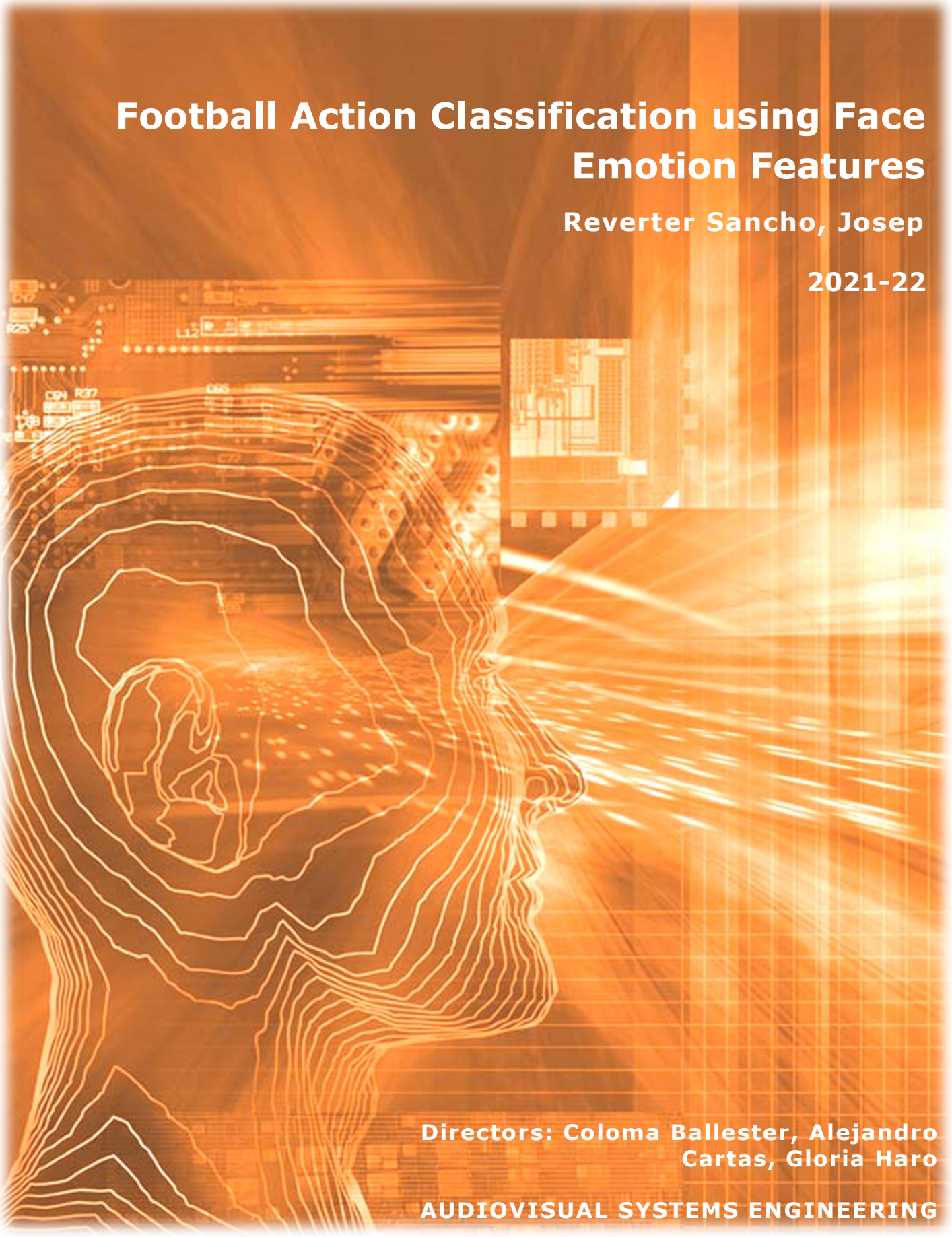


Football Action Classification using Face Emotion Features

Reverter Sancho, Josep

2021-22

Directors: Coloma Ballester, Alejandro Cartas, Gloria Haro

AUDIOVISUAL SYSTEMS ENGINEERING

upf. Universitat Pompeu Fabra Barcelona
Escola d'Enginyeria

Treball de Fi de Grau

FOOTBALL ACTION CLASSIFICATION USING FACE EMOTION FEATURES

UNDERGRADUATE THESIS

Josep Reverter Sancho

Directors: Coloma Ballester, Alejando Cartas, Gloria Haro

Audiovisual Systems Engineering

2021-2022

Universitat Pompeu Fabra Barcelona
Escola d'Enginyeria

A la meua família, en especial, als meus pares i les meues iaies.

Agreements

First, I would like to thank my supervisor Alejandro for sharing his knowledge that has helped me to implement the practical part and to write the project, and for the time he has dedicated on solving my doubts. Also, to my other two supervisors Coloma and Gloria, for advising and mentoring me throughout this time.

I want to thank all the people who have given me the knowledge that has made this work possible.

Finally, I would like to thank my parents for their effort in giving me the best education and for always supporting me.

Abstract

Football is evolving very fast thanks to the introduction of new technologies. Statistics, Artificial Intelligence, and Computer Vision have many applications in sports in general, and football is no exception. These technologies can be used for two very different purposes: for tactical purposes and on-field situations, or live broadcasting of actions. For example, to provide live annotations or automatic summaries. To contribute to this last-mentioned area, we have decided to make an action classifier. As there are already some studies on Action Classification considering on-field situations, we decided to approach the problem differently and look at the facial emotions of fans and players. To do this, we implement a face detector, extract emotional features from these faces, and create an action classification model based on them.

Resum

El Futbol està evolucionant molt ràpid gràcies a la introducció de les noves tecnologies. L'Estadística, la Intel·ligència Artificial i la Visió per Computador tenen moltes aplicacions als esports en general, i el futbol no és una excepció. Aquestes tecnologies poden ser utilitzades per dos objectius diferents: per finalitats tàctiques i situacions al joc, o per la comunicació i la retransmissió de les accions. Per exemple, proveir anotacions en directe o resums automàtics. Per contribuir en aquest àrea, hem decidit fer una classificador