENA DE LOS ANGELES , Ph.D.

Tutor



Firmado electrónicamente por:
**JULIO JOAQUIN
ARMAS
ARCINIEGA**

Dr. ARMAS ARCINIEGA, JULIO JOAQUIN , Ph.D.
Miembro No Tutor

TORRES MONTALVÁN, TATIANA BEATRIZ
Secretario Ad-hoc

Autoría

Yo, **Mauricio Andres Aguilera Jaramillo**, con cédula de identidad 0704482223, declaro que las ideas, juicios, valoraciones, interpretaciones, consultas bibliográficas, definiciones y conceptualizaciones expuestas en el presente trabajo; así como, los procedimientos y herramientas utilizadas en la investigación, son de absoluta responsabilidad de el/la autor(a) del trabajo de integración curricular. Así mismo, me acojo a los reglamentos internos de la Universidad de Investigación de Tecnología Experimental Yachay.

Urcuquí, Julio 2020.

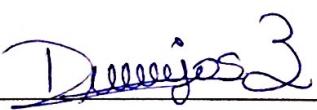


Mauricio Andres Aguilera Jaramillo
CI: 0704482223

Autoría

Yo, **Daniel Alejandro Armijos Bustamante**, con cédula de identidad 1105801466, declaro que las ideas, juicios, valoraciones, interpretaciones, consultas bibliográficas, definiciones y conceptualizaciones expuestas en el presente trabajo; así como, los procedimientos y herramientas utilizadas en la investigación, son de absoluta responsabilidad de el/la autor(a) del trabajo de integración curricular. Así mismo, me acojo a los reglamentos internos de la Universidad de Investigación de Tecnología Experimental Yachay.

Urcuquí, Julio 2020.



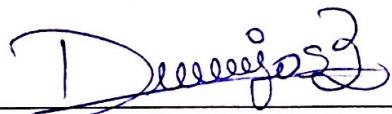
Daniel Alejandro Armijos Bustamante
CI: 1105801466

Autorización de publicación

Yo, **Daniel Alejandro Armijos Bustamante**, con cédula de identidad 1105801466, cedo a la Universidad de Tecnología Experimental Yachay, los derechos de publicación de la presente obra, sin que deba haber un reconocimiento económico por este concepto. Declaro además que el texto del presente trabajo de titulación no podrá ser cedido a ninguna empresa editorial para su publicación u otros fines, sin contar previamente con la autorización escrita de la Universidad.

Asimismo, autorizo a la Universidad que realice la digitalización y publicación de este trabajo de integración curricular en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Urcuquí, Julio 2020.



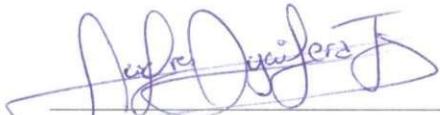
Daniel Alejandro Armijos Bustamante
CI: 1105801466

Autorización de publicación

Yo, Mauricio Andres Aguilera Jaramillo, con cédula de identidad 0704482223, cedo a la Universidad de Tecnología Experimental Yachay, los derechos de publicación de la presente obra, sin que deba haber un reconocimiento económico por este concepto. Declaro además que el texto del presente trabajo de titulación no podrá ser cedido a ninguna empresa editorial para su publicación u otros fines, sin contar previamente con la autorización escrita de la Universidad.

Asimismo, autorizo a la Universidad que realice la digitalización y publicación de este trabajo de integración curricular en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Urcuquí, Julio 2020.



Mauricio Andres Aguilera Jaramillo
CI: 0704482223

Dedication

This thesis is dedicated to my parents Leslie and Mauricio who with their love, patience and effort have allowed me to fulfill one more dream, thanks for instilling in me the example of effort and courage, not to fear adversity because you are always with me. To my friends, for supporting me when I need them most, for extending their hand in difficult times and for the love given every day, really thank you, I will always carry them in my heart. To all those who did not believe in me, to those who expected my failure at every step I took, to all those who bet that I give up halfway, to all who assumed that I would not succeed. Finally I want to dedicate this thesis to Naomi, the source of my motivation, who as nobody managed to bring out the worst and the best of me. Thank you for being the one who formed a large part of my being. Thank you for being that strong woman who is not afraid of anything. - Mauricio Aguilera Jaramillo

First of all, I dedicate this project to my parents, Lorena Bustamante and Carlos Armijos. With their unconditional love and support they knew how to guide me from the beginning until I completed this great challenge. I dedicate it also to my brother Manuel, and sister Sofía, and specially to my nieces Isabela Valentina, all of them have played a crucial role to define the kind of person I am today. Also, I want to dedicate this Thesis to my aunt Rebeca Bustamante and their family, I have always been treated as a member of them, and I will always be grateful for all the affection they have given me. I dedicate this project to all of my friends, to the new ones I had the opportunity to meet in Yachay, the ones who live in Loja, and to the glorious apartment H 3-2. It is difficult to name all of the amazing people that life has presented to me, but I'm definitely sure they are a reason to keep me motivated throughout my existence here. Finally, I dedicate this project to all the great professors at Yachay Tech from whom I had the opportunity to learn. Without their commitment and that of the students, nothing about this dream of what Yachay is would have been possible. - Daniel Armijos Bustamante

Resumen

En el campo del análisis de videovigilancia, la detección de anomalías es una tarea esencial para garantizar la protección pública durante la detección de tramos que tienden a contener eventos inusuales como robos, asaltos, peleas, entre otros. Recientemente, Las Redes Neuronales Convolucionales (CNN en inglés), una categoría basada en técnicas de aprendizaje profundo (Deep Learning), ha mostrado un gran progreso en las tareas de visión por computadora, con aplicaciones relacionadas con la clasificación / reconocimiento de imágenes y videos, y más específicamente para tareas de detección de anomalías. Sin embargo, no han sido capaces de manejar la precisión en escenarios reales debido a la presencia de ruido, contexto / situaciones específicas, variabilidad de cómo se definen los diferentes eventos, datos limitados para fines de entrenamiento, altos recursos computacionales necesarios para responder en tiempo real, entre otros. Este trabajo explora dos arquitecturas de CNN de clasificación de video sobresalientes para analizar su estructura, su capacidad para ejecutarse en escenarios en tiempo real y la capacidad de clasificar adecuadamente los cuadros de video en eventos normales y anormales. Tales anomalías incluyen abuso, arresto y asalto. También se presenta una nueva arquitectura de CNN, llamada Frankensnet, centrada en la clasificación de anomalías del cuadro de video. Frankensnet tiene como objetivo tomar la característica estructural de las CNN exploradas, teniendo en cuenta la precisión lograda, la capacidad de detección en tiempo real y el tiempo de entrenamiento. Los experimentos se realizan en referencia al conjunto de datos UCF-Crime. Como resultados preliminares, este documento proporciona un punto de vista para seleccionar arquitecturas CNN para la identificación de anomalías, considerando la precisión, así como el tiempo de entrenamiento y ejecución. Además, FrankensNet demostró ser adecuado para escenarios que requieren una alta precisión. Sin embargo, entrenar la arquitectura toma aproximadamente el doble de tiempo que las arquitecturas exploradas. Finalmente, se proporciona un programa de escritorio para probar el rendimiento de cada arquitectura CNN en escenarios en tiempo real con tareas de detección de anomalías.

Palabras clave: Clasificación de video, visión por computadora, detección de anomalías, redes neuronales convolucionales.

Abstract

In the field of video surveillance analysis, anomaly detection is an essential task to ensure public protection throughout detecting frames which tend to contain unusual events such as robberies, assaults, fights, among others. Recently, Convolutional Neural Network (CNN), a category of deep learning techniques, have shown great progress in computer vision tasks, with applications related to both image and video classification / recognition, and more specifically for anomaly detection tasks. Nonetheless, they fail to handle accuracy in real-scenarios due to the presence of noise, specific context/situations, variability of how different events are defined, limited data for training purposes, high computational resources required to respond on real-time, among others. This work explores two outstanding video classification CNNs architectures to analyze their structure, their capability to run in real-time scenarios, and the ability to properly classify video frames into normal and abnormal events. Such abnormal anomalies include abuse, arrest, and assault. It also introduces a new CNN architecture, called Frankensnet, focused on anomaly classification from video frame. Frankensnet aims to take the structural characteristic of the explored CNNs, taking into account the accuracy achieved, capability of detection in real time and training time. Experiments are performed referring to UCF-Crime dataset. As the preliminary results, this paper provides a point-of-view to select CNN architectures for anomaly identification, considering accuracy as well as training and execution time. Moreover, FrankensNet demonstrated to be suitable for scenarios requiring a high accuracy. Nevertheless, training the architecture takes approximately twice as much time as the explored architectures. Finally, a desktop program is also provided to test the performance of each CNN architecture on real-time scenarios with tasks of anomaly detection.

Keywords: Video Classification, Computer Vision, Anomaly Detection, Convolutional Neural Networks.

Contents

1	Introduction	6
1.1	Problem Statement	8
1.2	Justification	8
1.3	Contribution	9
1.4	Thesis overview	9
1.5	Objectives	10
1.5.1	General Objective	10
1.5.2	Specific Objectives	10
2	Theoretical Framework of Video Classification	11
2.1	Video Classification Challenges	11
2.2	Video Classification Applications	12
2.3	Deep Learning Approaches for Video Classification: State-of-the-art	14
2.3.1	Autoencoder	14
2.3.2	Recurrent Neural Networks (RNNs)	15
2.3.3	Convolutional Neural Networks (CNNs)	16
2.4	Theoretical Foundation of CNNs	17
2.4.1	Hyperparameters	19
2.4.2	Major concerns	20
2.5	CNNs Models for Video Classification	20
2.5.1	Performance Criteria	20
2.5.2	CNN Models	22
2.5.3	Inception V-3 [1]	25
2.5.4	Residual Network (ResNet) [2]	27
2.5.5	Convolutional 3D Network (C3D) [3]	28
2.5.6	DenseNet [4]	28
3	Methodology for Designing an Anomaly Detection System using CNNs	30
3.1	Analysis	31
3.1.1	Performance Criteria to Build the New Model for Video Classification	31
3.1.2	Hardware and Software Tools	31
3.1.2.1	Software Resources	31
3.1.2.2	Hardware Resources:	33
3.2	Design and Implementation	33

3.2.1	New Approach: FrankensNet	33
3.2.1.1	Definition of FrankensNet Architecture	33
3.2.1.2	Advantages and Disadvantages	35
3.2.2	Implementation of CNNs Models	35
3.3	Data Preparation	36
3.3.1	Dataset	36
3.3.2	Pre-processing	37
3.4	Parameter Settings	40
3.5	Training	40
3.6	Validation	41
3.7	Implementation of a Graphical UI for an Anomaly Detection Prototype	41
4	Experimental Setup	42
4.1	Training Implementation	42
4.2	Validation Implementation	42
4.3	Metrics	43
4.4	Data Preparation:	44
4.4.1	Data Augmentation:	44
4.4.1.1	Rotate:	44
4.4.1.2	Crop:	45
4.4.2	Data Pre-processing:	45
4.4.2.1	Contrast Limited Adaptive Histogram Equalization	45
4.4.2.2	Median Filter & Unsharp Masking	45
4.5	Experiments	45
4.5.1	Experiment 1: Inception V-3 Settings	45
4.5.2	Experiment 2: ResNet-50 Settings	46
4.5.3	Experiment 3: FrankensNet Settings	46
4.5.4	Experiment 4: Training Time / Execution Time	47
4.5.5	Experiment 5: FP/FN rate, TP/TN rate	47
5	Results	48
5.1	Experiments	48
5.1.1	Experiment 1: Inception-V3	48
5.1.2	Experiment 2: ResNet-50	50
5.1.3	Experiment 3: FrankensNet	52
5.1.4	Experiment 4: Training Time / Execution Time	55
5.1.5	Experiment 5: FP/FN rate, TP/TN rate	56
5.2	Additional Results	57
5.2.1	Overall results	57
5.2.2	A Graphical UI for an Anomaly Detection Prototype	59
6	Conclusions and Future work	62
6.1	Conclusions	62
6.2	Future Work	63

6.3	Glossary	64
References		67
Appendices		71
A	DataSets	73
A.1	Common Datasets Used for Image & Video Classification	73
B	Algorithm Codes	74
B.1	Thesis Web Page	74
B.2	FrankensNet Implementation Code	74
B.3	Training Code	78
B.4	Rotate Operation Code	80
B.5	Crop Operation Code	80
B.6	Contrast Limited Adaptive Histogram Equalization Code	80
B.7	Median Filter & Unsharp Masking	81

List of Figures

2.1	CNN General Architecture	17
2.2	Convolutional and Pooling Layers	18
2.3	Architectures 1	23
2.4	Architectures 2	24
2.5	Inception Module A [5]	25
2.6	Inception Module B [1]	26
2.7	Inception Module C [1]	26
2.8	Auxiliary & Main Classifiers [1]	27
2.9	Residual Block. [2]	27
2.10	DenseNet Architecture [4]	29
3.1	General Block Diagram	30
3.2	FrankensNet Architecture	34
3.3	UCF-Crime Dataset Categories [6]	36
3.4	Data Augmentation operation on UCF-Crime dataset [6]	38
3.5	CLAHE example on UCF-Crime dataset [6]	39
3.6	Medial Filter example on UCF-Crime dataset [6]	39
3.7	Unsharp Masking Example on UCF-Crime dataset [6]	40
5.1	Minimum Accuracy. Parameters: LR=0.001 ; BS=100	49
5.2	Maximum Accuracy. Parameters: LR=0.00001 ; BS=100	50
5.3	Minimum Accuracy. Parameters: LR=0.000001 ; BS=100	51
5.4	Maximum Accuracy. Parameters: LR=0.00001 ; BS=100	52
5.5	Minimum Accuracy. Parameters: LR=0.00001 ; BS=100	53
5.6	Maximum Accuracy. Parameters: LR=0.001 ; BS=200	54
5.7	Results of learning rate experiments 1, 2, 3	54
5.8	Results of batch size experiments 1, 2, 3	55
5.9	Model's Accuracy: Epoch 0	57
5.10	Model's Accuracy: Epoch 1	58
5.11	Model's Loss: Epoch 0	58
5.12	Model's Loss: Epoch 1	59
5.13	ROC curve	60
5.14	Anomalous Detection options	60
5.15	Anomalous Detection in real time	61

List of Tables

2.1	Results on the number of layers and computational complexity of Network Architectures	22
5.1	Experiments performed on Inception-V3 with three different Learning Rates	48
5.2	Experiments performed on Inception-V3 with three different Batch Sizes .	48
5.3	Experiments performed on ResNet-50 with three different Learning Rates .	50
5.4	Experiments performed on ResNet-50 with three different Batch Sizes . .	51
5.5	Experiments performed on FrankensNet with three different Learning Rates	52
5.6	Experiments performed on FrankensNet with three different Batch Sizes . .	53
5.7	Model's Training Time	55
5.8	Models predictions	56
5.9	Classification Rates	56

Chapter 1

Introduction

Video classification is an actual computer vision problem that was presented with the purpose of being able to automatize this type of classification tasks. Given the fact that the problem is considerably recent, there still exist quite a lot of gaps left to be discovered. Nevertheless, its applications are becoming largely varied, starting from only detecting the type of sports or daily activity that is happening on the scene, to actual health and security problems, just to name a few [7].

In the last years, different approaches have been introduced for image/video frame classification, they range from background modeling ones (model the background to detect characteristic objects) such as background subtraction, to those that analyse entire image/video frame applying specialized detectors for car, pedestrian, etc. such as Artificial neural network, Deep learning, Machine Learning classifiers, among others. However, it is a challenging task due to general image classification factors related to the illumination changes and shadows, low-resolution images, noise presence, low quality images, real-time performance requirements and even the same image data management to applications in real-world (such as video surveillance systems focused on understanding image/video sequences to assign them semantically meaningful anomaly categories) [8].

When research began to be conducted mainly focused in video classification, different solutions for the same target appeared. For example, the introduction of 3D convolutional neural networks applied to Human Action Recognition [9]. This kind of CNN was used to extract features from each action and to later predict them. Such actions are divided into 3 classes: cell to ear, object put, and pointing. Another approach is presented to work with action sports classification tasks, based on three main aspects: feature extraction, dictionary learning and lastly classification [10]. Finally, there are also big companies that are interested in this new field, Google introduced a different approach, based once again on convolutional [7] neural networks to work with YouTube-8M dataset, containing 8 million videos with 4800 different classes, like sports, activities, animals, foods, products, tourist attractions, games, and many more [7].

Additionally, when referring to video classification, there is a subgroup based on

anomaly detection tasks. The definition of an anomaly depends on the context of interest. Nevertheless, anomaly is related with the existence of events that are considered as irregular, sudden or unanticipated. A combination of deep learning techniques based on machine learning and neural networks (Support Vector Machines, Autoencoders, LSTM) [11] were introduced to detect anomalies. In this case, anomalies are considered as anything different than pedestrians walking on the sidewalk (bikes, skateboards, etc). Moreover, a group of researchers proposed to learn anomalies by exploiting both normal and anomalous videos. In their approach, they consider normal and anomalous videos as bags and video segments as instances in multiple instance learning (MIL), and automatically learn a deep anomaly ranking model that predicts high anomaly scores for anomalous video segments [8]. For them, an anomaly consists of crime activities, such as fighting, robbery, shooting, etc.

The increase in computer performance and modern high-speed technologies have provided a wide range of effective and efficient solutions for security systems, where video sequences is an affordable method for data gathering. In many countries have been implemented different security services in order to face delinquency, sometimes by the government itself. Such services commonly merge diverse emergency services provided by fire departments, armed forces, national police, etc. It is an interesting fact because even though surveillance cameras have taken an important role as a part of such services to ensure the safety of citizens, crimes are usually not adequately reduced [12]. Nevertheless, these cameras have a capability of working on a 24/7 basis and are also able to capture all the information contained in the scene. In this sense, due to the growth of image/video data collected from surveillance cameras, automated video analysis has become necessary in order to detect automatically abnormal events [8].

This work proposes FrankensNet, a network capable of detecting anomaly activities from surveillance camera videos. This solution is compared against two existing architectures: ResNet-50 [2] and Inception V-3 [1]. Each model is trained separately with the UCF-crime dataset, [6], containing a total of 128 hours of video recordings and 1900 real-world surveillance videos of 13 different anomalous events and normal activities captured by surveillance cameras [8]. Due to limited computational resources, the abnormal class is composed of three main anomalies obtained from video surveillance which corresponds to: acts of abuse, arrests, and arson.

Experimental results, based mainly on the models accuracy, training time, and execution time show that FrankensNet and ResNet-50 are the best options when accuracy is of primary importance. Those networks were able to reach an accuracy superior to 90%, while InceptionV-3 was not able to surpass this value. Nevertheless, InceptionV-3 stands out as the winner on training time, being able to perform such process on the shortest time. In addition, a desktop application system is also proposed, where each trained model can be tested to perform classification tasks in real time. Models and program implementation were all made on Python programming language.

This chapter provides an introduction to the current work, the problem statement, justification, the outline of the contribution and a brief overview of the organization of this work.

1.1 Problem Statement

One of the major challenges in Artificial Intelligence (AI) is the capability to interpret a given sequence of images or videos to adequately identify and recognize patterns as well as classify scenes from pixel analysis. In the last decade, automatic scene understanding has been widely researched to provide methods that allow to immediately recognize and classify the meaning of the scene and the global structure similar to how human scene understanding is done. Particularly, in video surveillance the ability to understand a scene relies on finding the reliable description and classification on frames, which define an anomaly scene (thefts, robberies, acts of vandalism, etc) [8].

Although several works have been introduced to image and video classification, determining the most appropriate deep learning model for anomaly detection on real scenarios is still a challenge in vision system. It is attributed to factors as: accuracy, execution in real-time, determination of the most optimal method, reduction of computational complexity, adaptation the algorithm to specific application, etc. Moreover, depending on the application, some of them may require real-time execution and resource optimization. Particularly, the difficulties in video classification are also related to their high dimensional structure and the non-local temporal variations across frames [3].

1.2 Justification

The aim of video anomaly detection is to recognize unusual events, actions and/or operations that are harmful to personal or public security. The automated and precise identification of anomalies may prevent potential accidents or crowd disasters, such as thefts, robberies, acts of vandalism, etc. Many existing methods support the statement that there is no current method that efficiently works for all application scenarios. Even with human assistance, the anomaly detection may fail in real scenarios with unknown anomalies, dynamic backgrounds, illumination changes, and in almost all cases, due to huge amount of data to be analyzed on real-time

Recently, deep learning techniques have become increasingly popular and have been applied to a variety of tasks such as leaf disease classification and detection, cancer diagnosis, anomaly detection, among others. They have demonstrated outperform traditional methods. However, it is difficult to determine the adequate method for specific application. Therefore, this work presents an evaluation of some deep learning models to provide a point-of-view to select the most appropriate model for anomaly classification, regarding to accuracy and precision in terms of false or true detection in real time, computational

time, among others. Furthermore, the models evaluation allows to identify the best deep learning characteristic that can be taken to create a more robust model for anomaly detection. In this sense, it provides the chance to develop deeper studies on social security or even commercial fields. In addition, data used to evaluate the performance of each model is provided for free by the University of North Carolina at Charlotte. This data is used solely for the purpose of research required by the project. It does not have an income acquirement whatsoever.

1.3 Contribution

The contributions of this project are:

- Develop an appropriate methodology to evaluate a set of deep learning models to provide a point-of-view on how deep learning architectures work for anomaly detection, in terms of precision, accuracy, training time, and execution time.
- Introduce a new deep neural network approach, which will be able to learn and classify scenes with anomalies from surveillance cameras.
- Compare the performance between each of the models in order to determine the optimal candidate for anomaly detection tasks.
- Develop a graphic user interface, which will display the image capture information in real time, with its classification information.

1.4 Thesis overview

This work is divided into 7 main Chapters which are named as follows: Introduction, Theoretical Framework of Video Classification, Methodology for designing an Anomaly detection System using CNNs, Design and Implementation, Experimental Setup, Results, and Conclusions.