d'accions. Com ja hi ha alguns estudis sobre la classificació d'accions considerant situacions del terreny de joc, hem volgut encarar el problema de diferent manera i utilitzar les emocions de les cares del públic i dels jugadors. Per fer-ho, hem implementat un detector de cares, extret informació característica de les respectives emocions, i creat un model que donada aquesta informació sobre un període de temps, ens classifiqui l'acció.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

There is no need to explain Football since it is the most popular and most played sport in the world. It is played by more than 20 million people in more than 120 countries [21], and it is by far the most followed sport, with around 4 billion followers worldwide [43]. Although there are studies and research that Football is much older, the first Football Association, the "English Football Association", was created in London in 1863 [18]. Since then, it has become much more than a sport for many people, especially for the world economy. Over the years it has evolved a lot, not only because it has become more and more professionalized, but also thanks to the introduction of new technologies. Statistics, Artificial Intelligence, and Computer Vision are extensively being used more and more into this sport for tactical purposes, on-field situations [4], and to communicate to its wide audience what is happening or has happened in the field. For example, one of the clearest applications of detecting the action that is happening is to broadcast in live written annotations and highlights of the match, or generate automatic summaries containing the highlights and the important actions. Around this objective, many studies have been based on detecting the different actions throughout a match. However, it is not an easy task, since there are many types of actions, some even open to interpretation by the observer, and in some cases, they can happen at the same time.

1.1 Motivation

Soccer is growing a lot technologically and the great advantages that it would have to make a good classification of actions are the biggest reasons why this research has been done. The objective is to classify what real action of soccer matches takes place. Most of research is made using features extracted from the football field, such as the position of the players and the ball. What we decided and what

would make us innovators was to change the point of view of these studies and to be able to detect what happens in the match without looking at the pitch but looking at the emotions of the players and fans.

To detect emotions, we decided to look at faces, as this is what gives the clearest visual information about the emotion a person is feeling. Therefore, the main objective of our research is to classify football actions using (fans and players) face emotion features.

1.2 Objectives

The main objective is to create a model that classifies the actions of soccer matches considering the emotions of fans and players. In order to reach out this final goal we have the following set of milestones.

- Determine an adequate face detector for the correct and accurate detection of faces.
- Implement an efficient emotion feature extractor from faces.
- Develop an action classification model that takes as input the extracted emotion features.

1.3 Overview

In *Chapter 1* we describe the basis of our project, the motivation that has led us to do it and the objectives we have set, as well as a small overview of the project. *Chapter 2* provides some of the main theoretical concepts that are used throughout the project. In *Chapter 3* are described the previous studies related to our topic. *Chapter 4* describes the proposed methodology to achieve the objectives, and explains the Database and models used. At the beginning of this chapter, a diagram of the chapter itself can be seen, listing the most important parts. In *Chapter 5* we present the results of our model, using different parameters and classes, where we describe which are the best and why. Finally, in *Chapter 6* we explain the conclusions we have reached, comparing them with the objectives proposed at the beginning, and we present ideas to improve our model in the future.

Chapter 2

THEORETICAL BACKGROUND

This chapter describes 4 important concepts to understand how the implemented models work.

- A **Neural Network (NN)** is a very simplified model of how our brain works, which simulates how our brain processes information. The basic units are neurons, which are organized in layers. Each neuron has an activation function, which, given an input, will determine its output. Normally all the neurons between layers are connected to each other, and each connection has a different weight. It is this weight what we have to estimate and optimize to get a good result, and the process of determining the right weight is called training. Typically, Neural Networks consist of 3 types of layers: input, output and hidden layers. The input layer receives a series of input values, called features. The hidden layers are the ones that have the interconnected neurons and each respective weights, and the output layer will give us a classification or a prediction for the given features. In supervised learning, the training process of the network is as follows: given a series of features with their respective classification or prediction already known, and the initial weights between connections, the features are introduced and propagated among the layers of the network *(Forward Propagation)*, giving a result in the output layer that will be different from the real one. Then, the predicted and the real result are given to a *loss function*, which tells how different the predicted result is from the real result. In addition, a *gradient descent* is performed on this function, going back layer by layer, in order to find the weights that give a smaller loss *(Back Propagation)*. Finally, the weights are updated, and the process is repeated with all the examples as many times as necessary to obtain the best possible prediction or classification.

- A **Convolutional Neural Network (CNN)** is based on the same idea as a Neural Network, but in this case applied to the use of images as initial features. Its main objective is to assign the importance (using weights) of the different aspects of the image. They do this by using Poolings and Convolutions of the initial image by different filters, reducing the size of the image into an easier shape to process. Then, it converts the features that are obtained after a certain number of convolutions (which according to our model is the important part that defines our image) into a vector of a certain length. This vector will be the input of the last part of the CNN, a Neural Network that takes care of classifying it. The training also has a methodology similar to the previous one: it compares the predicted classification with what it really is and obtains a loss function, where a gradient descent is made for each layer and the new weights are obtained, for the last layers and for the convolutional ones. Finally, the weights are updated and the process is repeated as many times as necessary to obtain a good classification.

- A **Feature Pyramid Networks (FPN)** are commonly used to extract features in small objects. Its methodology is based on a *bottom-up* and *top-down* pathway. The first one behaves as a Convolutional Neural Network to extract features, and as we have described in the previous point, it decreases the resolution as the convolutions are done, but the semantic value gets bigger as we go through the layers. Then, on the *top-down* pathway, the path is the other way around. From the high semanthic layer, higher resolution layers are reconstructed, in order to obtain rich semantic layers with a high resolution. To make this reconstruction more accurate, it uses information that he has previously stored while doing the *bottom-up* pathway [12].