Chapter 1 states the problem statement of the project, along with its justification, contribution, and general and specifics objectives.

Chapter 2 starts with a brief analysis made on video classification, what are its challenges and applications on given works or researches. In addition, a set of previous works given on deep learning approaches are presented. The last sections in this chapter will be focused on the theoretical foundations of Convolutional Neural Networks, and a couple of architectures used for video classification.

Chapter 3 contains the methodology used in this research, from the models which have been chosen to implement and evaluate on future experiments.

Chapter 3.2 starts with a new model definition named FrankensNet, and how it was developed. Also, the data preparation, referred to the dataset, parameter settings, etc. are mentioned briefly in this chapter.

Chapter 4 contains the configurations that each one of the experiments will have. In addition, the metrics established to test an architecture performance are also presented.

Chapter 5 contains the results and discussion performed on each one of the experiments defined.

In Chapter 6, the conclusions obtained from this work are presented. Also, future works that can improve the proposed methodology and help to establish open issues are specified.

1.5 Objectives

1.5.1 General Objective

To develop a method capable of classifying anomalous scenes with high accuracy in video surveillance by using deep learning.

1.5.2 Specific Objectives

- To implement and explore the two most outstanding deep learning models for the classification of images and videos, in terms of precision, accuracy, training and execution time.
- To design and implement a new deep learning architecture for video classification taking the best characteristic of explored methods.
- To determine the performance of the proposed approach with respect to the two most outstanding deep learning architectures in terms of precision, accuracy, training and execution time.
- To implement a computational software system for video classification, using the trained model of the proposed deep learning architecture.

Chapter 2

Theoretical Framework of Video Classification

In this chapter, video classification fundamentals is presented. Firstly, common concepts related with image and video classification are stated. Next, an overview of its applications, challenges, major concerns and accuracy metrics are also described. Then, several video classification approaches based on different deep learning techniques are presented. Finally, convolutional neural networks fundamentals will be depicted. After that, the hardware and software used to implement these different deep learning techniques will be presented.

2.1 Video Classification Challenges

In video image classification, fixed camera, constant illumination, a single environment, etc. are required conditions to obtain high accuracy as possible. Nevertheless, it is not possible in real-life. Therefore, a video classification approach should face the following challenges under real-life environments:

- **Noise:** It might be introduced in the image due to a poor quality image source, during transmission from the source to the further processing, or caused by environmental factors such as wind, fog, sun-rays, and clouds.
- **Different Environments:** Given the fact that videos are obtained from video surveillance cameras, there is going to be many changes in the time of the day, on indoor and outdoor environments, sometimes with an static or moving camera, etc.
- **Camera Resolution:** Since not all video surveillance cameras have the same resolution at the moment of recording, different resolutions might appear on the videos gathered.
- **Network Architecture Design:** Considering that for videos, it is necessary to capture spatiotemporal information, the options to design an architecture with such capabilities become expensive to evaluate.

- **Number of Classes:** Due to the fact that there exist different activities or actions which can be considered as an specific class according to the needs.
- **Capturing the Context of the Scene:** As videos usually involve capturing spatial temporal features across frames. On some cases, information may vary in terms of camera movement, light changes, etc.
- **Data Management:** Depending on the type of videos which are being sought, they often become difficult to collect, annotate and store.

In the next sections some video classification approaches are presented. They have provided good accuracy for exiting applications in security sector. Nevertheless, its employments go beyond security only, this due to the fact that they also have presented good results on other topics, such as health, sports, and even daily activities, etc.

2.2 Video Classification Applications

Video classification is not limited to security applications. They are often used in the following computer vision applications.

- **Action Recognition:** Where human actions, such as: walking, playing the guitar, playing sports, etc. are identified in each frame of a certain scene [13]. It refers to the identification of different actions happening in a given scene.

The Computer Science Department, located at Stanford University implemented a CNN model that is evaluated on large scale video classification using the Sports 1-M Youtube dataset containing 487 classes, such as Aquatic Sports, Team Sports, Winter Sports, etc. As it is one of the first works carried out in this field, its accuracy, even surpassing older techniques, is not so formidable by reaching a value of 63.9% [13].

Another research is presented by Ji Shuiwang et al. on the IEEE. Here, they introduced a novel 3D CNN model for action recognition using the TRECVID 2008 dataset. Such dataset consists of 49 hour videos collected from London Gatwick Airport with three different classes [9], such as talking on the phone, pointing with the finger, and dropping objects. Its accuracy in this tasks is considerably higher, achieving an average of 78.24%.

Finally, Luvizon et al. presents a new framework designed for human action recognition from skeleton sequences. This due to recent technologies that provide the skeletal representation of human body extracted in real time [14]. They performed the evaluation on three different datasets: MSR-Action3D, the UTKinectAction3D, and the Florence 3D Actions dataset, with classees such as throwing an object, waving hands, jumping, etc. Its accuracy exceeded 90%, reaching an average of 91.3%

among all datasets.

- **Object Detection:** It consists of keeping track of moving or static objects appearing on the scene at an specific frame, one by one.

A contribution made by the IEEE organization presents an approach based on a new spatio-temporal test and dichromatic reflection model [15], this allows the model to react accordingly for both the sun and the sky illuminations. This research did not use a public dataset. Instead, they collected a set of videos consisting of two classes: Vehicles, and people. The main feature of such model is its robustness when working with illumination changes at any time of the day.

A group of researchers from John Marshall's College and the University of Minnesota implemented a CNN along with a Single Shot Detector Architecture for a real time analysis of blood borne pathogens in Microscopy [16]. For this task, they used a Dark field microscopy dataset, containing only two classes: Malaria and Syphilis. It proved to be an effective technique of detection in real time but without an acceptable accuracy, achieving a maximum value of 43% on average.

Cheng et al. proposed a novel approach based on a CNN model for advancing the performance of object detection. This is achieved by introducing and learning a new rotation invariant layer on the basis of the existing CNN architectures [17]. The architecture is trained and tested on the VHR optical remote sensing image dataset, containing ten object categories, such as airplane, ship, storage tank, baseball diamond, tennis court, etc. The accuracy achieved by this approach in average is of 72.63%.

- **Anomaly Detection:** Its definition depends on the context but it can be referred to crime detection or unusual activities [8],[18],[19],[11].

Sultani & Mubarak propose to learn anomalies by exploiting both normal and anomalous videos. In their approach, they consider normal and anomalous videos as bags and video segments as instances in multiple instance learning (MIL), and automatically learn a deep anomaly ranking model that predicts high anomaly scores for anomalous video segments [8]. In this research, the UCF-crime dataset was used. It consists of 1900 real world surveillance videos with 13 different anomalies, such as fighting, burglary, road accidents, etc. Although its capacity to work in real time is good, the accuracy achieved did not met the expectations, reaching a 75%.

Ribeiro et al. presented a method based on Convolutional Autoencoder (CAE) [18], both the encoder and decoder part is conformed by three convolutional layers, and

two pooling layers to detect anomalies in three different scenarios obtained from the 3 datasets, since it captures the 2D structure in image sequences during the learning process. They used a CAE in the anomaly detection context, by applying the reconstruction error of each frame as an anomaly score. For their research, three different datasets were used based on pedestrian walkaway surveillance cameras: UCSD pedestrian dataset (including two subsets, Ped1 and Ped2) , and Avenue dataset. On average, the accuracy obtained by this architecture reached a total of 73.20%.

Khaleghi & Shahram propose a new method based on deep learning techniques for anomaly detection in video surveillance cameras with the use of the UCSD dataset for experimentation. This dataset is related to the pedestrian walkaway surveillance camera, any objects other than people are identified as anomaly, such as the appearance of a bicycle or a car in the scene [11]. The accuracy obtained by this architecture showed a great improvement, reaching a total of 86%.

Sabokrou et al. introduces a paper where they present an efficient method for detection and localization of anomalies in crowded scenes using fully convolutional neural networks with the use of UCSD Ped2 dataset and Subway Benchmarks dataset [19]. The UCSD dataset contains a set of videos obtained from a given surveillance camera showing pedestrians walking. Here, anything different from a pedestrian will be considered as an anomaly (The presence of different objects, such as bicycles, car, skateboards, etc). On the other side, the Subway dataset contains a set of videos obtained from video surveillance cameras placed on subway stations. The accuracy obtained by this model in detection tasks is the greatest of them all, achieving a final value of 90.20%

2.3 Deep Learning Approaches for Video Classification: State-of-the-art

2.3.1 Autoencoder

Autoencoders are simple learning circuits which their objective consist on transform its inputs into its outputs with the least possible amount of distortion [20]. In general, this type of network is composed of two main parts:

- **Encoder:** It is referred to section of the network which compresses the input into a latent space representation. This latent space contains a compressed representation of the data.
- **Decoder:** It consists of the second section of the network, after the latent space representation. Its objective is to reconstruct the input from the previous representation.

The autoencoder can then be described with the equation which appears on 2.1

$$g(f(x)) = \tilde{x} \quad (2.1)$$

Where $f(x)$ consists on the function given by the encoder, while $g(f(x))$ becomes the function of the decoder, and \tilde{x} is expected to be as close as the original input x .

In the simplest case, assuming that the autoencoder contains only one hidden layer, as it was mentioned earlier, the encoder stage takes the input x and maps it to z (latent space representation), with the following equation:

$$z = \sigma(Wx + b) \quad (2.2)$$

Here, σ corresponds to an activation function. W is a weight matrix and b is a bias vector. Both of these variables are initialized at random, and then updated continuously during training through backpropagation. Then, the decoder stage maps z to a reconstruction \tilde{x} which is of the same shape as x in the following way:

$$\tilde{x} = \sigma'(W'z + b') \quad (2.3)$$

Autoencoders have been used widely in different video classification tasks. For example in anomaly detection, where autoencoders are used as a form to identify normal cubic patches [21] on the UCSD dataset [22] by performing the same task as in the case of the previous two techniques, or a convolutional autoencoder to perform feature extraction on videos [18].

2.3.2 Recurrent Neural Networks (RNNs)

It is a type of artificial neural network where the output obtained from last step is being fed as input to the present step. RNNs differ from feedforward architectures in a way that they do not only operate on an input space but also on an internal state space (a mark of the information that already has been handled by the network) [23].

RNNs can be derived from differential equations. First, let $\vec{s}(t)$ correspond to the value of the d-dimensional state signal vector, and lets consider the following differential equation, which describes the evolution of the state signal as a function of time, t:

$$\frac{d\vec{s}(t)}{dt} = \vec{f}(t) + \vec{\phi} \quad (2.4)$$

On equation 2.4, $\vec{f}(t)$ represents a d-dimensional vector-valued function of time, and $\vec{\phi}$ is a constant d-dimensional vector.

Now, $\vec{f}(t)$ can be represented as the following canonical form:

$$\vec{f}(t) = \vec{h}(\vec{s}(t), \vec{x}(t)) \quad (2.5)$$

Here, $\vec{x}(t)$ is the d-dimensional input signal vector, and $\vec{h}(\vec{s}(t), \vec{x}(t))$ corresponds to a vector-valued function of vector-valued arguments [24]. In the end, the final equation can be defined as:

$$\frac{d\vec{s}(t)}{dt} = \vec{h}(\vec{s}(t), \vec{x}(t)) + \vec{\phi} \quad (2.6)$$

One of the main characteristics of RNNs is that they are capable of remembering the past and its decisions are being determined by what it has learnt from such past. In other terms, it means that the outputs are influenced not just by weights being applied on inputs like a common neural network, but also by a hidden state vector that represents the context based on prior inputs/outputs.

Such network capabilities have become of great help when working with classification tasks. As it is the case for detecting anomalies, where the main architecture is composed of a convolutional neural network and an LSTM and the end of it to perform the classification part [8] with the [6]. In this case, anomalies are no longer considered as something different than people walking on the sidewalk. Instead, an anomaly is now considered as a crime, for example: arrest, assault, arson, abuse, etc. And a normal situation occurs when there is not happening a danger situation on the scene.

2.3.3 Convolutional Neural Networks (CNNs)

CNNs are analogous to traditional artificial neural networks in a way that they are comprised of neurons that self-optimize through learning. Nonetheless, the main difference between them is that CNNs are mostly used in the field of pattern recognition within images [25].

Traditional CNNs are comprised primarily of three types of layers: convolutional, pooling and fully-connected layers. Each layer performs a specific function, and there exist also other factors that are considered of great importance, such as the hyperparameters, or the stride and padding used at a specific layer to transfer information [26].

Subsequently with the success of CNNs in some real world application such as medical diagnosis [27], traffic sign [28], forest fire [29], and face detection [30], just to name a few. It is also providing exciting solutions with good accuracy for image and video classification to, as it is in the case of the INLSVR challenge [31], action recognition [32], or even sports recognition [33]. However, they are still been widely investigated to handle challenges mainly related to real time performance requirements to applications in real scenarios such as anomaly identification / detection. Thus, with the aim to explore how a CNN works, all of CNN terms are going to be explained more deeply on the following section.

2.4 Theoretical Foundation of CNNs

A CNN is a type of artificial neural network that uses convolutional layers to filter inputs for obtaining useful information for the network such as edges [34], shapes [35], among others. This type of multi-layer network is widely used to recognize visual patterns such as characters, symbols, figures, etc. from pixel images [36]. A CNN commonly is compounded of many kind of repeating layers and activation functions such as convolutional layer, ReLu, Pooling and Fully Connected layer, as is shown in 2.1. The most common layers and functions will be described in the following:

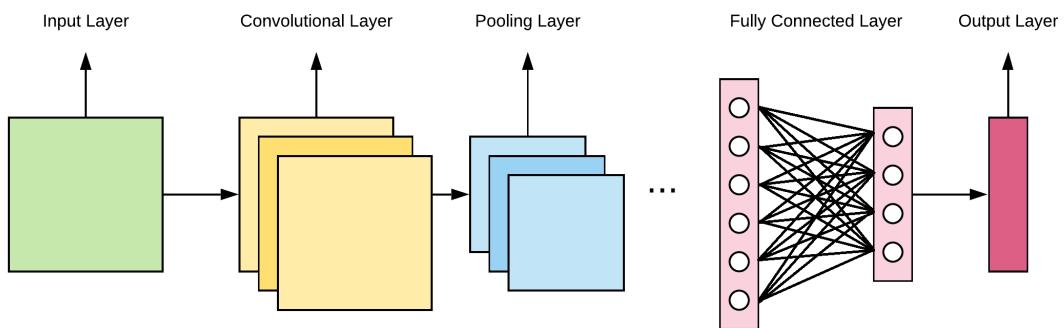


Figure 2.1: CNN General Architecture

Input Layer: This layer contains image data represented by a three dimensional matrix. Image data needs to convert it into a single column of dimension width x height x number of channels.

Convolutional Layer: This layer uses convolutional filters often called kernels, with a defined size, that will go over the entire input data, and perform a convolution operation, which is defined by equation 2.7:

$$(f_k)_{ij} = (W_k * x)_{ij} + b_k \quad (2.7)$$

The filter slides over the input matrix with a stride S. This process is done to learn and detect patterns from the previous layers. x represents the input data, w_k and b_k are the weight and the bias, respectively. The result is a feature map.

Pooling Layer: Also referred to as a down-sampling layer, it is used to reduce the spatial dimensions, but not depth on a CNN. Among the main features it appears that by having less spatial information the architecture gains computational performance. In addition, less spatial information also means less parameters, so less chance to over-fit. In general, the pooling layer:

- Receives a volume of size $P_1 x A_1 x I_1$

- Two hyperparameters are required: their spatial extent K and the stride J
- Returns a volume of size $P_2 \times A_2 \times I_2$ where:

$$P_2 = (P_1 - K)/J + 1 \quad (2.8)$$

$$A_2 = (A_1 - K)/J + 1 \quad (2.9)$$

$$I_2 = I_1 \quad (2.10)$$

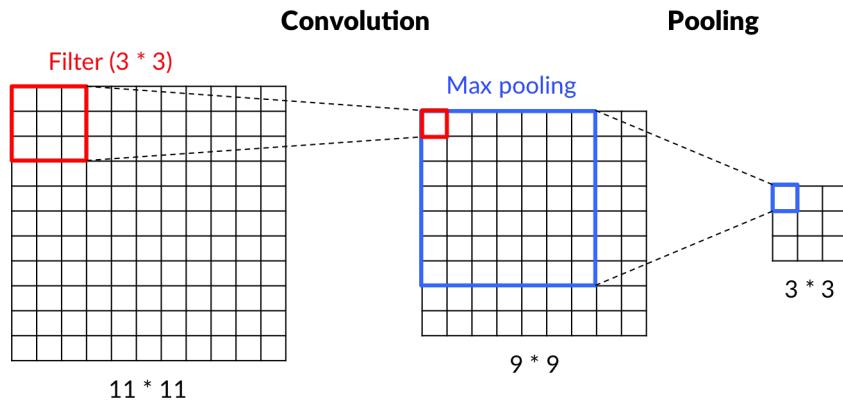


Figure 2.2: Convolutional and Pooling Layers

In Fig. 2.2. one can observe what happens on both convolutional and pooling layers. First, the filter of size 3×3 is being convolved across the width and height of the input volume which is usually an image, and computes the equation 2.7 between the entries of the filter and the input. Then, the pooling section appears to down sample the information, and it does it through the equations 2.8, 2.9 and 2.10. It is important to denote how the matrix size is being reduced on each layer. This depends on the size of the kernel and the stride established on each step.

Activation Function: It normally goes after the pooling layer or the fully connected layer. Its objective is to apply a non-linear transfer function to encode patterns through transformations. The most common activation functions are sigmoid, a logistic function with continuous values which has two asymptotes and an easy to obtain derivative [37], hyperbolic tangent, easily defined as the ratio between the hyperbolic sine and the cosine functions, commonly used for recognition tasks [38], and Rectified Linear Units (ReLU), a linear function that its purpose is to transmit the input as the output if and only if the value entering is positive, else it will have a zero value [39]. ReLU is given by equation 2.11.

$$f(z) = \max(0, x) \quad (2.11)$$

Fully-Connected Layer: A fully connected layer is a function of R^m to R^n . In this type of layer, each neuron is connected to each of the previous layer, and each connection has its own weight. The input to the FC layer corresponds to the output of the last pooling or convolutional layer. Each one of the FC layers perform the calculations based on a specific activation function.

Its importance relies on the fact that FC layers very useful at the moment of learning the different features presented by either the pooling or convolutional layers. Also, it plays a crucial role at the classification stage.

Output Layer: It is the final layer in the network, also being a fully connected one. This layers contains the activation function which is used to obtain the probabilities of the given input belonging to a particular class with its corresponding label, and the output neurons.

2.4.1 Hyperparameters

For training purposes, the number and diversity of hyperparameters such as batch size, learning rate, number of epoch and number of layers are quite specific to each model. Nevertheless, there exist a few of them that should always be taking into account with the aim to improve CNN performance.

- **Learning Rate:** This hyperparameter is in charge to quantify the learning progress of a model in such a way that will be used to optimize its capacity. It is also considered as the most important one due to the fact it affects the training time and the accuracy of a neural network. For instance, a learning rate which is too large or too small can take many training time if is not in an adequate range. The best accuracy is obtained by set a learning rate of 0.05 [40].
- **Batch Size:** This hyperparameter defines the number of training samples to work before updating the model parameters. Its size must be more than or equal to one and less than or equal to the training dataset size. It has an effect on the resource requirements of the training process, speed and number of iterations in a not so trivial way. An evaluation was carried out about the performance of CNNs, specifically LeNet and a generic CNN with seven convolutional layers, by different batch size from 16 to 1024 by numbers to the power of two and from 50 to 250 by numbers with a difference of 50 between each one. Batch size demonstrated to have a great effect on accuracy. So, the higher size means greater accuracy. However, it brings a big challenge related to the computational cost [41].
- **Number of Epochs:** Is the number of iterations that the CNN will work through the whole dataset. It can be set between one and infinity, stop it using other criteria

besides a fixed number of epochs, such as a change or lack of change in model error over time. Actually, there exists a technique called Early Stopping to determine when to stop training an specific model. For instance, by using a combination of the validation error and the training error to stop the model when the error is over than a specific threshold [42].

- **Number of Layers:** In practice its something usual that a three-Layer-Neural-Network-architecture will outperform a 2-layer one. Nevertheless going deeper rarely helps much more. A big exception to this is the type of architectures that are used in this project, CNN, where the deeper they are, the better they perform (In most cases) [43].
- **Convolution Kernel Width:** As it was mentioned earlier, another very important hyperparameter in CNN corresponds to the convolution kernel width because it influences the number of parameters in a model which, as a matter of fact, influences its capacity. The kernel size can be changed across the network. However, a common size predefined corresponds to a 3x3. Here, the stride and padding play a crucial role to determine the size of the output image.

2.4.2 Major concerns

There exist different factors which can be considered to determine how good a method or architecture for video classification is. These factors which are taken into consideration represent a crucial role when designing a network architecture, specially when it comes to working with videos. In addition, these issues have become greatly followed by numerous researchers in previous and current investigations in order to improve them. Referring to the main factors there are the following:

- **Computational Complexity:** It refers to the number of operations needed by a network architecture and it depends on the number and types of layers being used.
- **Computational Speed:** Highly related to the computational complexity. In summary, the greater the number of operations required by the network architecture, the slower will become its computational speed when training a model.
- **Accuracy:** It is also a crucial factor to determine if an architecture can be a feasible solution in terms of true and false detection.

2.5 CNNs Models for Video Classification

2.5.1 Performance Criteria

From literature, the most relevant CNNs models are characterized based on the following criteria:

- **Accuracy:** It is crucial for this project to work with CNNs where its accuracy capability stands out. This because of detection and classification tasks. Inception V-3 demonstrates that it is one of the best networks in terms of accuracy, by achieving a top-5 error of 3.58% [1]. On the other side, ResNet appears also as a great candidate, due to the fact that its top-5 error on the ILSVRC rate was 3.57% [2]. Currently, both networks are able to surpass human-level performance in terms of image classification. In addition, C3D also acts as a candidate, because this network, is capable to work with a stack of images (videos), instead of just one. For this reason, it can not be directly compared with the previous ones, because they work with images only. Nevertheless, C3D's accuracy on anomaly detection task also outperforms the rest ones [8], therefore, it is also selected as a candidate to be explored.
- **Size (depth) of the Network:** In general, the greater the number of layers of a network, the greater will be its depth. This selection criteria is useful at the moment of determining other aspects related to the network's computational time/-complexity. Candidates showed the following results: Inception V-3 [1] contains a total of 48 layers in total, a value that do not differ widely with the number of layers of ResNet-50 [2], as the name says, it contains a total of 50 layers. Nevertheless, when C3D [3] network appears, its number of layers reduced at least 3 times when compared with the previous ones, containing only 15 layers in total.
- **Computational Complexity:** Estimating the computational complexity becomes very important when determining if a network can be trained on an specific computer. Actually, computation in deep neural networks is dominated by multiply-adds in Fully Connected and Convolutional Layers. Nevertheless, the number of floating points operations per second (FLOPs) and parameters of a network must be estimated to obtain the computational complexity. At this point, based on previous criteria analysis, it can be observed that there exists a notable similarity between Inception V-3 [1] and Resnet-50 [2] in terms of accuracy and number of layers. Nevertheless, based on the number of FLOPs of each network, it can be observed that the first candidate holds a total of 6 billion, while the second candidate requires only 3.8 billion. These values surely looks enormous, but lets consider that previous networks have been used only for image classification. Now, C3D outstands both Inception V-3 and ResNet-50 in this aspect, having a total of 38.5 billion [3] FLOPs.
- **Capability to work with Average Computational Resources:** It is important to determine, particularly for real scenarios, if a CNN is going to be able to function adequately on a desktop computer or laptop. Based on the computational complexity on the previous criteria, it is obtained that Inception V-3 [1] & ResNet contains almost the same number of parameters, being 24 million for the first candidate, and 25.6 million [2] for the second one. In addition, even though C3D has a total of 72.9 million parameters, almost three times greater than the previous networks [8], it was selected because in terms of video classification, its number of parameters is

considerably lower.

<i>Network</i>	<i>N Layers</i>	<i>N Parameters</i>	<i>Flops</i>	<i>Common Application</i>
GoogLeNet [5]	27	6.7977×10^6	1.5×10^9	Image Classification
C3D	15	72.9×10^6	38.5×10^9	Video Classification
Inception V-3 [1]	48	24×10^6	6×10^9	Image Classification
ResNet-18 [2]	18	11.8×10^6	1.8×10^9	Image Classification
ResNet-34 [2]	34	21.2×10^6	3.6×10^9	Image Classification
ResNet-50 [2]	50	25.6×10^6	3.8×10^9	Image Classification
ResNet-101 [2]	101	44.7×10^6	7.6×10^9	Image Classification
ResNet-152 [2]	152	60.2×10^6	11.3×10^9	Image Classification
DenseNet-121 [4]	121	27.2×10^6	4×10^9	Image Classification

Table 2.1: Results on the number of layers and computational complexity of Network Architectures

Table 2.1 summarizes CNNs analysis mainly applied to image and video classification tasks in terms of layers, parameters and Flops [5], [2], [8]. Based on the above mention criteria and information, this work selected Inception V-3, ResNet-50, C3D and DenseNet models to be studied in order to take the best characteristics to propose a new approach for video classification.

2.5.2 CNN Models

The main structure of the selected CNNs is illustrated in Fig. 2.3 and Fig 2.4.

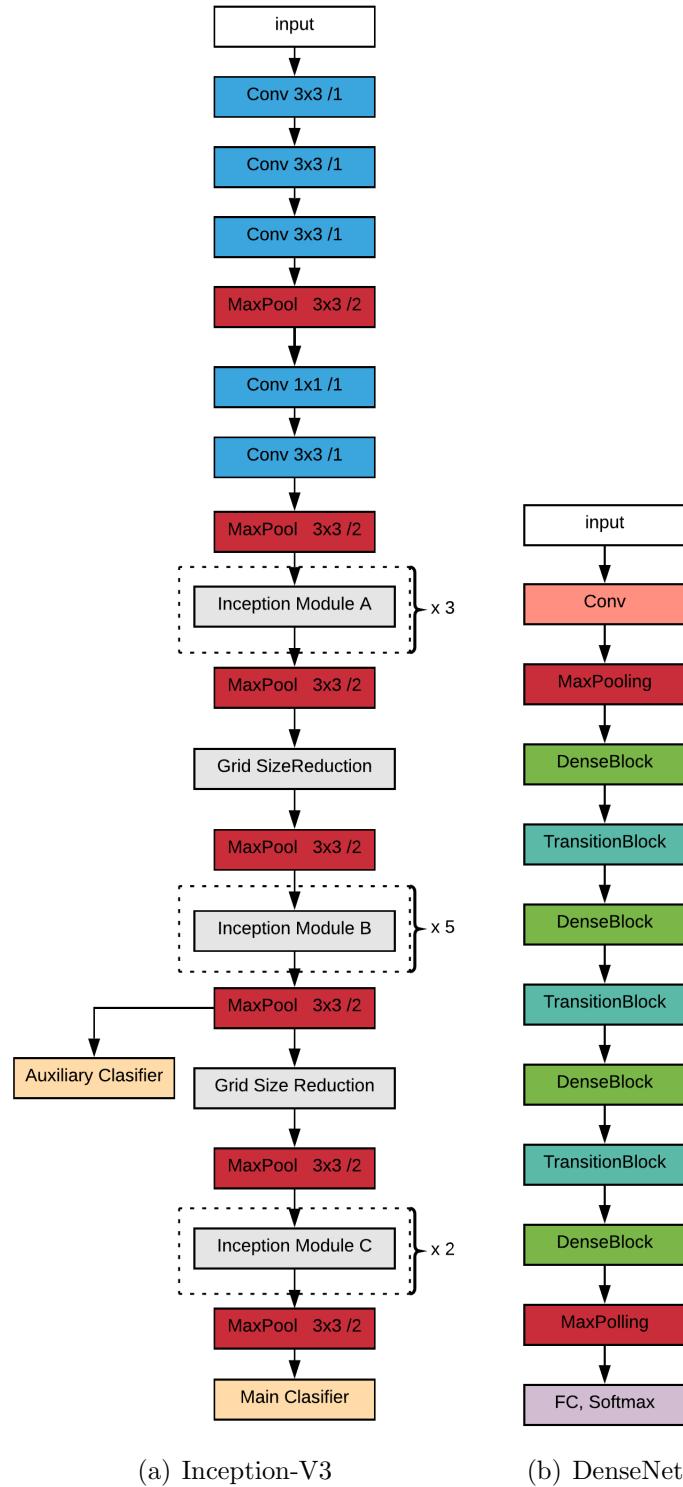


Figure 2.3: Architectures 1

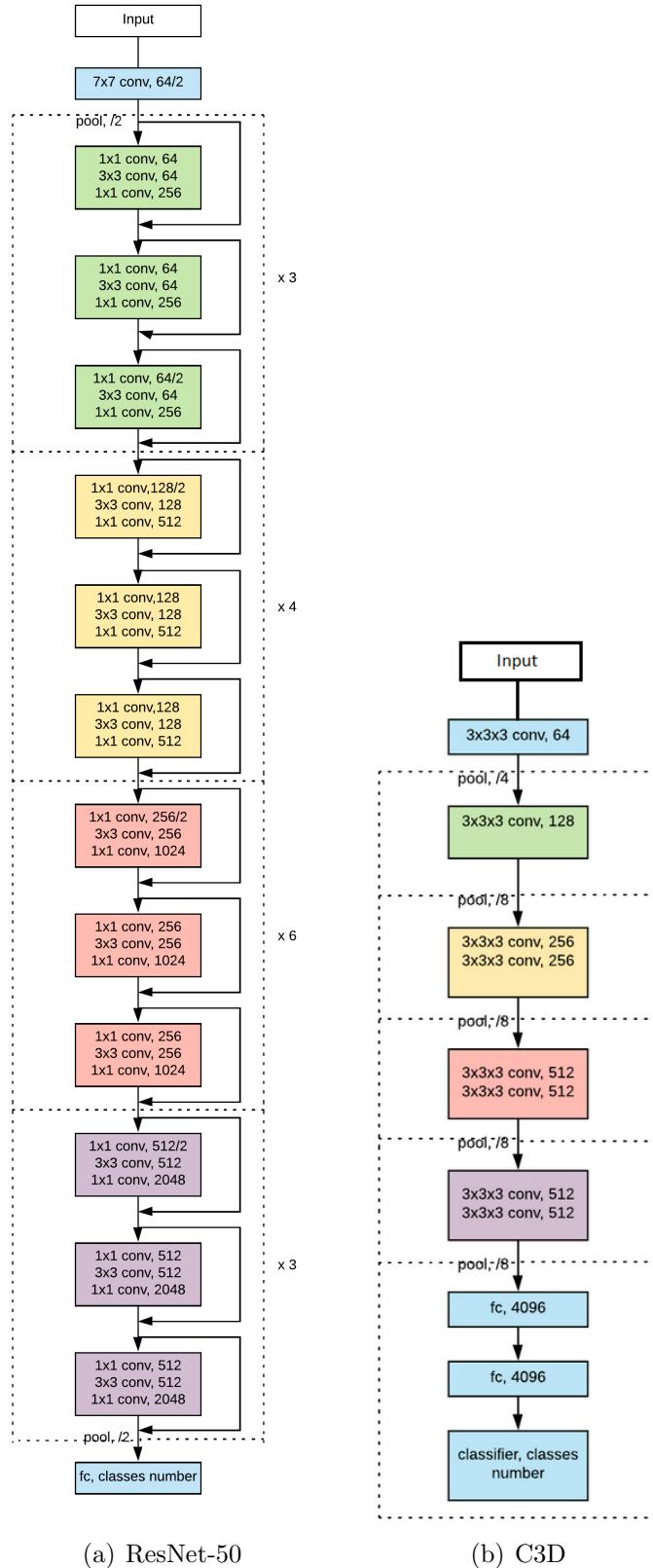


Figure 2.4: Architectures 2

It can be seen that the most distinctive characteristic of each CNN architecture goes as follows:

- Inception V-3 [1] contains a set of different inception modules, grid size reduction and an auxiliary classifier.
- ResNet-50 [2] main feature is the implementation of Residual Blocks all over the network.
- C3D [3] contains [8] a set of 3-Dimensional convolutional layers, something that none of the previous networks has.

This types of networks are some the most commonly used architectures for video classification tasks, due to the fact that each one of them have been used and demonstrated a good performance in classification and object detection tasks withing images.

2.5.3 Inception V-3 [1]

It is an enhanced version of its predecessor GoogleNet [5]. Inception V-3 is characterized by size changes of the convolutional layers of the inception module as is presented in Fig. 2.3(a). It contains three specialized modules: A, B, and C depicted in Fig. 2.5, 2.6, and 2.7, respectively. Module A replaces its convolutional layers of 5×5 with two ones of 3×3 ones as is shown in Fig. 2.5. Module B takes each 3×3 convolution of the Inception Module A Fig. 2.5 to replace them with $1 \times n$ convolution follow by a $n \times 1$ convolution. Author in [5] demonstrated that a value of n to 7 allows to reach good results. Module C takes each 3×3 convolution output of the Inception Module A Fig. 2.5 to expand them by 1×3 and 3×1 in a parallel way.

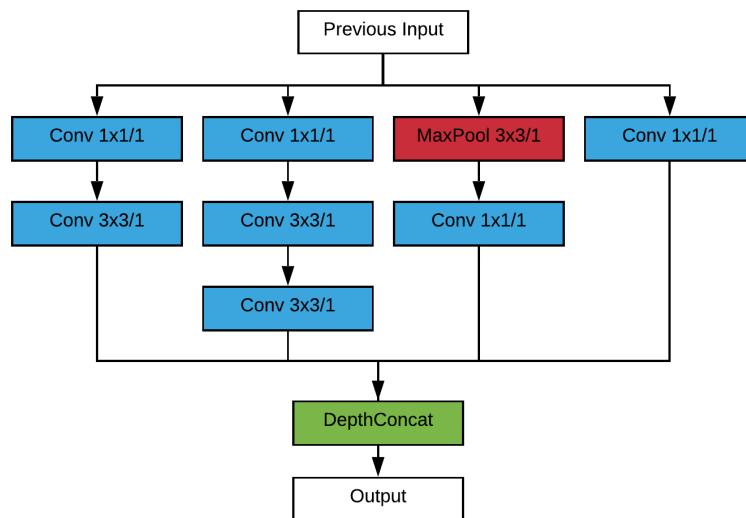


Figure 2.5: Inception Module A [5]

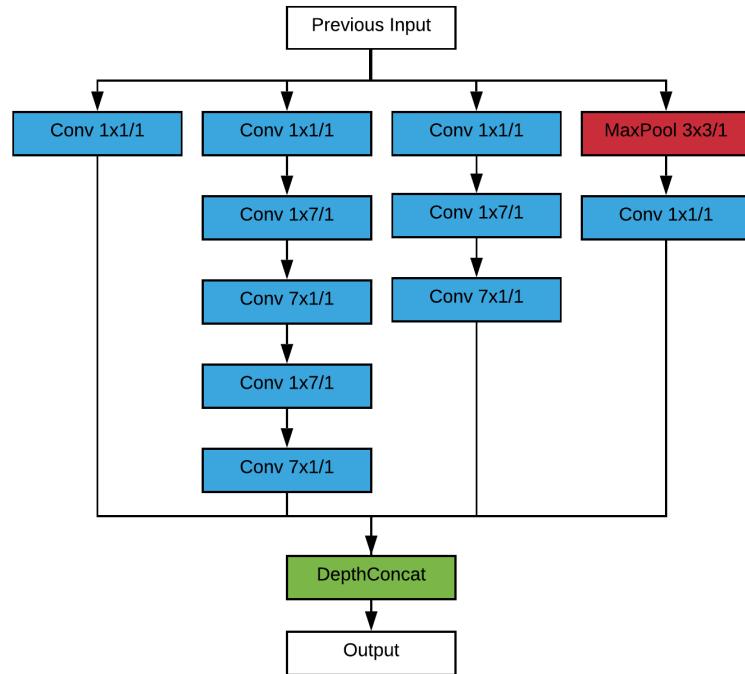


Figure 2.6: Inception Module B [1]

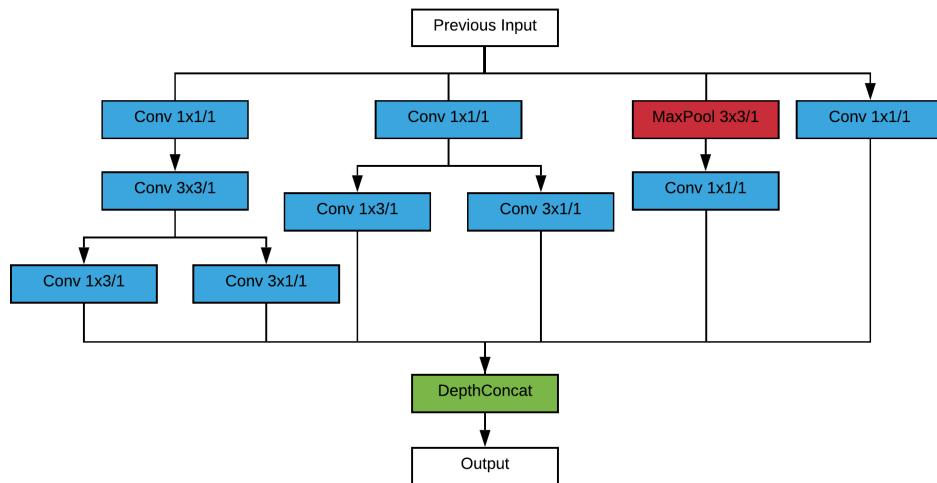


Figure 2.7: Inception Module C [1]

In addition, there are two types of classifiers on this network. Both of them can be found on Fig. 2.8 On the left side is the Auxiliary classifier which is used to regularize the weights of the intermediate layers until the first ones, on the other side is the Main classifier that is in charge of classification.

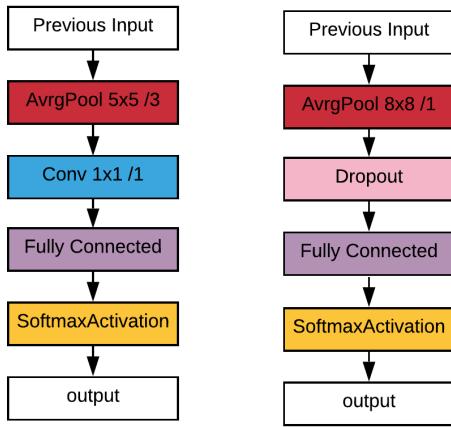


Figure 2.8: Auxiliary & Main Classifiers [1]

2.5.4 Residual Network (ResNet) [2]

The ResNet layers are explicitly formulated as learning residual functions with reference to the layer inputs, instead of learning unreference functions. Its depth varies between 18 and 152 convolutional layers. ResNet introduces shortcut connections that bypass a signal from one layer to the next. Such connections pass through the gradient flows of networks from later layers to early layers, and ease the training of very deep networks. Residual Block illustrated in Fig. 2.9 allows the connection to bypass a signal from the top of the block to the tail. Currently, ResNets consists of multiple residual blocks as is shown in Fig. 2.4(a).

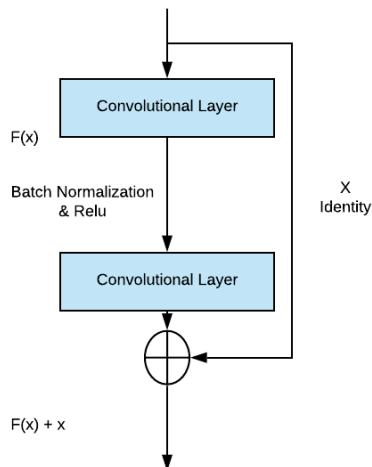


Figure 2.9: Residual Block. [2]

2.5.5 Convolutional 3D Network (C3D) [3]

It is well suited for spatio-temporal feature learning. C3D has the ability to model temporal information in a better way due to the 3D pooling operations and convolution it uses as is shown in Fig.2.4(b). 3D convolution mainly preserves the temporal information of the input signals that result in an output volume.

Its common structure consists of 5 convolution layers, each one followed by a pooling layer, and then 2 fully-connected layers and a softmax loss layer to predict action labels. The number of filters for the convolution layers often are 64, 128, 256, 256 and 256, respectively. All the filters often called convolution kernels have a size d that represents its temporal depth. All convolutional layers are applied with the appropriate padding and a stride of 1. Then, all pooling layers are max pooling with a kernel size of 2x2x2 (with exception of the first layer that has a size of 1x2x2) with a stride of 1, so the reduction factor is 8. Normally, both fully connected networks have 2048 outputs, and the initial rate is 0.0003 that is divided by 10 after 4 epochs.

Fig. 2.4(b) presents an homogeneous setting with convolution kernels of 3x3x3. With this kernel, the training of the network and its deepness depends on the computation affordability and the machine memory limit. The network consists of 8 convolutional layers, 5 pooling layers, 2 fully connected layers and a softmax output layer represented as 'classifier'. All the convolutional filters used in the network are 3x3x3 with a stride 1x1x1. The pooling layers are with a stride 2x2x2, except the first that has a size of 1x2x2 and a stride of 1x2x2, with the objective of preserving early information. As a result of each fully connected network, there are 4096 output units.

2.5.6 DenseNet [4]

A promising solution for overcoming the ResNet performance in several aspects. In this sense, DenseNet [4] has several advantages such as the reduction of the vanishing-gradient problem, increasing feature propagation, feature reuse and reduction of parameters number. There are different versions of DenseNet which differs by the numbers of layer as is shown in Fig. 2.10. The main structure of any DenseNet version (see Fig. 2.3(b)) consists on a Convolution layer and a MaxPooling layer followed by a sequence of 4 Dense Blocks (see Fig. 2.10(a)) with a Transition Block (see Fig. 2.10(c)) between each one, ending with a FC layer with softmax activation as the main classifier.

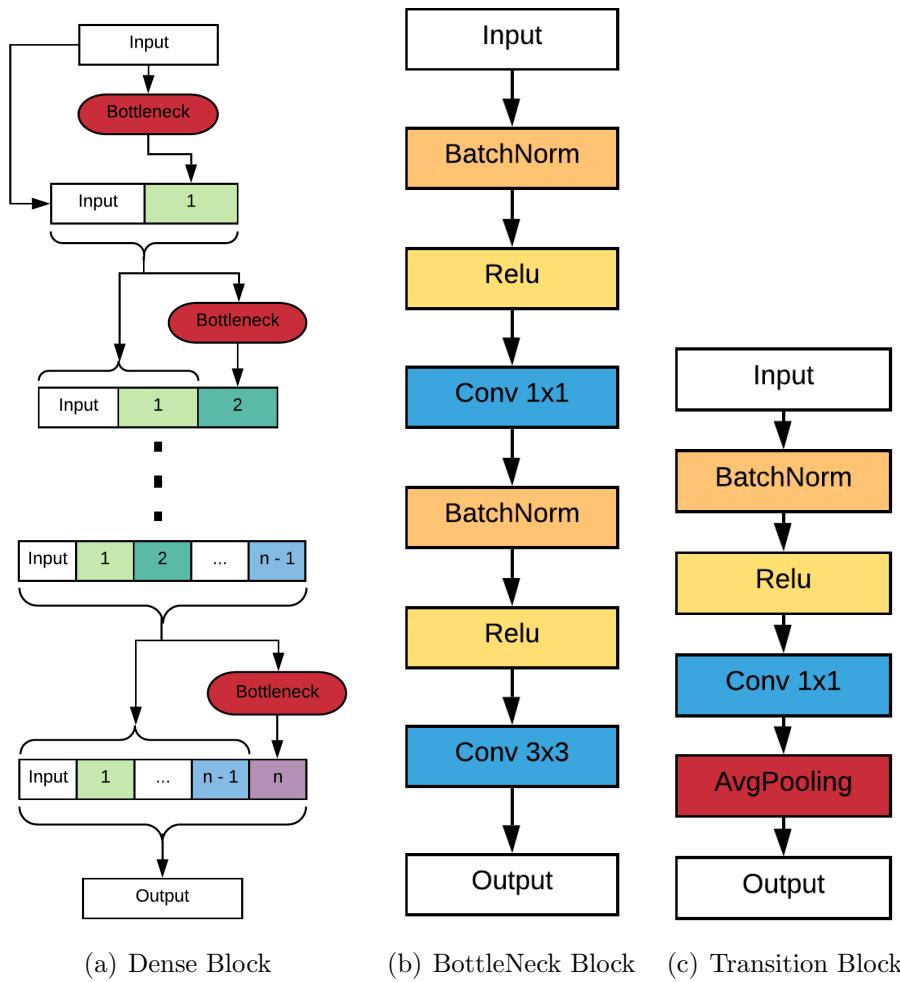


Figure 2.10: DenseNet Architecture [4]

Particularly, DenseNet [4] takes the following characteristic:

On Dense Block (Fig. 2.10(a)), each n layer obtains additional information from all the $n - 1$ previous layers and gives its feature maps to all following layers. Each n layer passes by the Bottleneck Block (Fig. 2.10(b)) and its results is concatenate with the same n layer in order to obtain the next $n + 1$ layers.