- **Deformable Convolutional Networks (DCN)** [5] are based on Convolutional Neaural Networks, but new modules are added to enhance CNN's potential. These new modules are a *Deformable Convolution* and *Deformable RoI Pooling*. A *Deformable Convolution* is a standard convolution suffering a deformation in the resulting sampling grid, conditioned on the input features. *Roi Pooling* is a type of pooling that converts an input region with variable size into a fixed size feature. Hence, a *Deformable RoI Pooling* is the same as a *RoI Pooling* but adding an offset in the resultant sampling grid [5].

Chapter 3

STATE OF THE ART

Face detection and Face analysis are tasks that have been at the center of many studies since the beginning of Computer Vision [26] because they can be used in many domains and with many purposes, such as surveillance and compliance, authentication and identification, or even in medical diagnoses [23, 40]. Over the years, researchers have done several Computer Vision studies in sports, too [42, 13, 45]. Especially in football (the sport we are working on in this thesis), there are many objectives and purposes to achieve with Computer vision techniques. For example, player tracking [13] and ball localization [10] using object detection methods to approximate where the players are placed accurately and obtain their positional data, which is a piece of beneficial information to, for example, observe and study the movements of the rival team.

On the other hand, research has also been done in detecting what happens in the field of play, not with tactical objectives, but to be able to detect actions and classify them to generate automatic annotations and statistics, live written retransmissions, and automatic summaries at the end of the match. These are the purposes for which we use Action Classification, which consists of determining what action takes place over a period of time [42].

Since our project is focused on action classification using emotions features from faces, we briefly described face detection, emotion classification, and their use in action recognition.

3.1 Face Detection

Face detection is the process of detecting and locating human faces in an image, without storing extra information about it (such as facial attributes). It is a fundamental part and the first to be considered in many face processing, recognition, and more models, such as predict the age or the emotion of the face. Any algorithm that works with faces needs good and reliable face detection.

In the last few years, there has been a lot of research on face detection, and therefore, it has been implemented in many different ways, such as knowledge-based methods, based on a set of rules and attributes (defined by humans) that a face is supposed to have; template matching methods, which is based on predefined templates of faces; feature-based methods, which uses techniques such as the Histogram of Oriented Gradients (HOG) [6], and appearance-based methods, where there are trainable models such as Neural Networks [29].

Nowadays, the most typically used models for optimal and reliable face detection (and recognition) are the ones based on Deep Learning. Specifically, the most popular ones are SSD [16], MTCNN [44], and RetinaFace [9][35]. However, other classical methods are used as they are less computational demanding, such as OpenCV [30].

Single Shot Detector (SSD) uses a simple Deep Neural Network and uses multi-scale feature maps and convolutional predictors as features, which makes it another very optimal option to use in terms of computational speed.
Multitask Cascaded Convolutional Networks (MTCNN) is based on a cascaded architecture with three convolutional networks. It is more complex than SSD and OpenCV, but unlike the others, their outputs are detailed facial landmarks (mouth, nose and eyes coordinates) and not simply bounding boxes.
RetinaFace [9, 22] is the most robust state-of-the-art face detector. Dense Neural Networks are used, based on five levels feature pyramids and a multi-task loss. It is the best model if we want to detect faces in images where many people appear. For example, in a picture with 1000 people, it can detect 900 faces setting a score threshold up to 0.5, which makes it a perfect model for use in events with many spectators. We will explain how it works in more detail later on.

3.2 Face Emotion Recognition

After a good detection, we can get detailed information about the face, such as Face Verification, based on verify if faces of two pictures belong to the same person; Face Recognition [34], being able to recognize who is in the photo (doing the verification process many times), predict the person's age and gender [32], or predict the emotion is experiencing [31], that is the part we have to study in more depth, and there are many studies on this subject.