Between each Dense Block there is a Transition Block (Fig. 2.10(b)) which is responsible for the compression of feature maps. If the output of a Dense Block is of size m , the transition layer generates an output of size θm , where $0 < \theta \leq 1$ is the compression factor. For this architecture, a value of $\theta = 0.5$ is used, as in [4].

Chapter 3

Methodology for Designing an Anomaly Detection System using CNNs

The established methodology for designing an anomaly detection approach is depicted in Fig. 3.1. It starts with a preliminary analysis mainly to identify the most relevant CNN models and their distinctive characteristic that influence to performance results. The next stages are named as: Design and implementation (Selected and Proposed CNNs models), Data Preparation, Parameter Settings, Training, Validation and Implementation of a UI Prototype.

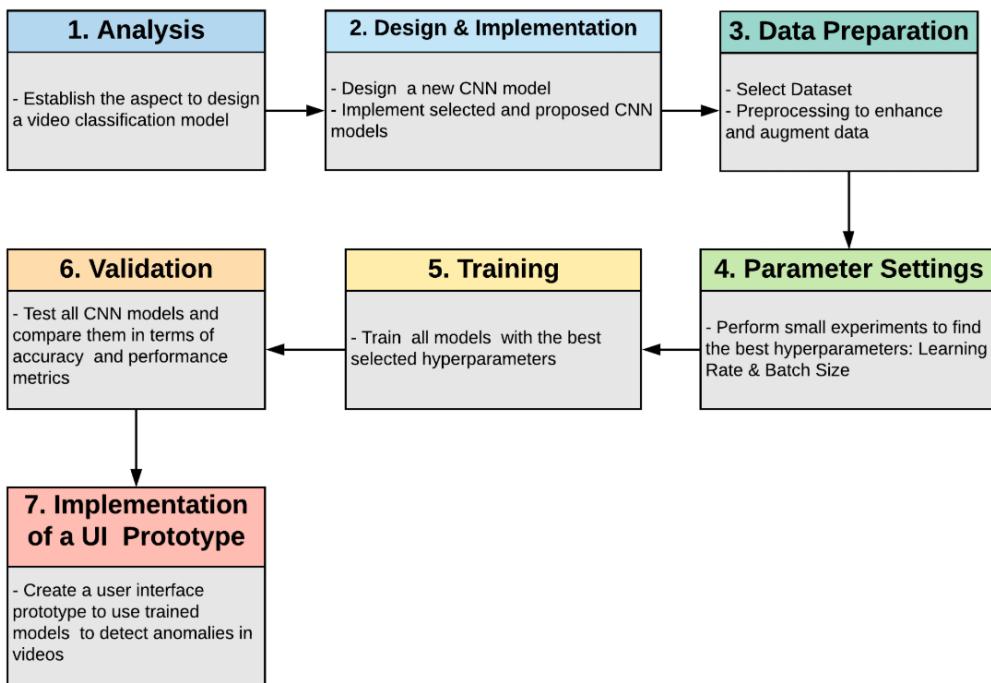


Figure 3.1: General Block Diagram

3.1 Analysis

On this stage, the main factors to take into account at the moment of design a network architecture are defined. Then, the selection of hardware and software tools is done. The final step is to establish the selection of the optimal models for video classification for future implementation and comparison with respect to the new proposed model.

3.1.1 Performance Criteria to Build the New Model for Video Classification

First and foremost, this project is expected to obtain a model that is able to perform video classification tasks, specifically anomaly detection in real time. In addition, it is important that the model can be trained with average computational resources. Having said this, the crucial points to take into account to design such model are the following:

- **High Accuracy:** The model to be designed must meet this requirement. A good accuracy will allow the model to work with anomaly detection tasks, where the errors given at the moment of classification must be minimal.
- **Average Computational Cost:** This research project does not count with high computational resources, meaning that the model must be able to perform the training process on a machine with average resources found on the technology market nowadays. The specifications can be found on 3.1.2.2.
- **Response in Real Time:** It is not only necessary that the accuracy must be high. In addition, the designed model must be capable of responding in real time, so that a future system could be implemented using such model.

As it was explored earlier in this project, two CNNs have been selected in order to be implemented and tested. Such networks corresponds to Inception V-3 [1] and ResNet-50 [2], this because of all the points gathered on section 2.5.1 where it was demonstrated that these networks meet appropriately the requirements established.

3.1.2 Hardware and Software Tools

For training, implementation and validation purposes the following tools have been selected, also considering the computational resources available, and the accessible documentation of each one of them:

3.1.2.1 Software Resources

- **Tensorflow [44]:** It is an open source software library released in 2015 by Google to facilitate the way of work of developers and researchers at the moment of training deep learning models. It has the following pros and cons:

- (+) All of its functions can be modified if needed.
 - (+) Can be used on the most known Operative Systems: Windows, Linux, and MacOS.
 - (-) It does not support AMD GPUs. Only works with Nvidia at the moment of working with the GPUs, instead of CPUs.
- **Keras [45]:** An open source neural network library capable of running on top of tensorflow, Microsoft Cognitive Toolkit, and Theano. Designed to a fast immersion on deep neural networks. It has the following pros and cons:
 - (+) Keras models can be easily deployed on many platforms: iOS, Android, Windows, Google Cloud, Raspberry Pi, etc.
 - (+) Keras supports multiple backend engines (not only tensorflow) and thus, does not press the user into a single ecosystem.
 - (-) As it is in the case of tensorflow, keras also does not support AMD GPUs for distributed training.
 - **Python 3.7:** It is an interpreted, interactive, and object-oriented programming language. this programming language fuse modules, exceptions, dynamic typing, very high level dynamic data types, and classes. It has the following pros and cons:
 - (+) It holds an extensive support base thanks to the fact that it is open source and community developed.
 - (+) Nowadays, there exist a great amount of automation, data mining, and big data platforms that rely on Python. A clear example is tensorflow and keras.
 - (-) It is not a good choice for memory intensive tasks. This is because of the flexibility of the data-types which Python offers.
 - **OpenCV [46]:** It is a python library designed specifically to work with computer vision tasks. It is commonly used to perform data pre-processing tasks. It is actually a wrapper based on the original C++ library. It has the following pros and cons:
 - (+) Since it is based on C/C++, its algorithms can be executed through the GPU of the computer, making it faster.
 - (+) The library was designed to work effectively with a low computer ram usage.
 - (-) Due to the fact that this version of OpenCV is written in python. It does not provide the same ease of use when compared with other programming languages (MATLAB).
 - **Pygame [47]:** An open source python library created to develop multimedia applications. Its programming language simplicity is what allows to work together with the any of libraries that can be needed. It has the following pros and cons:

- (+) It requires less assembly than other libraries at the moment of start developing programs or applications.
- (+) As it is developed in Python, it can work all together with all types of libraries, either for computer vision or artificial intelligence, etc.
- (-) Pygame with python is not the easiest library/language to distribute apps. Python is also not necessarily very high performance, meaning that it won't work on many platforms directly.

3.1.2.2 Hardware Resources:

Based on the premise that this work is an exploratory study with research purposes. Training and Validation use available hardware resources, so those tasks are performed on a laptop computer with the following specs:

- Intel Core i7 8th Generation.
- Nvidia MX 150 Max Q
- Ubuntu 18.04 Operative System
- 16 GB Ram

3.2 Design and Implementation

In this stage, firstly, the proposed CNN model is designed. Then, all of the coding part takes place. Once the best network options were explored on previous chapter, next step is to implement each one of them, based on the selected frameworks/libraries, in terms of programming. This means that, by the end of this stage, all of the networks have been correctly created and will be ready to start working with different experiments for improvement purposes.

3.2.1 New Approach: FrankensNet

FrankensNet is the proposed CNN architecture for video frame classification. It aims to unite the advantages of the best networks studied previously and ones from DenseNet [4], a logical extension of ResNet, that will be explained in the current section.

3.2.1.1 Definition of FrankensNet Architecture

In order to keep the advantages of DenseNet [4] (see Fig. 2.10), the layers responsible for feature-extraction were maintained and the classification layers were changed by new modules that will be explained below:

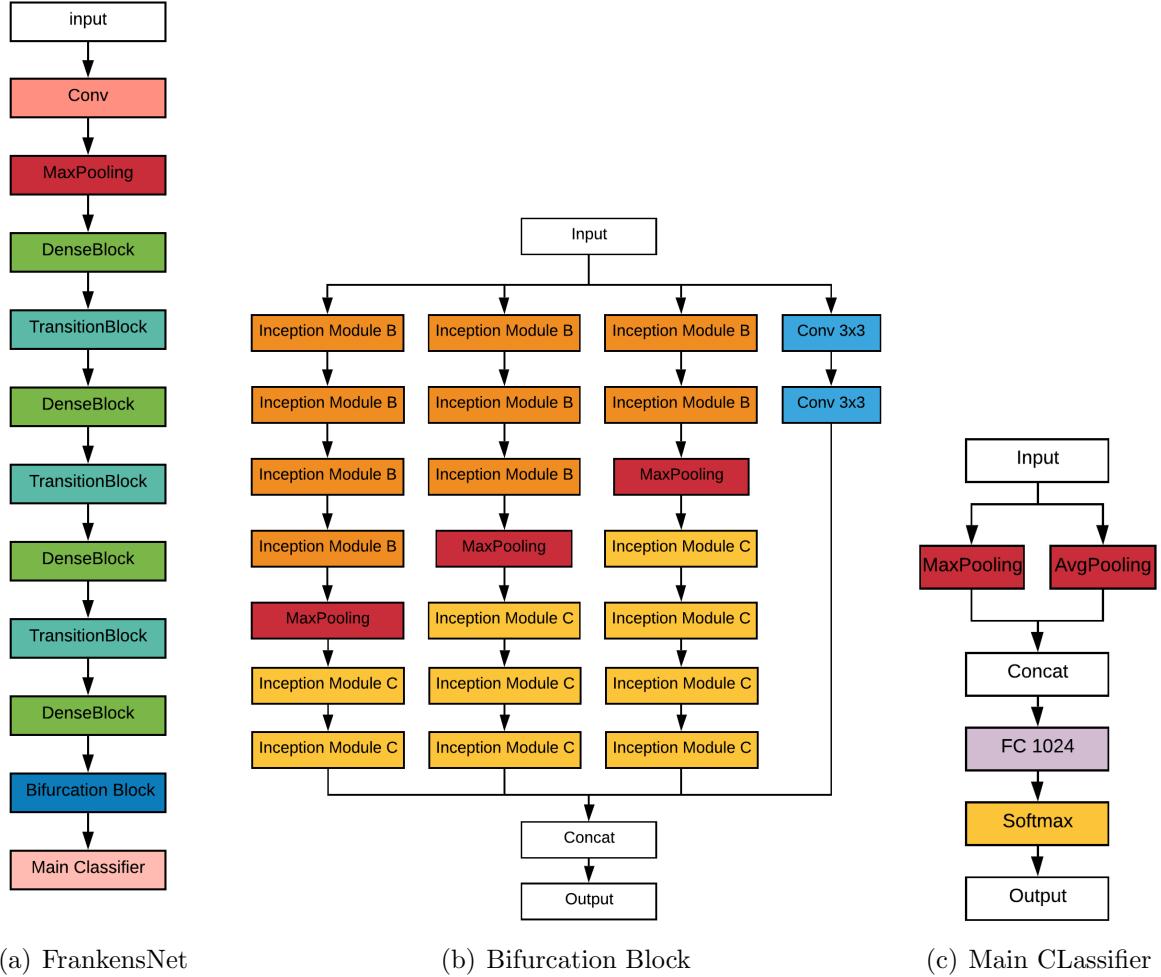


Figure 3.2: FrankensNet Architecture

All the constructions related to inception models [1], [5], [48] use modules which perform a combination of several convolution with different sizes in order to obtain varied spatial information, with this idea, the research project proposes a module that will process the feature maps of the previous layer in different ways in order to obtain different types of information.

At this point it is important to find a way to process and reduce the large amount of information obtained by Dense Blocks (Fig. 2.10(a)) on previous layers, for this reason the best option to choose is the Inception Blocks that meet these characteristics, several ways of obtaining spatial information and dimension reduction, specifically Inception Modules B (Fig. 2.6) and C (Fig. 2.7) according to [1] using these factorizations do not work well in the first layers.

Bifurcation Block 3.2(b) consists of 4 branches, 3 of which are divided into 3 groups, m Inception Modules B, one MaxPooling, and n Inception Modules C, respectively. The values of m and n depend on the branch. In practice, branch 1 has $m = 4$ and $n = 2$, branch 2 has $m = 3$ and $n = 3$, and branch 3 has $m = 2$ and $n = 4$. Branch 4 consists of

two 3×3 convolutions this because the four outputs must have the same dimensions to be able to concatenate them.

On Main Classifier 3.2(c), a max pooling and an average pooling are performed in order to extract the most important features and in a smoothest way. Its result is concatenated and flattened to be passed to a FC with ReLu activation, and Dropout the 0.3 to then pass to a FC with Softmax activation which will be the output layer.

3.2.1.2 Advantages and Disadvantages

FrankensNet has the following improvements and deficiencies:

- (+) It takes advantage of the qualities of the DenseNet by obtaining and re-using as much information as possible on the first layers [4].
- (+) It uses Inception Modules in order to better process information with different spatial information. Thus, it reduces dimensionality of information through different pooling sections [1].
- (-) It requires a greater number of parameters to train. When compared with Inception V-3 [1] and ResNet-50 [2], it quadruples the value.
- (-) It needs more computational resources to function properly [2].

3.2.2 Implementation of CNNs Models

Here are presented the steps needed to implement the explored and proposed CNN models using software tools established in Section 3.1.2 bases.

- Identify the different types of layers that the architecture must contain. For example, convolutional, fully connected, max pooling, etc.
- It must be taken into consideration that a layer parameter might differ in a specific section of the architecture, for example the kernel sizes, padding values, among others.
- Next, it is necessary to establish the structure, this means how layers are going to be ordered all along the network. A great example can be seen on Fig. 3.2.
- In some cases, as it is with Inception V-3 [1] and ResNet-50 [2], it can be of great help to define specific blocks according to the requirements of each network. They are often created with the purpose of improving the organization when implementing the network, and through this, optimizing time.
- Finally, it is possible to achieve the network implementation successfully.

The implemented code of all the models are available in B.1. Nevertheless, the script used to create FrankensNet can be found in this project on the appendix section B.2.

3.3 Data Preparation

3.3.1 Dataset

According to the anomalies established with detection purposes, UCF-Crime Dataset [6] is used. On Appendix section A.1, it can be seen that this is actually the only dataset containing this type of videos with its different crime classes, and even more, containing that amount of information.

The main features of this video surveillance dataset are the following:

- It contains 13 classes in total: Abuse, Arrest, Arson, Assault, Road Accident, Burglary, Explosion, Fighting, Robbery, Shooting, Stealing, Shoplifting, and Vandalism. Fig. 3.3 shows some video frames belonging to each class.
- In total, it consists of 1900 real-world surveillance videos, all obtained on different environments and places, and each one containing an specific realistic anomaly.
- This leads to a total of 128 hours of videos.
- If all of the dataset is downloaded, it is required a total of 100 GB of storage capacity. This is due to the great amount of videos gathered.

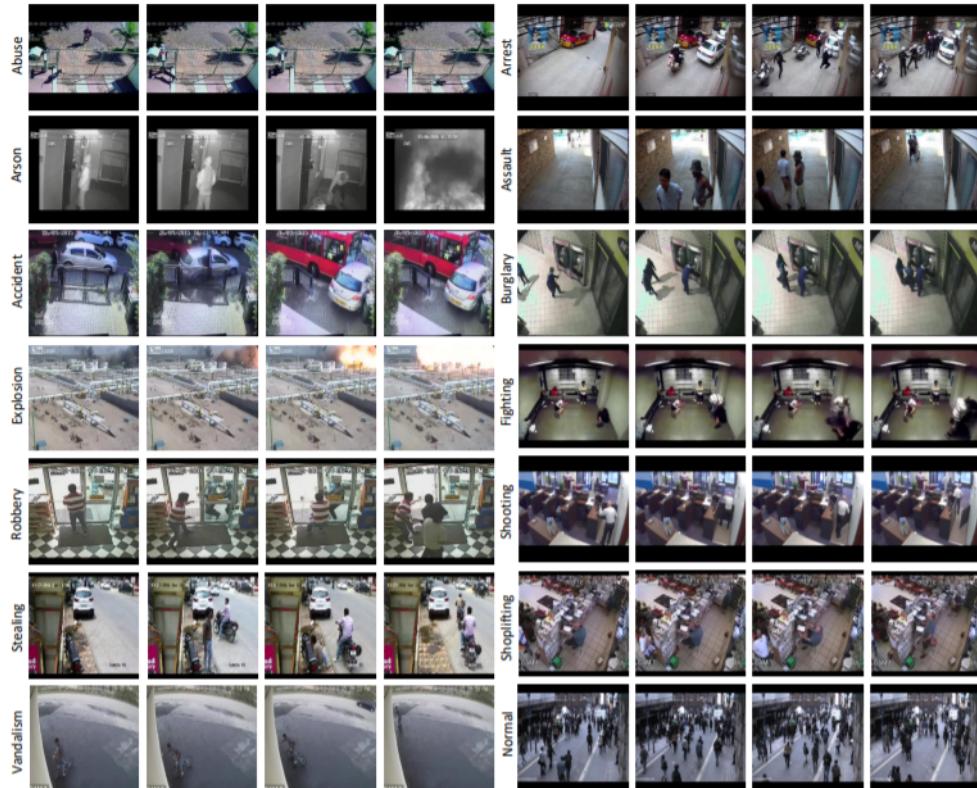


Figure 3.3: UCF-Crime Dataset Categories [6]

3.3.2 Pre-processing

Before entering the images into the network for training purposes, a set of pre-processing operations are applied to them in order to obtain cleaner information. For example, by removing noise from images or to sharpen its edges.

For this project, three different techniques have been applied to identify each scene, enhance images, and augment data, as explained below:

- **Data Labelling:** Prior to any computer vision pre-processing task, it is decisive to make the annotations of each video in order to differentiate the abnormal scenes from the normal ones. For this, it is necessary to manually describe the parts of each video containing an anomaly. Then, the remaining parts of the video are used to fulfill the normal scenes information.
- **Data Augmentation:** It allows to enlarge the variety of data available to train an specific model, without really gathering new information. After using this technique, it is expected that the accuracy of the network will be enhanced. For this project, only 2 operations will be passed down:
 - **Rotate:** The image is being rotated about its center by an specified number of degrees.
 - **Crop:** This operation consists on removing unwanted areas on the image and stay only with the important area.

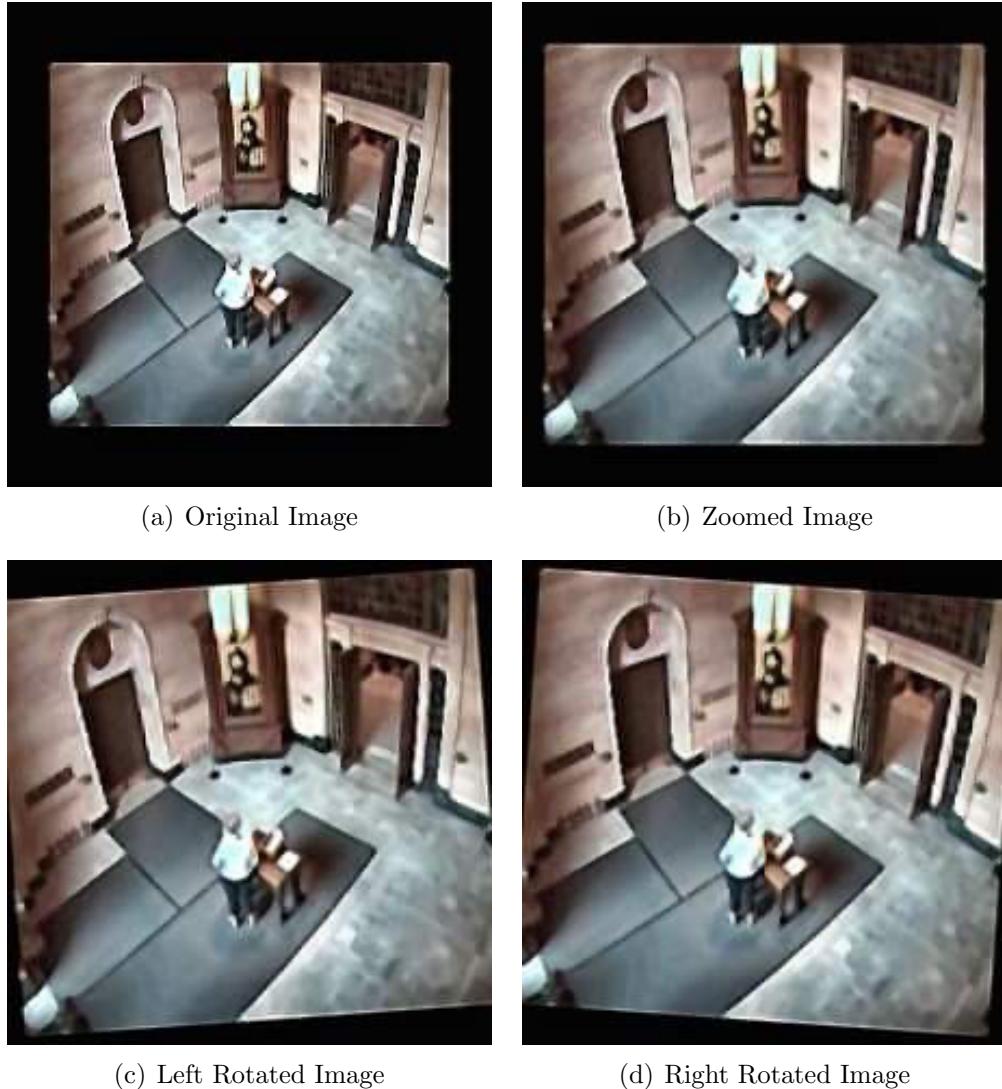


Figure 3.4: Data Augmentation operation on UCF-Crime dataset [6]

- **Contrast Limited Adaptive Histogram Equalization (CLAHE):** Commonly used to improve contrast in images. This is achieved by elongating the most frequent intensity values. This technique is used in the first step in order to grant areas of lower local contrast to achieve a higher contrast overall and to avoid over amplification related to noise.



(a) Original Image

(b) Image with CLAHE

Figure 3.5: CLAHE example on UCF-Crime dataset [6]

- **Median Filter:** It is a digital filtering technique, used to remove noise from an image or signal. Explicitly, the median filter works by replacing a pixel by the median of all pixels in a given neighborhood. One of the main advantages of using this filter as the second step, is that it keeps the edges while removing noise from the image.



(a) Original Image

(b) Image with Median Filter

Figure 3.6: Median Filter example on UCF-Crime dataset [6]

- **Unsharp Masking:** It is an image processing technique used to sharpen images. This is done by obtaining first, the Laplacian of the image, which corresponds to the second derivative, and later removing the previous Laplacian obtained from the original image. This is the third and final step because it enhances the edges, along with

the median filter, in order to improve information obtained from videosurveillance cameras.



Figure 3.7: Unsharp Masking Example on UCF-Crime dataset [6]

3.4 Parameter Settings

This stage determines which are the hyperparameter values that best fit the network to improve its accuracy and thus, its effectiveness, such as learning rate and batch size. The main definition and importance of each hyperparameter is already established on subsection 2.4.1. Nevertheless, it is important to remark once again that determining the optimum values of the hyperparameters is crucial, because only this way, the greatest possible accuracy will be achieved by each network for anomalies detection. The hyperparameters are adjusted by experimental tests described in section 4.5.

3.5 Training

Once it is established which are the best hyperparameters for each architecture, along with data handling processes in terms of data augmentation and pre-processing tasks on previous stage. Here, network's training process starts. This means that they will learn how to solve tasks which have been assigned to them (in this case, anomaly detection) and then being tested to calculate its performance.

The training implemented code for FrankensNet is described in section 4.1 and the corresponding code of all the explored CNN models are available at B.1 .

3.6 Validation

This part contains the implemented code to determine the precision and accuracy of each network. Furthermore, tests are analysed based on metrics settled in section 4.3, e.g training time, false positive rate, true negative rate, etc. In the end, all of the information gathered from different tests is stored for subsequent analysis purposes.

When the training phase is done, each model contains the weights learned during the training. In order to use the model trained it is indispensable to make a script which must be able to load the weights of the model, and an specific video to start with detection tasks.

Due to the study presented on section 2.5.1, FrankensNet Model will be compared with Inception V-3 [1] and ResNet-50 [1], which have been implemented and executed similar sequences of the new proposed model. The validation implemented code for FrankensNet is described in section 4.2 and the corresponding code of all the explored CNN models are available at B.1.

3.7 Implementation of a Graphical UI for an Anomaly Detection Prototype

In order to provide a tool available to use trained CNNs models, this work implements a user interface for anomaly detection in real scenarios. To run the user interface prototype it is required the following:

- A computer which has installed Python 3.7 on it.
- OpenCV, Keras, Tensorflow and Pygame libraries.
- Video surveillance footage have to be stored on the computer.
- All of the weights obtained from training process must be saved on the computer.

Chapter 4

Experimental Setup

In this chapter, training and validation implementation, as well as the metrics and the applied tests to explore the performance of Inception V-3 [1], ResNet-50 [2], and Franken-sNet (proposed approach) are described.

4.1 Training Implementation

Here are presented the steps needed to train a network:

- Establish the following parameters: directories where train and validation images are located, batch size, height and width of the images, dropout, number of epochs, and learning rate.
- Generate train and validation data. This step is needed to create an input that is accepted by an specific model.
- Establish the behaviours that will be occurring on this process. For example, to save information of accuracy and loss from each epoch, plot figures at the end of the training, etc.
- Finally, establish where is going to be stored the trained model for future tasks.

The script used to train a model can be found on B.3

4.2 Validation Implementation

Once the model has been trained, by following the steps specified on previous section, it is time to qualify the network in terms of the metrics established on section 4.3.

Each model validation is performed in the following way:

- Determining training time of the network, already given one the training process concluded.

- Obtaining the network's final accuracy, loss, validation accuracy and validation loss during training process.
- Retrieve false positive, false negative, true positive and true negative rate. Taking in consideration this:
 - **False positive:** When an anomaly is not happening on the scene, but the model consider it as one.
 - **False negative:** When an anomaly is happening on the scene, but the model does not detect it.
 - **True positive:** When an anomaly is happening on the scene, and the model does actually consider it as one.
 - **True negative:** When an anomaly is not happening on the scene, and the model does not detect one.
- Establishing if the network is able to perform detection tasks in real time.

The corresponding validation code of all the explored CNN models are available at B.1, they are not presented here because of its size.

4.3 Metrics

To establish a performance comparison among these 3 different architectures designed for video classification, the metrics established corresponds to the following:

- **Training time:** This metrics is very useful because it will allow to understand the time it takes a given network to start working properly.
- **False positive, False negative:** Given the fact that an alarm system is being sought, knowing the different rate (When something that it is not an anomaly is considered as an anomaly, a fake detection, etc) will determine if the network can be used for this task.

False Positive rate:

$$\frac{FP}{FP + TN} \quad (4.1)$$

False Negative rate:

$$\frac{FN}{FN + TP} \quad (4.2)$$

- **True positive, True negative:** They allow to understand when the network is detecting anomalies and the cases when an anomaly appears on the scene but the network is not able to detect it, respectively.

True Positive rate (TPR):

$$\frac{TP}{TP + FN} \quad (4.3)$$

True Negative rate (TNR):

$$\frac{TN}{TN + FN} \quad (4.4)$$

- **Accuracy:** It corresponds to the fraction of the correct predictions over the total number of predictions. It is calculated with the following formula:

$$\frac{TP + TN}{TP + TN + FP + FN} \quad (4.5)$$

- **ROC curve:** Based on the values obtained previously, this will allow to determine the performance of a classification model at all classification thresholds. It is represented by two parameters: TPR and FPR. It illustrates how much a model can distinguish between classes.

4.4 Data Preparation:

Dataset used corresponds to a 20 % total of the UCF-Crime dataset [6] described in section A.1. In this project, the number of anomalies was reduced to only 3 categories: abuse, arrest, and assault. This decision has been made mainly for two reasons:

- **Optimization of resources:** The computer on which all the experiments are going to be performed (mentioned on section 3.1.2.2) has to be able to work effectively with all of the information collected for the network and its operations. If more categories were used, it would not be possible to perform the experiments as expected and lead to errors related to the lack of memory or space in the computer.
- **Low accuracy avoidance:** Only the three more similar anomalies were chosen in order to avoid the network to over-learn very specific cases of anomalies and work in an environment with more general information instead. For this reason, categories like road accidents or explosions were ignored, so that there is not so much variability of information. This is intended to gain network accuracy.

4.4.1 Data Augmentation:

4.4.1.1 Rotate:

In this case, the number of degrees corresponds to a value of 4 degrees to the left and 4 degrees to the right. The reason of choosing these values is because it is not sought to alter the original image too much and still make it look as it is being obtained from videosurveillance cameras. The function, created on python with the use of OpenCV library can be seen on B.4.

4.4.1.2 Crop:

Prior to anything, the image size is of 224×224 pixels. The parameters used for this operation were: $y1=10$, $y2=210$, $x1=10$, and $x2=210$. These parameters were selected because main information is located on areas near the center of the image in most cases. Once again, the function, created on python with the use of OpenCV library is on B.5.

4.4.2 Data Pre-processing:

4.4.2.1 Contrast Limited Adaptive Histogram Equalization

Here, the gridsize was established to a value of 8 (already set by default). This parameter refers to the size of rectangular tiles in which original image is going to be divided, in this case tiles of 8×8 . The code used to perform this task can be found on B.6.

4.4.2.2 Median Filter & Unsharp Masking

Parameters used here correspond to: $\text{sigma}=0.5$, and $\text{strength}=0.8$. Sigma value helps to control the action of the median filter, while strength adjust the amount of the second derivative of the image that is required to be added or removed. Both parameters were established by default. The code used to perform both tasks is located on B.7.

4.5 Experiments

To establish the best parameters settings, a set of experiments were performed on each of the networks using prepared dataset, focused mainly on testing different values for the learning rate and batch size to choose the best option. In addition, based on the metrics given on section 4.3, the rest of the experiments will allow to analyze the performance of each model. For example, determining a networks training time, accuracy and precision, etc.

4.5.1 Experiment 1: Inception V-3 Settings

First part of the experiment is to select the best learning rate value for the network. In this case, 3 different values will be selected for the learning rate: 0.001, 0.0001, and 0.00001. Those values are close to those set by default [1]. Thus, the network will have the following parameters established:

- Batch Size: 100.
- Height and Width of the image: 224. These values were used in order to reduce training time and the total of operations to be performed.
- Number of Epochs: 3. It is advisable to perform the experiment with an smaller number of epochs than in the final training.

- Dropout: 0.5. A value set by default for the network in order to prevent CNNs from overfitting.
- Total number of Images used: 80000. Not all of the images are necessary in this part of the experiment. For this reason, the number was reduced by a third of the total.

Once the best learning rate is selected, next step is to find the best value for Batch Size. Again, 3 experiments will be performed by using the following Batch Sizes : 100, 200, 300. Those values were also selected because they are close to the ones by default.

4.5.2 Experiment 2: ResNet-50 Settings

In the first place, the purpose of the experiment is to select the best learning rate value for the network. In this case, 3 different values will be selected for the learning rate: 0.0001, 0.00001, and 0.000001. It can be seen that these values differ a little from the ones used on Inception V-3 experiments, this is because such values were selected taking as reference the the ones used on [1] . In addition, those values are close to the ones set by default for the network [2]. Thus, the network will have the following parameters established:

- Batch Size: 100
- Height and Width of the image: 224. These values were used in order to reduce training time and the total of operations to be performed.
- Number of Epochs: 3. This experiment should not be executed to perform all epochs. It is given only to fine tune parameters.
- Dropout: 0.5. A value set by default for the network in order to prevent CNNs from overfitting.
- Total number of Images used: 80000. The number of images was reduced by a third of the total to avoid excessive training time for this experiment.

Once the best learning rate is selected, next step to proceed is to find the best value for Batch Size. Again, 3 experiments will me performed by using the following Batch Sizes : 100, 200, 300. As with the learning rate, those values are similar to the ones set by default for the network.

4.5.3 Experiment 3: FrankensNet Settings

Quite similar to the first two experiments, the best learning rate value for the network is being sought first. In this case, 3 different values will be selected for the learning rate: 0.001, 0.0001, and 0.00001. Once again, it can be seen that these values are similar to the ones used on Inception V-3 experiments, and given the fact that the network is actually using the best features of each model, those learning rates were chosen. Thus, the network will have the following parameters established:

- Batch Size: 100
- Height and Width of the image: 224. As in the previous experiments, to reduce training time and the total of operations to be performed.
- Number of Epochs: 3.
- Dropout: 0.5. Set by default for the network in order to prevent CNNs from overfitting.
- Total number of Images used: 80000. To fine tune the parameters it is not necessary to use the complete set of images.

Once the best learning rate is selected, next step to proceed is to find the best value for Batch Size. Again, 3 experiments will be performed by using the following Batch Sizes : 100, 150, 200. the values were chosen by taking into account the ones used for Inception V-3 and ResNet-50.

4.5.4 Experiment 4: Training Time / Execution Time

This experiment determines the time it takes a single network to completely finish the training process, as it is explained on section 4.1. The number of images that are going to be used in this part is of 250.000 in total. In addition, its execution time will also be tested, given the fact that the model should be able to perform classification in real time to work alongside with the desktop program.

4.5.5 Experiment 5: FP/FN rate, TP/TN rate

This experiment performs the calculations to obtain values of False Positive, False Negative, True Positive, and True Negative Rates. The experiment is done in the following steps:

- From a set of videos, each one is going to be decompressed into a stack of images to later predict its class with the network.
- Once the model have made the prediction, its correctness is verified automatically, and a value of 1 will be added to the corresponding type, based on if the prediction was correct or incorrect.

Chapter 5

Results

Here, the results of all the experiments described in section 4.5, one by one, are presented through tables and figures statistical measured based on metrics defined in section 4.3 .

5.1 Experiments

Given the fact that for this research project, 5 experiments were performed, their results are presented in the posterior subsections.

5.1.1 Experiment 1: Inception-V3

Table 5.1 denotes the results obtained when performing three experiments, each one with a different learning rate. This allows to determine which value used for the learning rate will fit better to improve accuracy of the network.

<i>Learning Rate</i>	0.001	0.0001	0.00001
<i>Inception-V3 Accuracy</i>	57.59%	79.43%	80.27%

Table 5.1: Experiments performed on Inception-V3 with three different Learning Rates

First, it can be seen that the lowest accuracy value was obtained with a learning rate of 0.001. Thus, the maximum accuracy value was achieved with a learning rate of 0.00001. Nevertheless, not everything is good because the lower the learning rate, the greater will be training time. However, given the fact that this work is looking for the best accuracy (due to anomaly detection tasks), the smallest learning rate value was selected. Next step is to determine which batch size, along with the learning rate already settled, is the best.

<i>Batch Size (with LR of 0.00001)</i>	100	200	300
<i>Inception-V3 Accuracy</i>	80.27%	79.88%	79.41%

Table 5.2: Experiments performed on Inception-V3 with three different Batch Sizes

Now, based on Table 5.2, it can be observed that the minimum accuracy was obtained with a batch size of 300. This is understandable because in practice, the greater the batch size, the lower will be the accuracy. For this reason, when using a batch size of 100, the maximum accuracy was achieved. Of course, this value is not that distant from the one obtained with a batch size of 200. Similar to the learning rate, the smaller the batch size, the longer will be training process.

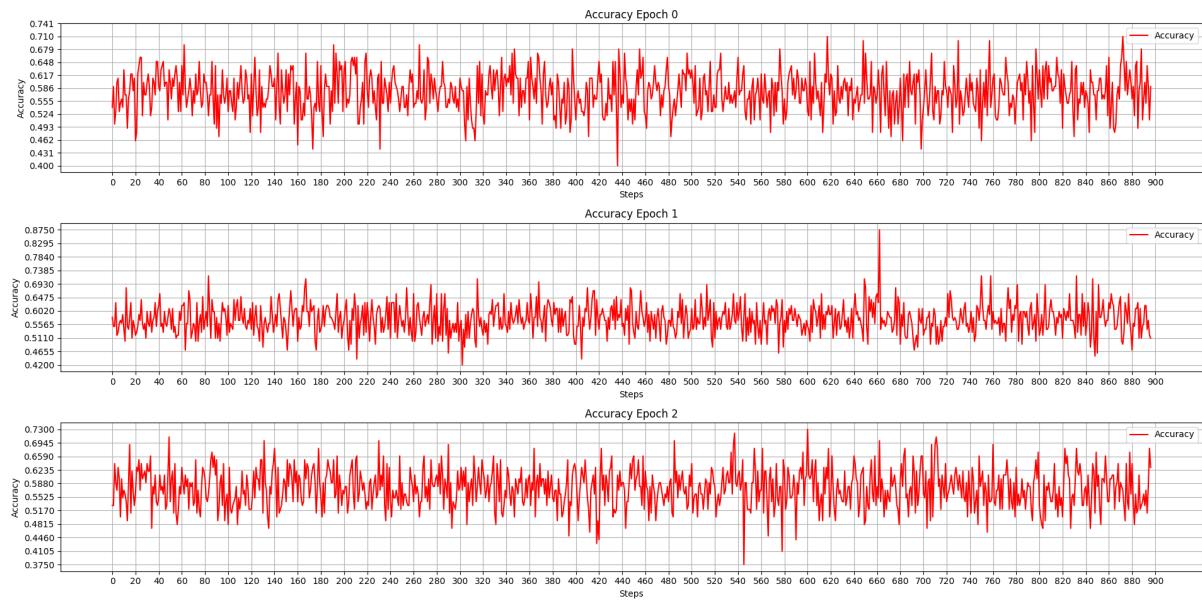


Figure 5.1: Minimum Accuracy. Parameters: LR=0.001 ; BS=100

On Fig. 5.1 can be observed how accuracy changes through each step in an epoch until the minimum is obtained. The main aspect that can be denoted here is how volatile are these changes, meaning that they do not necessarily follow a pattern.

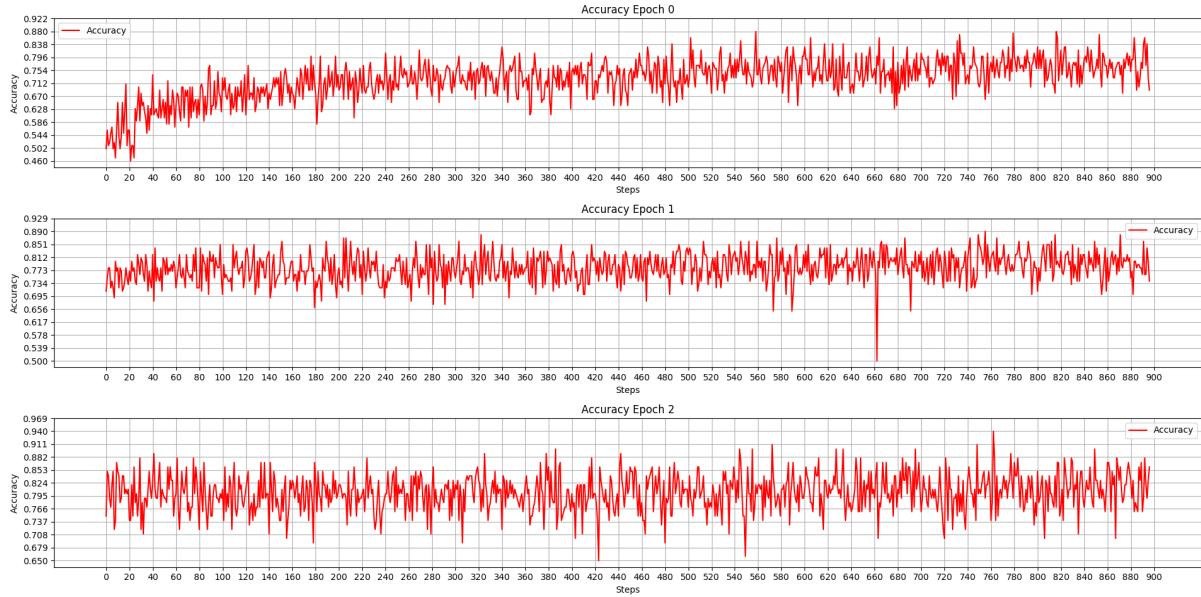


Figure 5.2: Maximum Accuracy. Parameters: LR=0.00001 ; BS=100

Finally, as it can be seen on Fig. 5.2, it shows how accuracy values changed through each of the 3 epochs. Above all, on the first epoch can be seen how values begin to stabilize after the first steps on a smaller range (60% - 90%). Nevertheless, on epoch 2 and 3, range keeps diminishing until staying between 70% -90% and this explains why the maximum value of accuracy was 89%.

5.1.2 Experiment 2: ResNet-50

As it can be seen on Table 5.3, a set of 3 experiments were tested on the network using different learning rate values. Here, it can be observed that the minimum accuracy is obtained with a learning rate of 0.000001. With this learning rate, the network was not able to learn adequately. On the other side, the maximum accuracy is achieved with a learning rate of 0.00001. It is important to remark that this learning rate value is the same winner as in the case of Inception V-3. Additionally, the maximum accuracy does not differ too much from the second winner, but as the maximum accuracy is being sought for the network, although training time will be longer, learning rate of 0.00001 is selected.

<i>Learning Rate</i>	0.0001	0.00001	0.000001
<i>ResNet-50 Accuracy</i>	89.46%	90.38%	83.77%

Table 5.3: Experiments performed on ResNet-50 with three different Learning Rates

Next experiment is focused on the batch size. In this case, based on the best learning rate value obtained from previous experiment (0.00001), different batch size values were tested in order to choose the advantageously option.

<i>Batch Size (with LR of 0.00001)</i>	<i>100</i>	<i>200</i>	<i>300</i>
<i>ResNet-50 Accuracy</i>	90.38%	89.89%	89.43%

Table 5.4: Experiments performed on ResNet-50 with three different Batch Sizes

Once again, as it can be seen on Table 5.4, results are quite more interesting. Each accuracy obtained on the second part of the experiment is considerably good and none of them differ too much between them. Nevertheless, only when using a batch size of 100, accuracy exceeded 90%, becoming the winner on this experiment. In addition, when using this batch size, training time lasts longer, but computational memory resources are reduced. Having said that, a batch size of 100 and learning rate of 0.00001 are the best options for our detection requirements.

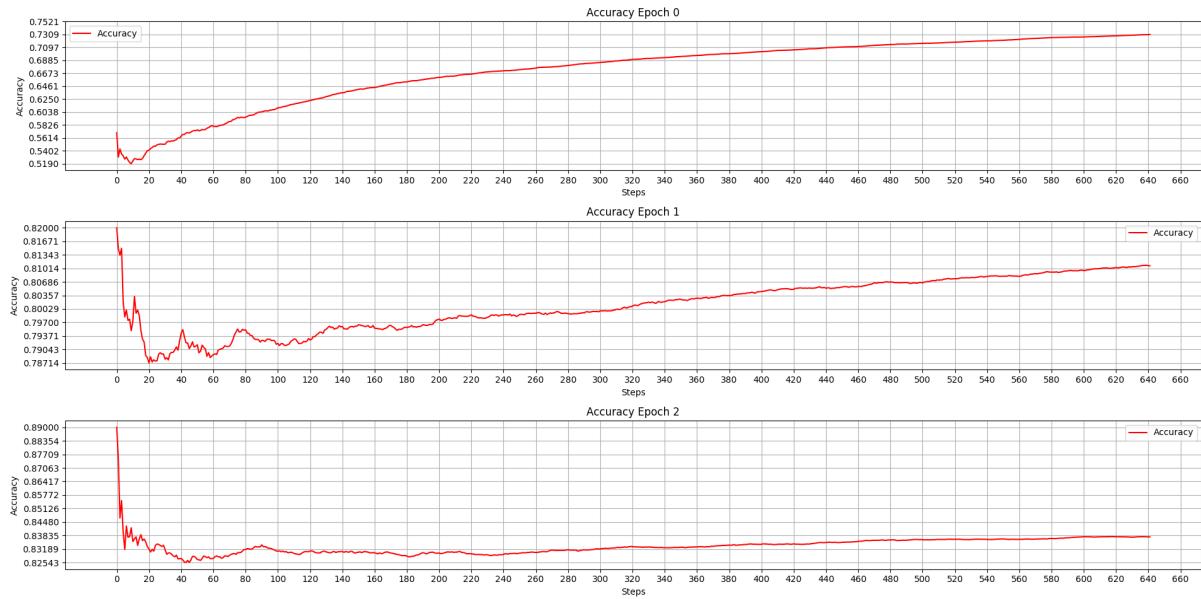


Figure 5.3: Minimum Accuracy. Parameters: LR=0.000001 ; BS=100

On Fig. 5.3 appears how accuracy changed on each epoch using the parameters that obtained the minimum value. Nevertheless, unlike the case of Inception V-3, this time accuracy shows a more common behaviour, meaning that values augment progressively and not just at random. This implies that the network has a more robust architecture to work with this types of information, referring to the residual block that allows the network to transfer more information on the learning process.

In the end, Fig. 5.4 shows how the model using the optimum parameters achieved the maximum accuracy after 3 epochs. Very similar to the previous case, on the first epoch happens the biggest changes. Finally, on epoch 2 and 3 accuracy begins to stabilize into an smaller range of values until the maximum was obtained.

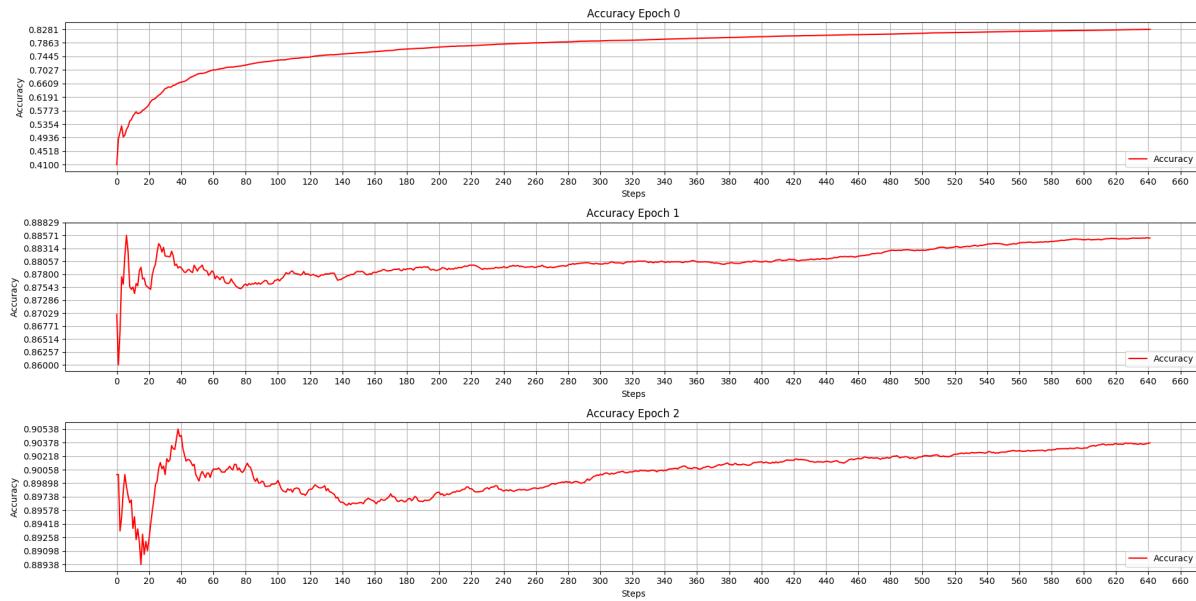


Figure 5.4: Maximum Accuracy. Parameters: LR=0.00001 ; BS=100

5.1.3 Experiment 3: FrankensNet

Based on Table 5.5, in the same way that the two previous cases were made, it can be seen that 3 experiments were performed first, each one with a different learning rate. Now, referring to the results obtained in this part, the minimum accuracy was achieved when using a learning rate of 0.00001. This happens to be a good result because, since this learning rate had the worst accuracy, it was not taken into account for the final training, thus training time gets reduced. On the other side, the maximum accuracy was obtained with a learning rate of 0.001, also being a good result because, being this learning rate the smaller of 3, it had the best capability to learn more appropriately, and the difference of accuracy achieved is quite great when compared with the minimum value.

Learning Rate	0.001	0.0001	0.00001
FrankensNet Accuracy	89.97%	89.03%	83.68%

Table 5.5: Experiments performed on FrankensNet with three different Learning Rates

From previous step, it was established that a learning rate of 0.001 happens to be the best option, in terms of accuracy, for our model. Now, on the second part of the experiment, the best batch size value for the model is being needed, of course also based on accuracy.

Very similar to the previous cases, it can be observed on Table 5.6 that three more experiments were performed. This time, all of them with the same learning rate of 0.001, but each one of them with a different batch size. Here, all the different values of accuracy obtained are very close to each other. Now, the minimum accuracy is achieved with a batch size of 100. This is a behaviour that was not expected because, as it could be

<i>Batch Size (with LR of 0.001)</i>	100	150	200
FrankensNet Accuracy	89.97%	90.23%	90.73%

Table 5.6: Experiments performed on FrankensNet with three different Batch Sizes

observed that both batch size winners on Inception V-3 [1] and ResNet-50 [2] correspond to 100 (the smaller of 3). Nevertheless, with FrankensNet is different, the greatest batch size obtained the best accuracy. On two cases, accuracy exceeded 90% and the lowest accuracy was also very close to this value.

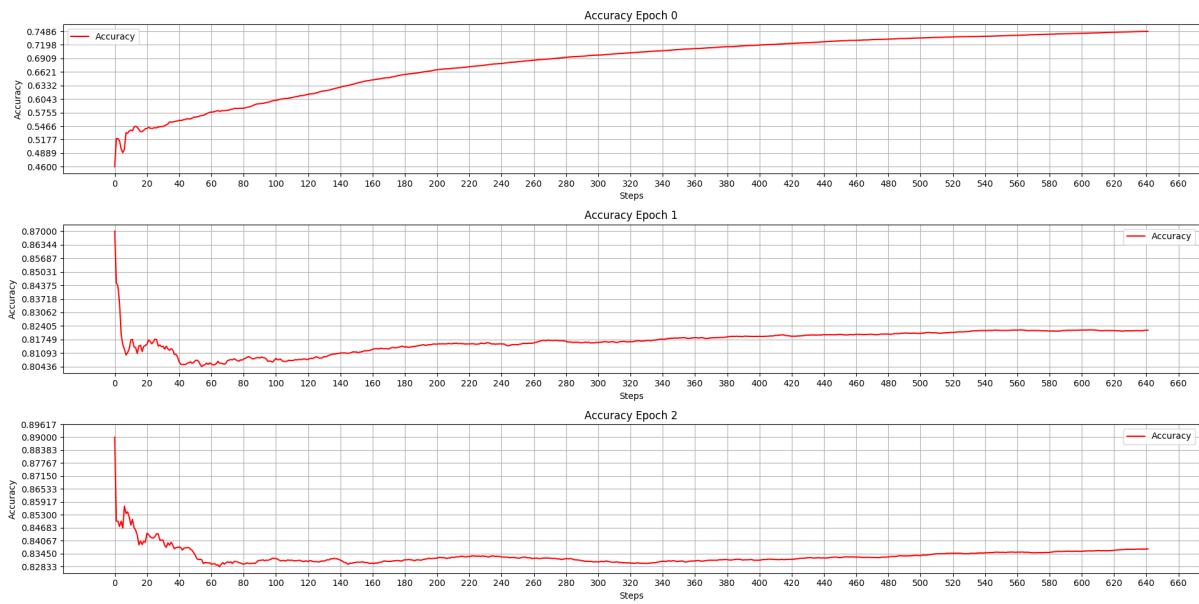


Figure 5.5: Minimum Accuracy. Parameters: LR=0.00001 ; BS=100

Fig. 5.5 shows how accuracy behaved throughout the entire experiment when the minimum value was achieved. These values are much more stable than the ones obtained in the case of Inception V-3 with the worst accuracy. Notwithstanding, the greatest changes happens only on the first epoch, on epoch 1 and 2, accuracy keeps augmenting but in a much smaller percentage.

Finally, on Fig. 5.6 can be observed the behaviour of accuracy on the experiment that obtained the maximum value in a total of three epochs. As in the worst accuracy case, values change widely only on first epoch. From here on, accuracy begins to stabilize in an smaller range of values. Nevertheless, on the last epoch of the experiment, there was a great descent on the first steps until accuracy started to improve again.

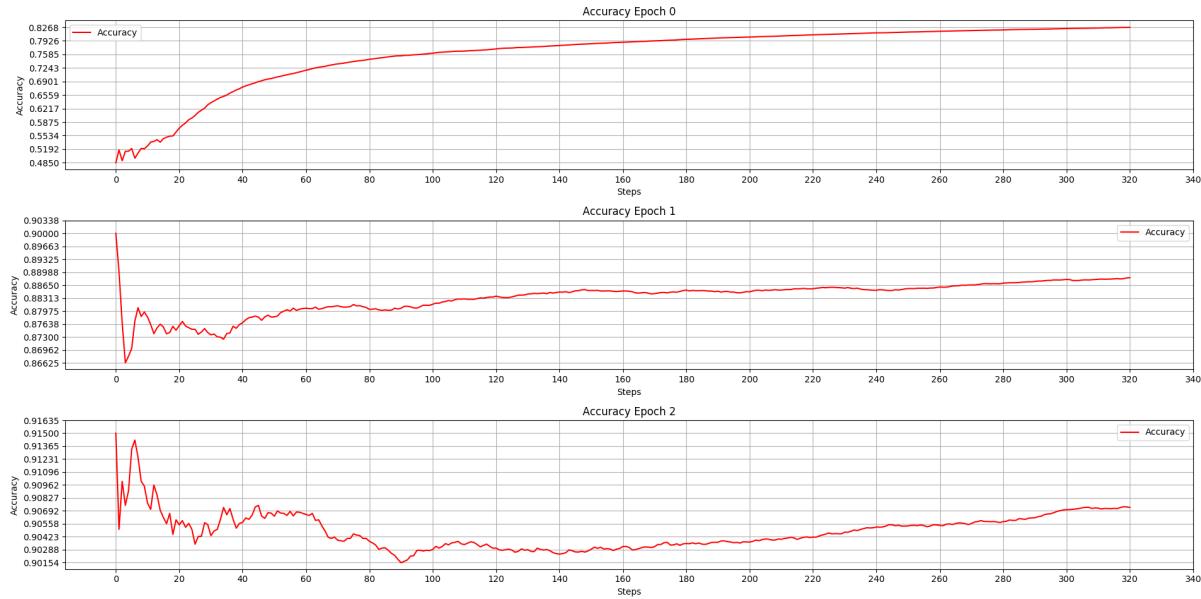


Figure 5.6: Maximum Accuracy. Parameters: LR=0.001 ; BS=200

In summary, gathering all the results obtained from the first three experiments on each model, the following bar charts are presented, the first one focused on the learning rate, and the second one on the batch size.

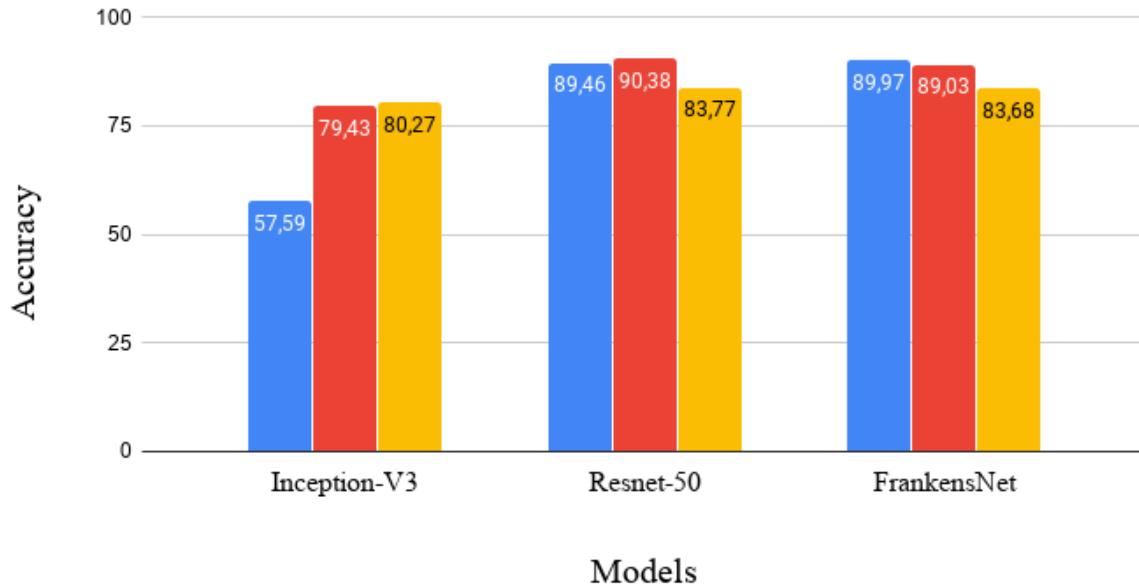


Figure 5.7: Results of learning rate experiments 1, 2, 3

Fig. 5.7 denotes the accuracy obtained by Inception V-3, ResNet-50 and FrankensNet

on each test related to the learning rate. In general, it is observed that the lowest accuracy value was obtained by Inception V-3 on the first experiment, while the greatest value achieved was given by the ResNet-50 model.

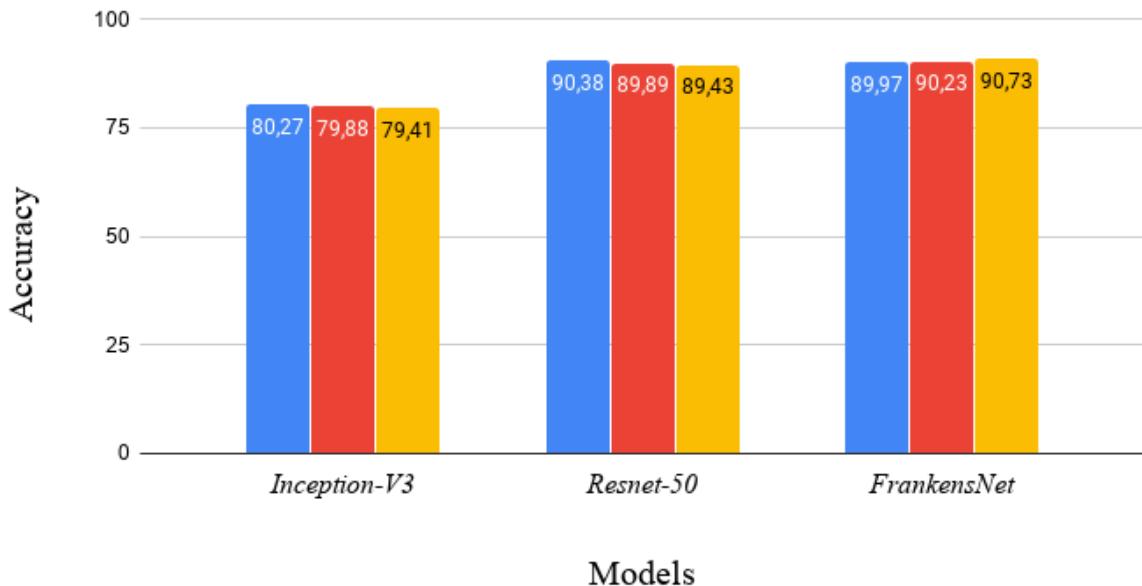


Figure 5.8: Results of batch size experiments 1, 2, 3

Then, on Fig. 5.8 appears now the accuracy obtained on the second part of the experiments, which is based on the batch size. Once again, the loser in this case was Inception V-3, achieving a maximum accuracy on test 1 with a 80.27%, but still, it is far away from the winner and the second place, where ResNet-50 surpassed a 90% accuracy, followed by FrankensNet very close to it with a 89.97%.

5.1.4 Experiment 4: Training Time / Execution Time

Training time allows to understand how much time does it take a network to learn properly how to solve a given task. This of course, in terms of accuracy

Model	Training Time	Execution Time	Final Accuracy
Inception V-3	1 day, 13:48:22	0.155579	84%
ResNet-50	2 days, 3:02:27	0.086877	91%
FrankensNet	5 days, 2:17:52	0.236781	92%

Table 5.7: Model's Training Time

Based on Table 5.7, it can be seen that Inception V-3 is the winner in terms of training time. Nevertheless, its accuracy is also the lowest one obtained among all models. On

the other side, ResNet-50 and FrankensNet achieved almost the same accuracy after two epochs, but with the difference that the time it took to train FrankensNet model doubles the time used for ResNet-50. This fact is important because depending on whether if the best accuracy is being wanted, or just to choose the model that takes less time to train, different options exist.

Now, based on the same table, the column of Execution Time can also be seen. It shows how much time it takes a network to classify a single frame. Here, ResNet-50 [2] becomes the absolute winner at the moment of classifying a frame in the shortest time, with only 0.08 seconds. The worst case scenario occurs with FrankensNet, taking the model almost three times than the one required by the winner. This is due to the fact that FrankensNet contains more layers, and therefore more operations to be performed, until all the information is gathered to obtain the final model. Inception V-3 [1] is at an intermediate point between the two models, using in average 0.16 seconds to classify a single frame.

5.1.5 Experiment 5: FP/FN rate, TP/TN rate

On table 5.8 can be observed all the values obtained from each model to test its efficiency. From a total of 61.012 frames used for this experiment, it seems that ResNet-50 [2] overcomes Inception V-3 and FrankensNet, in terms of True Positive results, and also achieving the lowest False Positive value.

<i>Model</i>	Inception V-3	ResNet-50	FrankensNet
<i>True Positive</i>	6578	14188	7691
<i>False Positive</i>	6391	3771	6043
<i>True Negative</i>	23061	26615	28367
<i>False Negative</i>	24982	16438	18911

Table 5.8: Models predictions

From the previous results obtained, and now based on the metrics used on section 4.3, the Table 5.9 shows the rates obtained from each classification type.

<i>Model</i>	Inception V-3	ResNet-50	FrankensNet
<i>TPR</i>	50%	79%	56%
<i>FPR</i>	50%	21%	44%
<i>TNR</i>	48%	68%	60%
<i>FNR</i>	52%	32%	40%

Table 5.9: Classification Rates

Here, it can be seen that ResNet-50 is the winner, achieving a 79% value of accuracy on True Positive and a 68% on True Negative. The worst case occurs on Inception V-3,

where it seems that the network is guessing the result with a value that is near or equal to 50%. Finally, not too far away from Inception V-3 classification accuracy, FrankensNet appears, where its True Positive rate slightly increases, but on the True Negative clearly overcomes the worst case by reaching a 60% on correct classification.

5.2 Additional Results

5.2.1 Overall results

First, lets analyze how accuracy behaved until reaching its top value on each model.

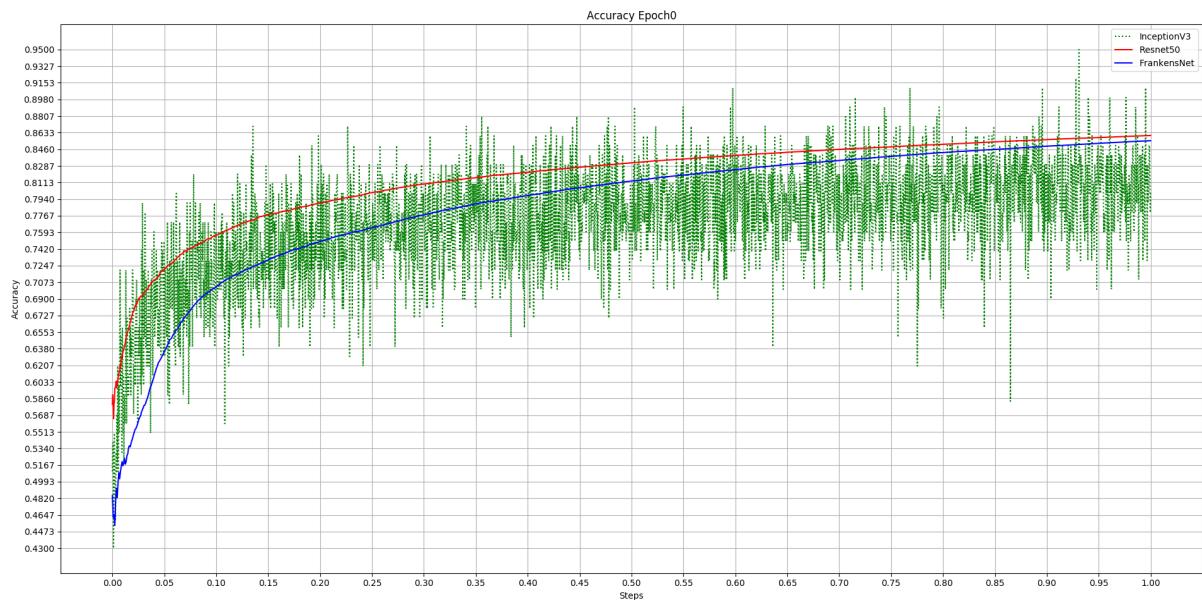


Figure 5.9: Model's Accuracy: Epoch 0

On Fig. 5.9 can be observed the model's accuracy on the first epoch. Something very interesting here is how unstable are Inception V-3 (green) accuracy changes when compared to ResNet-50 and FrankensNet. It clearly looks from the beginning how the last two models accuracy starts to stabilize and reducing its range of values, something that does not occur with Inception V-3 at any point given.

Then, on Fig. 5.10 this work shows the results obtained on accuracy on the final epoch. In this case, once again Inception V-3 appears not to have too much control on those accuracy changes, they are still too volatile. Nevertheless, FrankensNet and ResNet seem to still grow in an small course, but they both have their values stabilized without too many abrupt changes.

Now, lets focus on the loss obtained by each model. In practice, loss is expected to be minimized during the training of a given model. Here, the lower the loss value is,

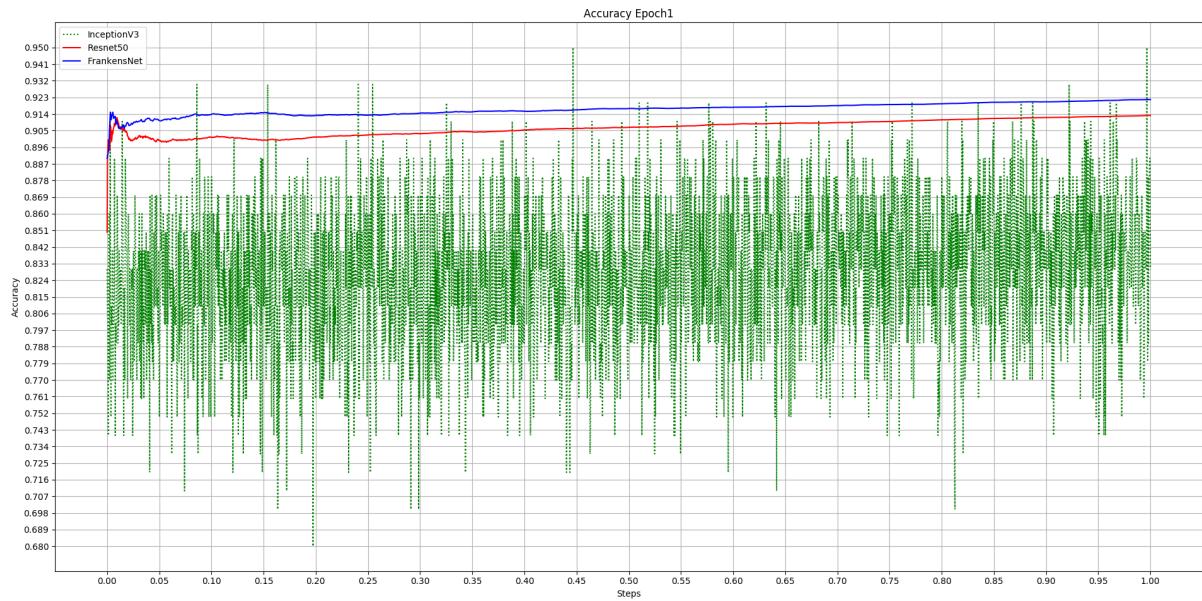


Figure 5.10: Model's Accuracy: Epoch 1

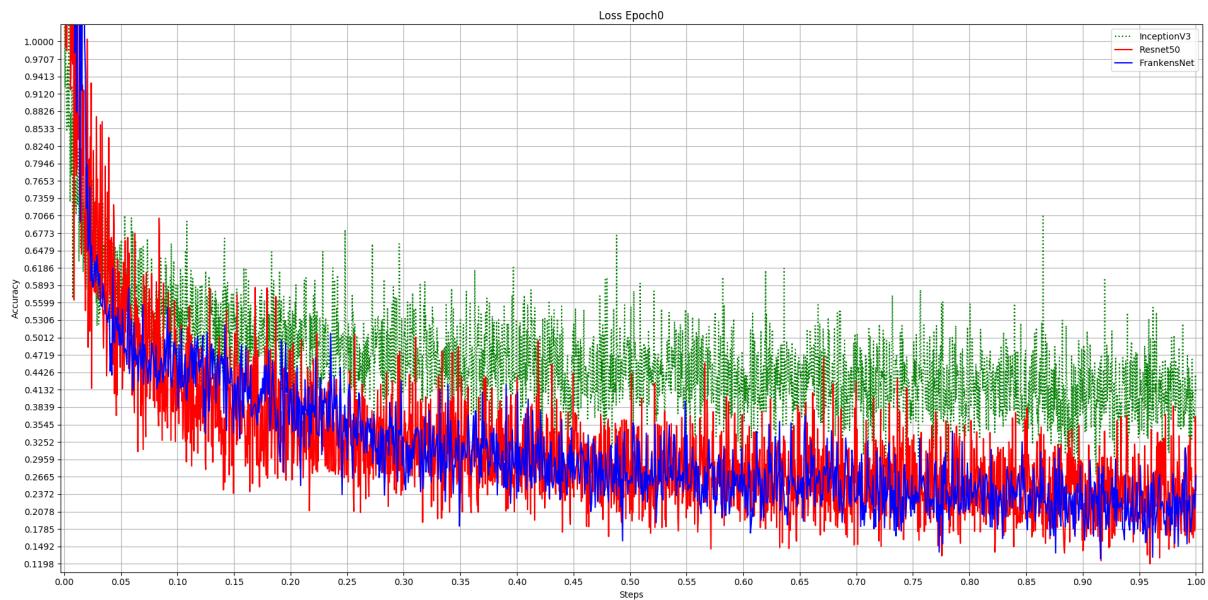


Figure 5.11: Model's Loss: Epoch 0

closer will be our predictions to true labels.

On this part, Fig. 5.12 shows the different Loss values obtained throughout the first epoch. It appears that the three models start from a point in common. Nonetheless, while the steps continue to increase, it starts to be noticed a big difference between Inception V-3 and the other models, and a slight difference between ResNet-50 and FrankensNet, but with better results.

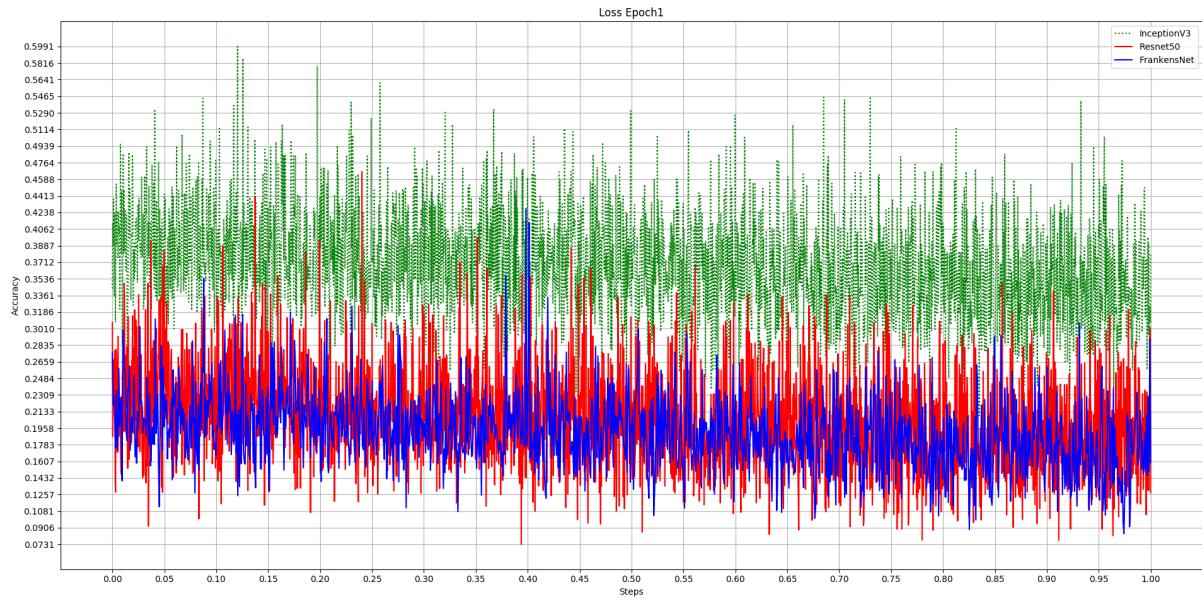


Figure 5.12: Model's Loss: Epoch 1

Finally, on the last epoch which can be seen on Fig. 5.12, it looks how Loss values are also stabilizing on a given range of values. The most important aspect here is how ResNet-50 and FrankensNet are still winning the race in terms of loss, achieving the lowest values when compared to Inception. This means that the predictions are expected to be highly accurate.

ROC curve allows to graphically represent sensibility (TPR) and specificity (FPR) of a binary classifier. with the values already obtained on section 5.1.5. ROC curve can be graphed.

Fig. 5.13 represents each model's ROC curve. It can be seen that ResNet-50 is the optimum choice to perform this task. Such results are very interesting, because ResNet-50 and FrankensNet achieved almost the same accuracy when training an specific model. Nevertheless, ROC curve shows the low predictive capacity of FrankensNet caused by the data size.

5.2.2 A Graphical UI for an Anomaly Detection Prototype

This part covers the development of a desktop program (written in python). The purpose of the program is to provide a user interface to test how each model works with anomaly detection tasks in real time. It works as follows:

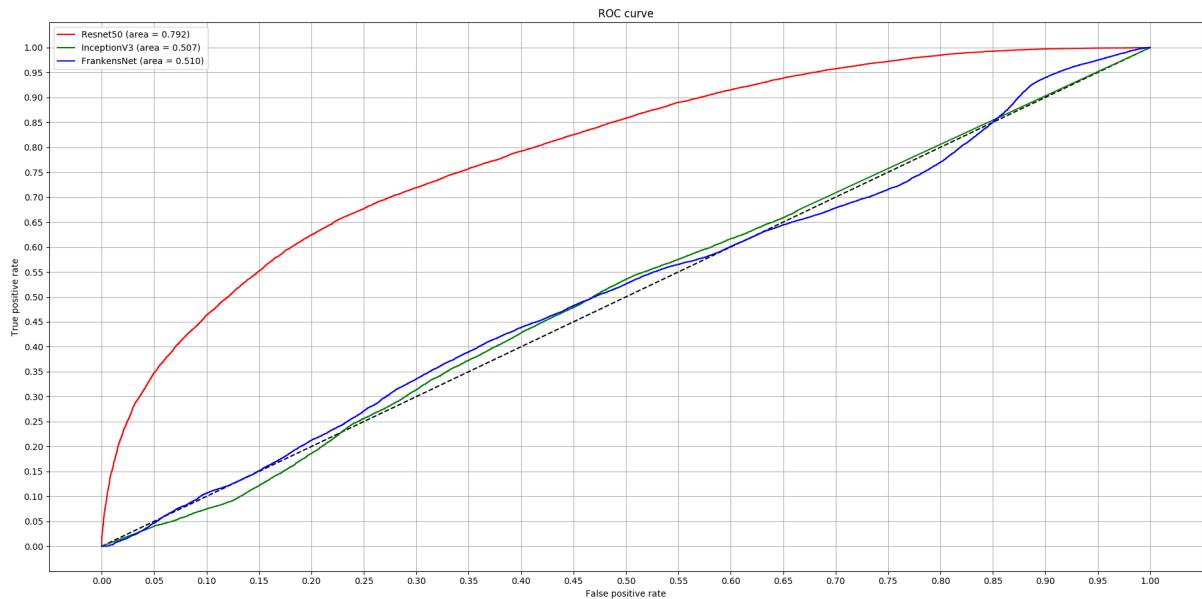


Figure 5.13: ROC curve

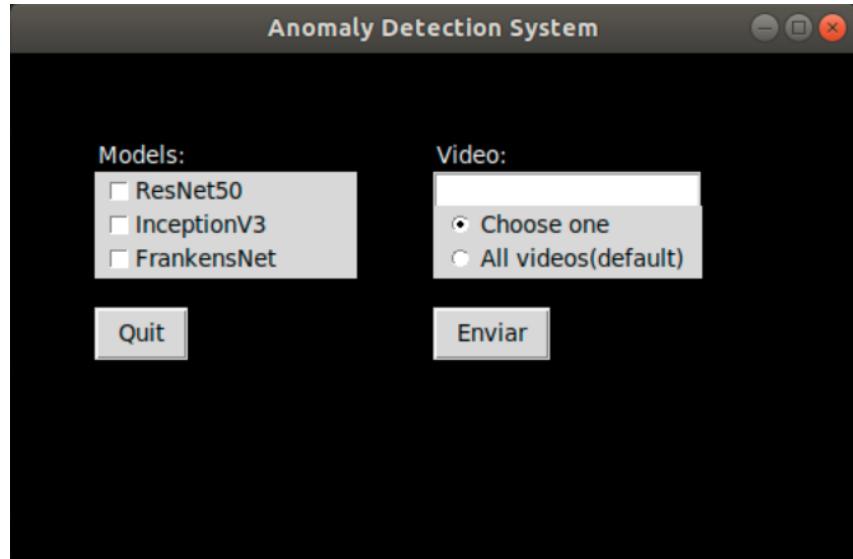


Figure 5.14: Anomalous Detection options

Fig. 5.14 illustrates the first window showed to the user. Here, the user can select the model to perform the anomaly detection, and also select if he wants to test a whole set of videos already established or select only one.

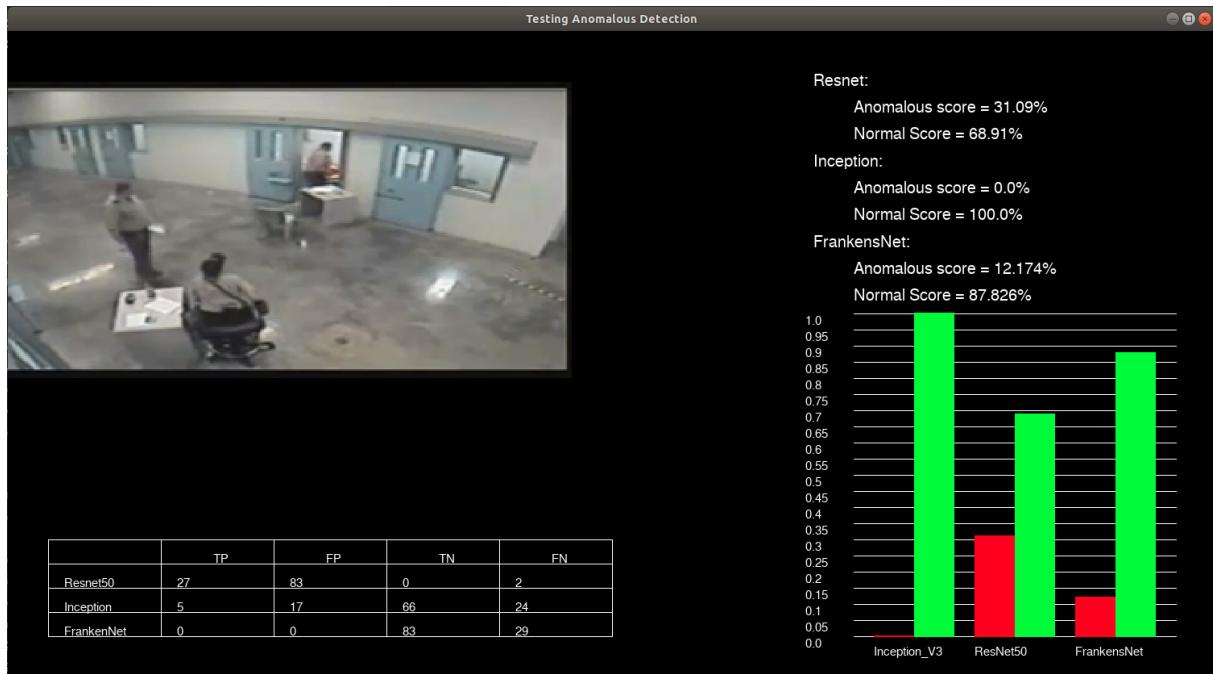


Figure 5.15: Anomalous Detection in real time

Once the model or models have been selected along with the videos to work with, a new window will appear. As it can be seen on Fig. 5.15, on top left screen's side appears the video scene to be classified. On the right appears the prediction values which each model achieves for anomalous and normal scenes. Finally, on left bottom are presented the different values obtained from each model referred to True Positive, True Negative, False Positive and False Negative. Here, it is important to mention that when only one model is being used to classify, it can be done in real time. Nevertheless, when more than 1 model is being used at the same time, the real-time detection feature will be lost.

In addition, a google site is developed with the complete information of this degree thesis. The information of this website is described in the appendix B.1.

Chapter 6

Conclusions and Future work

6.1 Conclusions

In general terms, this chapter presents a summary of every step needed to fulfill this work, starting from the literature review, followed by the implementation of different models, and finally the experiments done for improvement and evaluation purposes. At the end, this works shows an overall conclusion and future work in this area of interest.

- In the last decades, several deep learning approaches have been presented for video classification because of its successful reached in object recognition, image classification, pattern recognition, among others. However, those techniques still face some challenges in video classification due to a set of factors, such as capturing the spatial-temporal information adequately.
- After literature review, Inception V-3 [1], and ResNet-50 [2] were selected, later to be compared against our proposed FrankensNet. Such models were selected under a comparison study of accuracy, depth (number of layers) and number of parameters of each one. Its implementation was done by using 3 different tools: Python, Keras, and Tensorflow. The last 2 corresponds actually to Python deep learning libraries. Both of these libraries does not require too much time to invest on learning how to the, instead they were quite user friendly and uncomplicated to understand. All the networks were implemented using Python programming Language and its libraries. Due to the facilities that the libraries offer, there were no great difficulties to correctly define each model on its construction.
- A CNN requires a wide range of data to be trained and in many cases the data for specific applications are not available or if it exists, it does not contains labels. This problems become very time consuming. Particularly, for this project, the dataset used was carefully picked in order to adapt for anomaly detection application. In addition, it was augmented, improved, and labelled with the purpose of building a dataset appropriate for this work.

- CNN models have not been created with general purpose applications. Therefore, their parameters must be adjusted to work adequately with a specific task. In this case, for anomaly detection application, after tuning parameters, Inception V-3 demonstrated to be the model that requires the less amount of time when completing the given number of epochs in the shortest period. Behind Inception V-3 also appeared ResNet-50, almost doubling its training time when compared to the winner in terms of training time, but with the difference that its accuracy was considerably improved, surpassing 90%. In addition, the new model proposed in this research project (FrankensNet), happens to be the winner when testing the accuracy achieved, surpassing very closely to ResNet-50. Nevertheless, its training time was the longest between models.
- Besides, a program to provide a graphic user interface, created on python, was presented. Each trained model can be loaded into the program to start with video classification tasks. In addition, the option to select a specific video is also present in the program. Due to the fact that the program only uses python, It can be used on a different operating system than Ubuntu.

6.2 Future Work

In terms of accuracy, each network could be tested with many more learning rate and batch size values. Due to limited computational resources, there was not the capability of performing more of these experiments in order to improve accuracy. If in the future, such resources are enlarged, it would be of great help to check if accuracy keeps increasing as more tests related to parameter settings are performed.

In terms of the dataset used for this research project, it could be a great option to work with a dataset containing information of videosurveillance cameras, but located in Ecuador. This could result in obtaining a functional model and anomaly detection system with the capability to work with Ecuadorian environments, referring to places of the country itself. Thus, this would be of great help for different Government security systems.

6.3 Glossary

Here is presented a group of terms which are going to be found along the project.

- **Classification:** It is the set of categories (sub-populations) where a new observation belongs, on the basis of a training set of data containing observations.
- **Convolutional Kernel:** It consists of a small numerical matrix, mostly used in image processing for blurring, sharpening, embossing, edge detection, etc.
- **Convolutional Layer:** It uses convolutional kernels in order to extract features from input data through an activation map of that kernel.
- **Convolutional Neural Network:** It is a class of deep neural networks, in various cases is applied to analyzing visual imagery.
- **CPU:** It is known as the central processing unit, it is the computer component that is responsible for interpreting and executing most of the commands from the computer's hardware and software.
- **Dataset:** It is a collection of related sets of information that is composed of separated elements but can be manipulated as a unit by a computer.
- **Deep Learning:** It is a subset of machine learning in artificial intelligence that has networks capable of learning unsupervised from data that is unstructured or unlabeled.
- **Deep Neural Network:** It is a neural network with more than two layers.
- **Feature Extraction:** It is the generation of derived values coming from an initial dataset, intended to be informative to facilitate the subsequent learning and generalization steps.
- **Feature Map:** It is a matrix, or a set of matrices used for the mapping of where a certain kind of feature is found on the image which can be considered of interest for the Network.
- **Gaussian Mixture Model:** It is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters.
- **Graphics processing unit (GPU):** It is a computer chip that performs rapid mathematical calculations, primarily for the purpose of rendering images. Also widely used for deep learning purposes.
- **FLOPS:** Floating point operations per second, it is a measure of computer performance, useful in fields of scientific computations that require floating-point calculations.

- **ILSVRC:** ImageNet Large Scale Visual Recognition Competition. There is a competition every year.
- **Hyperparameters:** They are model-specific properties that are fixed before the network model starts to train.
- **Localization:** It consists of finding/detecting where the object of interest is and draw a box around it.
- **One Class SVM:** It is an unsupervised algorithm based on deep learning approaches which is commonly used for classification tasks.
- **Output Layer:** It refers to the final layer of the network which produces given outputs for the program for future interpretation.
- **Optimization:** It is a process to find an alternative with the most cost effective or highest achievable performance under the given constraints, by maximizing desired factors and minimizing undesired ones.
- **Over-Fit:** A modeling error which occurs when a function is too closely fit to a limited set of data points.
- **Parameter:** The parameters of a neural network are typically the weights of the connections. In this case, these parameters are learned during the training stage.
- **Post-processing:** Term used for quality improvement image processing methods.
- **Pre-processing:** It refers to the transformations applied to input data before feeding it to the algorithm. It is a technique used to convert the raw data into a clean data set.
- **Stride:** It corresponds to the number of steps that the filter will move each time on a given direction.
- **Testing dataset:** It is a dataset independent of the training dataset that follows the same probability distribution as the training dataset to validation purposes.
- **Training dataset:** It is a sample dataset used for learning purposes to fit the parameters (e.g., weights) of, for example, a classifier.
- **Validation dataset:** It is a dataset of examples used to tune the hyperparameters (i.e. the architecture) of a classifier.

References

- [1] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.
- [2] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [3] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, “Learning spatiotemporal features with 3d convolutional networks,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 4489–4497.
- [4] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.
- [5] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.
- [6] “Ucf-crime dataset (real-world anomalies detection in videos),” <https://webpages.uncc.edu/cchen62/dataset.html>, Jun. 2019.
- [7] S. Abu-El-Haija, N. Kothari, J. Lee, P. Natsev, G. Toderici, B. Varadarajan, and S. Vijayanarasimhan, “Youtube-8m: A large-scale video classification benchmark,” *arXiv preprint arXiv:1609.08675*, 2016.
- [8] W. Sultani, C. Chen, and M. Shah, “Real-world anomaly detection in surveillance videos,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6479–6488.
- [9] S. Ji, W. Xu, M. Yang, and K. Yu, “3d convolutional neural networks for human action recognition,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 1, pp. 221–231, 2012.
- [10] K. Soomro and A. R. Zamir, “Action recognition in realistic sports videos,” in *Computer vision in sports*. Springer, 2014, pp. 181–208.

- [11] A. Khaleghi and M. S. Moin, "Improved anomaly detection in surveillance videos based on a deep learning method," in *2018 8th Conference of AI & Robotics and 10th RoboCup Iranopen International Symposium (IRANOPEN)*. IEEE, 2018, pp. 73–81.
- [12] D. A. Jenks and J. R. Fuller, *Global Crime and Justice*. Routledge, 2016.
- [13] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.
- [14] D. C. Luvizon, H. Tabia, and D. Picard, "Learning features combination for human action recognition from skeleton sequences," *Pattern Recognition Letters*, vol. 99, pp. 13–20, 2017.
- [15] S. Nadimi and B. Bhanu, "Physical models for moving shadow and object detection in video," *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 8, pp. 1079–1087, 2004.
- [16] D. Fleury and A. Fleury, "Implementation of regional-cnn and ssd machine learning object detection architectures for the real time analysis of blood borne pathogens in dark field microscopy," *MDPI AG*, 2018.
- [17] G. Cheng, P. Zhou, and J. Han, "Learning rotation-invariant convolutional neural networks for object detection in vhr optical remote sensing images," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 54, no. 12, pp. 7405–7415, 2016.
- [18] M. Ribeiro, A. E. Lazzaretti, and H. S. Lopes, "A study of deep convolutional auto-encoders for anomaly detection in videos," *Pattern Recognition Letters*, vol. 105, pp. 13–22, 2018.
- [19] M. Sabokrou, M. Fayyaz, M. Fathy, Z. Moayed, and R. Klette, "Deep-anomaly: Fully convolutional neural network for fast anomaly detection in crowded scenes," *Computer Vision and Image Understanding*, vol. 172, pp. 88–97, 2018.
- [20] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proceedings of ICML workshop on unsupervised and transfer learning*, 2012, pp. 37–49.
- [21] M. Sabokrou, M. Fayyaz, M. Fathy, and R. Klette, "Deep-cascade: Cascading 3d deep neural networks for fast anomaly detection and localization in crowded scenes," *IEEE Transactions on Image Processing*, vol. 26, no. 4, pp. 1992–2004, 2017.
- [22] "Ucsd pedestrian dataset," <http://visal.cs.cityu.edu.hk/downloads/ucsdpeds-vids/>, Jul. 2019.
- [23] M. Bodén, "A guide to recurrent neural networks and backpropagation," 12 2001.
- [24] A. Sherstinsky, "Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network," *arXiv preprint arXiv:1808.03314*, 2018.

- [25] K. O’Shea and R. Nash, “An introduction to convolutional neural networks,” *ArXiv e-prints*, 11 2015.
- [26] S. Albawi, T. Abed Mohammed, and S. ALZAWI, “Understanding of a convolutional neural network,” 08 2017.
- [27] P. Arena, A. Basile, M. Bucolo, and L. Fortuna, “Image processing for medical diagnosis using cnn,” *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 497, no. 1, pp. 174–178, 2003.
- [28] A. Shustanov and P. Yakimov, “Cnn design for real-time traffic sign recognition,” *Procedia engineering*, vol. 201, pp. 718–725, 2017.
- [29] Q. Zhang, J. Xu, L. Xu, and H. Guo, “Deep convolutional neural networks for forest fire detection,” in *2016 International Forum on Management, Education and Information Technology Application*. Atlantis Press, 2016.
- [30] H. Jiang and E. Learned-Miller, “Face detection with the faster r-cnn,” in *2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017)*. IEEE, 2017, pp. 650–657.
- [31] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, “Imagenet large scale visual recognition challenge,” *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.
- [32] M. Ravanbakhsh, H. Mousavi, M. Rastegari, V. Murino, and L. S. Davis, “Action recognition with image based cnn features,” *arXiv preprint arXiv:1512.03980*, 2015.
- [33] M. Jian, S. Zhang, L. Wu, S. Zhang, X. Wang, and Y. He, “Deep key frame extraction for sport training,” *Neurocomputing*, vol. 328, pp. 147–156, 2019.
- [34] X. Yan, J. Han, and R. Afshar, “Clospan: Mining: Closed sequential patterns in large datasets,” in *Proceedings of the 2003 SIAM international conference on data mining*. SIAM, 2003, pp. 166–177.
- [35] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” *arXiv preprint arXiv:1511.06434*, 2015.
- [36] G. Lorena, G. Robinson, P. Stefania, C. Pasquale, B. Fabiano, and M. Franco, “Automatic microstructural classification with convolutional neural network,” in *Conference on Information Technologies and Communication of Ecuador*. Springer, 2018, pp. 170–181.
- [37] A. Iliev, N. Kyurkchiev, and S. Markov, “On the approximation of the step function by some sigmoid functions,” *Mathematics and Computers in Simulation*, vol. 133, pp. 223–234, 2017.

- [38] B. Karlik and A. V. Olgac, “Performance analysis of various activation functions in generalized mlp architectures of neural networks,” *International Journal of Artificial Intelligence and Expert Systems*, vol. 1, no. 4, pp. 111–122, 2011.
- [39] K. Hara, D. Saito, and H. Shouno, “Analysis of function of rectified linear unit used in deep learning,” in *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2015, pp. 1–8.
- [40] T. Wilson, D. Martinez, “The need for small learning rates on large problems,” in *IJCNN’01. International Joint Conference on Neural Networks*. IEEE, 2001, pp. 115–119.
- [41] P. M. Radiuk, “Impact of training set batch size on the performance of convolutional neural networks for diverse datasets,” *Information Technology and Management Science*, vol. 20, pp. 20–24, 2017.
- [42] L. Prechelt, “Early stopping-but when?” in *Neural Networks: Tricks of the Trade, this book is an outgrowth of a 1996 NIPS workshop*. Springer, 1998, pp. 55–69.
- [43] H. N. Mhaskar and T. Poggio, “Deep vs. shallow networks: An approximation theory perspective,” *Analysis and Applications*, vol. 14, no. 06, pp. 829–848, 2016.
- [44] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org. [Online]. Available: <https://www.tensorflow.org/>
- [45] F. Chollet *et al.*, “Keras,” <https://keras.io>, 2015.
- [46] Itseez, “Open source computer vision library,” <https://github.com/itseez/opencv>, 2015.
- [47] P. Shinners, “Pygame,” <http://pygame.org/>, 2011.
- [48] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, “Inception-v4, inception-resnet and the impact of residual connections on learning,” in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.
- [49] “Imagenet dataset,” <http://www.image-net.org/index>, Jun. 2019.
- [50] “Avenue dataset for abnormal event detection,” <http://www.cse.cuhk.edu.hk/leojia/projects/detectabnormal/dataset.html>, Jun. 2019.

Appendices

Appendix A

DataSets

A.1 Common Datasets Used for Image & Video Classification

Most of the existing Video Classification approaches pretrain and test their models using the following datasets:

- **ImageNet Dataset [49]:** ImageNet is an image dataset constructed with approximately 14197122 images belonging to nouns of the WordNet such as tree frog, banded gecko, agama, green lizard, sea snake, diamondback, tarantula, ptarmigan, African grey, etc.
- **UCSD Pedestrian Dataset [22]:** It contains video of pedestrians walkways on University of California San Diego (UCSD), taken from a stationary camera. This dataset consist on two main groups:
 - **UCSD Ped1:** This dataset consists on clips of groups of people walking towards and away from the camera.
 - **UCSD Ped2:** It contains scenes with people walking in a parallel way to the camera plane.
- **Avenue Dataset [50]:** It contains 16 training and 21 testing videos captured in The Chinese University of Hong Kong (CUHK) campus avenue with 30652 (15328 training, 15324 testing) frames in total.
- **Subway Benchmarks Dataset [19]:** It contains two sequences recorded at the entrance and exit of a subway station with a duration of 96 minutes and 43 minutes respectively.
- **UCF-Crime Dataset [6]:** It consists of 1900 real-world surveillance videos with 13 different anomalies, such as fighting, burglary, road accidents, assault, and so on. This dataset provides different environments form where extract information.

Appendix B

Algorithm Codes

B.1 Thesis Web Page

The whole work of this thesis is already uploaded in a web site for free access. It contains the full thesis, code of the anomaly detection system, training and validation scripts, results and other sources as articles and videos. The web site was developed in Google Sites and belongs to my tutor Lorena Guachi. This research work can be accessed at the following link.

<https://sites.google.com/site/degreethesislorenaguachi/2020-anomaly-detection-system-using-deep-learning-techniques>

B.2 FrankensNet Implementation Code

```
1
2 from __future__ import absolute_import
3 from __future__ import division
4 from __future__ import print_function
5
6 import os
7 from keras.applications.densenet import DenseNet201
8 from keras import layers
9 from keras import backend
10 from keras import models
11 from keras.regularizers import l2
12 from keras.applications import imagenet_utils
13
14
15
16 HEIGHT, WIDTH = 224, 224
17
18
```

```

19 # -----INCEPTIONV3-----
20 def conv2d_bn(x,
21             filters,
22             num_row,
23             num_col,
24             padding='same',
25             strides=(1, 1),
26             name=None):
27     """Utility function to apply conv + BN.
28
29     # Returns
30     Output tensor after applying
31     'Conv2D' and 'BatchNormalization'.
32     """
33
34     if name is not None:
35         bn_name = name + '_bn'
36         conv_name = name + '_conv'
37     else:
38         bn_name = None
39         conv_name = None
40     if backend.image_data_format() == 'channels_first':
41         bn_axis = 1
42     else:
43         bn_axis = 3
44     x = layers.Conv2D(
45         filters, (num_row, num_col),
46         strides=strides,
47         padding=padding,
48         use_bias=False,
49         name=conv_name)(x)
50     x = layers.BatchNormalization(axis=bn_axis,
51                                   scale=False,
52                                   name=bn_name)(x)
53     x = layers.Activation('relu', name=name)(x)
54     return x
55
56 def InceptionModel_B(x, i):
57     branch1x1 = conv2d_bn(x, 192, 1, 1)
58     branch7x7 = conv2d_bn(x, 128, 1, 1)
59     branch7x7 = conv2d_bn(branch7x7, 128, 1, 7)
60     branch7x7 = conv2d_bn(branch7x7, 192, 7, 1)
61     branch7x7dbl = conv2d_bn(x, 128, 1, 1)
62     branch7x7dbl = conv2d_bn(branch7x7dbl, 128, 7, 1)
63     branch7x7dbl = conv2d_bn(branch7x7dbl, 128, 1, 7)

```

```

63     branch7x7dbl = conv2d_bn(branch7x7dbl, 128, 7, 1)
64     branch7x7dbl = conv2d_bn(branch7x7dbl, 192, 1, 7)
65     branch_pool =
66     layers.AveragePooling2D((3, 3),
67                             strides=(1, 1),
68                             padding='same')(x)
69     branch_pool = conv2d_bn(branch_pool, 192, 1, 1)
70     x = layers.concatenate(
71         [branch1x1, branch7x7, branch7x7dbl,
72          branch_pool],
73         axis=3,
74         name='mixed_b_' + str(9 + i))
75     return x
76
77 def InceptionModel_C(x, i):
78     branch1x1 = conv2d_bn(x, 320, 1, 1)
79     branch3x3 = conv2d_bn(x, 384, 1, 1)
80     branch3x3_1 = conv2d_bn(branch3x3, 384, 1, 3)
81     branch3x3_2 = conv2d_bn(branch3x3, 384, 3, 1)
82     branch3x3 = layers.concatenate(
83         [branch3x3_1, branch3x3_2],
84         axis=3,
85         name='mixed9_' + str(i))
86     branch3x3dbl = conv2d_bn(x, 448, 1, 1)
87     branch3x3dbl = conv2d_bn(branch3x3dbl, 384, 3, 3)
88     branch3x3dbl_1 = conv2d_bn(branch3x3dbl, 384, 1, 3)
89     branch3x3dbl_2 = conv2d_bn(branch3x3dbl, 384, 3, 1)
90     branch3x3dbl = layers.concatenate(
91         [branch3x3dbl_1,
92          branch3x3dbl_2], axis=3)
93     branch_pool = layers.AveragePooling2D(
94         (3, 3), strides=(1, 1),
95         padding='same')(x)
96     branch_pool = conv2d_bn(branch_pool, 192, 1, 1)
97     x = layers.concatenate(
98         [branch1x1, branch3x3, branch3x3dbl,
99          branch_pool],
100        axis=3,
101        name='mixed_c_' + str(9 + i))
102    return x
103
104 def FrankensNet(input_shape=None, classes=2):
105
106     dense_model = DenseNet201(

```

```
107     include_top=False, weights='imagenet',
108     input_shape=input_shape)
109     for layer in dense_model.layers:
110         layer.trainable = False
111     x = dense_model.output
112
113     branch_1 = InceptionModel_B(x, 1)
114     branch_1 = InceptionModel_B(branch_1, 2)
115     branch_1 = InceptionModel_B(branch_1, 3)
116     branch_1 = InceptionModel_B(branch_1, 4)
117     branch_1 = layers.MaxPooling2D(name='max_pool_1')(branch_1)
118     branch_1 = InceptionModel_C(branch_1, 1)
119     branch_1 = InceptionModel_C(branch_1, 2)
120
121     branch_2 = InceptionModel_B(x, 5)
122     branch_2 = InceptionModel_B(branch_2, 6)
123     branch_2 = InceptionModel_B(branch_2, 7)
124     branch_2 = layers.MaxPooling2D(name='max_pool_2')(branch_2)
125     branch_2 = InceptionModel_C(branch_2, 3)
126     branch_2 = InceptionModel_C(branch_2, 4)
127     branch_2 = InceptionModel_C(branch_2, 5)
128
129     x1 = layers.Conv2D(256,3)(x)
130     x1 = layers.Conv2D(256,3)(x1)
131
132
133     branch_3 = InceptionModel_B(x, 8)
134     branch_3 = InceptionModel_B(branch_3, 9)
135     branch_3 = layers.MaxPooling2D(name='max_pool_3')(branch_3)
136     branch_3 = InceptionModel_C(branch_3, 6)
137     branch_3 = InceptionModel_C(branch_3, 7)
138     branch_3 = InceptionModel_C(branch_3, 8)
139     branch_3 = InceptionModel_C(branch_3, 9)
140
141     branches1 = layers.concatenate([branch_1,
142                                     branch_2, branch_3,x1])
143
144
145
146     x3 = layers.MaxPooling2D(name='max_pool_6')(branches1)
147     x2 = layers.AveragePooling2D(name='apf')(branches1)
148     x = layers.concatenate([x2, x3])
149     x = layers.Flatten()(x)
```

```

151     x = layers.Dense(1024, activation='relu', name='fc1')(x)
152     x = layers.Dropout(0.5)(x)
153
154     predictions = layers.Dense(2, activation='softmax',
155                                 name='predictions')(x)
156
157     model = models.Model(inputs=dense_model.input, outputs=[
158                               predictions], name='frankensnet')
159
160     return model

```

B.3 Training Code

```

1    ###
2    ## Parameters Values ##
3
4    TRAIN_DIR = Train Paths
5    VALIDATION_DIR = Validation Paths
6    BATCH_SIZE = #According to needed
7    HEIGHT = 224
8    WIDTH = 224
9    NUM_EPOCHS = 5 # In general
10   class_list = ["anomalous", "normal"]
11   FC_LAYERS = [2048,1000]
12   dropout = 0.5
13   LEARNING_RATE = #According to needed
14
15   ###
16   ## Function to save models information ##
17
18   def GuardarEpocas(history, model_name):
19       try:
20           acc = history.history['accuracy']
21       except:
22           acc = history.history['acc']
23       try:
24           val_acc = history.history['val_accuracy']
25       except:
26           val_acc = history.history['val_acc']
27       loss = history.history['loss']
28       val_loss = history.history['val_loss']
29       print(filepath_epoch)
30       file =

```

```

30     open(f "{filepath_epoch}/{model_name}_{datetime.now()}.txt"
31     , "w")
32     for a, va, l, vl in zip(acc, val_acc, loss, val_loss):
33         print(a, va, l, vl)
34         file.write((str(a)+"\t"+str(va)+"\t"+str(l)
35             +"\t"+str(vl)+"\n"))
36     file.close()
37
38 ### Train and Validation Data Generator ####
39
40 train_datagen = ImageDataGenerator(
41     preprocessing_function=preprocess_input,
42     rotation_range=90,
43     horizontal_flip=True,
44     vertical_flip=True
45 )
46
47 validation_datagen = ImageDataGenerator(
48     preprocessing_function=preprocess_input,
49     rotation_range=90,
50     horizontal_flip=True,
51     vertical_flip=True
52 )
53
54 train_generator =
55 train_datagen.flow_from_directory(TRAIN_DIR,
56                                     target_size=(
57                                         HEIGHT, WIDTH),
58                                     batch_size=BATCH_SIZE)
59 validation_generator =
60 validation_datagen.flow_from_directory(VALIDATION_DIR,
61                                     target_size=(
62                                         HEIGHT, WIDTH),
63                                     batch_size=BATCH_SIZE)
64
65 ### Training start with fine tuned model ####
66
67 history = finetune_model.fit_generator(
68     train_generator,
69     epochs=NUM_EPOCHS,
70     workers=8,
71     steps_per_epoch=num_train_images // BATCH_SIZE,
72     shuffle=True, callbacks=callbacks_list,
73     validation_data=validation_generator,
```

```

74     validation_steps=num_validation_images//BATCH_SIZE
75 )
76 GuardarEpochs(history , "Models Name")

```

B.4 Rotate Operation Code

```

1
2 def rotate(self , image , angle=90 , scale=1.0):
3     if angle < 0 and False:
4         angle = 360 + angle
5
6     w = image.shape[1]
7     h = image.shape[0]
8     # rotate matrix
9     M = cv2.getRotationMatrix2D((w/2 , h/2) , angle , scale)
10    # rotate
11    image = cv2.warpAffine(image , M , (w , h))
12    return image

```

B.5 Crop Operation Code

```

1
2 def crop_image(self , image , y1=0 , y2=223 , x1=0 , x2=223):
3     # print(image.shape)
4     image = image[y1:y2 , x1:x2]
5     # print(image.shape)
6     image = cv2.resize(image , (224 , 224))
7     # print(image.shape)
8     return image

```

B.6 Contrast Limited Adaptive Histogram Equalization Code

```

1 def CLAHE(self , img , gridsize=8):
2
3     lab = cv2.cvtColor(img , cv2.COLOR_BGR2LAB)
4
5     lab_planes = cv2.split(lab)
6

```

```
7     clahe = cv2.createCLAHE(
8         clipLimit=2.0, tileSize=(gridsize, gridsize))
9
10    lab_planes[0] = clahe.apply(lab_planes[0])
11
12    lab = cv2.merge(lab_planes)
13
14    return cv2.cvtColor(lab, cv2.COLOR_LAB2BGR)
```

B.7 Median Filter & Unsharp Masking

```
1
2 def sharpen_image(self, img, sigma, strength):
3     # Image Median filtering
4     img_mf = median_filter(img, sigma)
5
6     # Laplacian Calculation
7     lap = cv2.Laplacian(img_mf, cv2.CV_64F)
8
9     # Sharpen the Image
10    sharp = img - strength * lap
11
12    # Saturate Pixels
13
14    sharp[sharp > 255] = 255
15    sharp[sharp < 0] = 0
16
17    return sharp
```