For example, *Li et al.* [17] did a very interesting study about the facial expression recognition, taking advantage of the Kaggle Facial Expression Recognition Challenge. They used the data from the Kaggel Challenge, consisting of 30k examples classified in 7 different emotions (anger, dis- gust, fear, happiness, sadness, surprise, and neutral). The authors implemented three different classifiers, using Convolutional Neural Networks (CNN). The first one has only one convolutional and one fully connected layer (with softmax as the activation function), and it is used to define a baseline. The second one is a bit more complex. A CNN with 5 convolutional layers and a fully connected one is implemented. They use ReLU as a activation function and a batch normalization after all the layers, and between the third and fourth layers, and the fifth and fully connected, a max pooling and dropout is implemented. Finally, a deeper Convolutional Neural Network is proposed. It is very similar to the previous one, but with more fully connected layers and parameterizable layer depths and filter sizes, that provide a better capability to learn [17]. The best result that they achieved was a 48 $\%$ of test accuracy, and it was reached by the deeper network.

Shi et al. [36] also developed a very interesting approach about facial expression recognition, where they introduce a new architecture named Amending Representation Module (ARM) that will replace the pooling layer. It consists on three blocks, and the architecture is divided into two different paths: one for the feature map (extracted with a ResNet-18) and the other for the entire batch. The first block is called Feature Arrangement, and it is used to amplify the function of the second, that is the one that defines the weights by Means of Convolution. Finally, the last block is to share the information between the two paths.

Finally, another interesting face emotion classifier is the one that DeepFace [35] provides to us. It also uses a Convolutional Neural Network (CNN) with only 3 layers, and it also tries to predict the emotion between 7 classes. We will not explain it in depth now, because this model will become more prominent later.

3.3 Face detection and emotion analysis for Action Classification

Neural Networks give us many options to create a good action classifier, for example, most methods use Recurrent Neural Networks, Convolutional Neural Networks, or Deep Neural Networks. However, there are models also give a lot of importance to the pooling layers applied to the extracted features [42]. Hence, most of the action classification methods we will study will be based on Deep Learning.

A very interesting investigation for us is the one that is done in [42]. Where video features are used to spot and classify the different actions. They also extract the video features using the *SoccerNet* dataset, but they only focus on 3 soccer events. Each feature is a representation of each frame and extracted using ResNet-152.

In [11], the classification task is done using shallow pooling Networks, trying different pooling techniques, such as max pooling, average pooling, NetVLAD, and NetRVLAD among others [42].
However, the model that they suggest and the best performance one is based on a pooling layer after extracting the features (this pooling layer uses NetVLAD with many clusters), followed by a fully-connected layer with a sigmoid as the activation function [42].

The closest related work to ours is presented in [20]. They create a model to detect the highlights of tennis and golf matches, considering the reaction of the player, the audio of the spectators, the comments of the commentator, and sometimes (when it is available) text-based metadata and game analytics.
For the player reaction, they trained an action recognizer to detect when a player is celebrating or not. This classifier is applied in every frame and is based on VGG-16 and ResNet architectures, with a fine-tuning to detect celebrations.
Facial expressions were categorized into four types: aggressive, tense, smiling, and neutral, and they classified these expressions by fine-tuning a VGG-face model.
Once they obtain the scores of every classifier (audio-based detection, commentator excitement detector, player reaction, facial expression, and sometimes TV graphics and annotations), they apply a max operation within the defined time window, and a weighted sum between the scores is done to obtain a general score. Finally, they decide whether it is a highlight or not depending on whether the overall score is high or not.

Chapter 4

PROPOSED METHODOLOGY

As it is shown in *Figure 4.1*, our proposed methodology consists of 4 main parts. This is a general outline of what we do, and it gives us a clear view of the important sections, showing what we have before and what we obtain after each part.

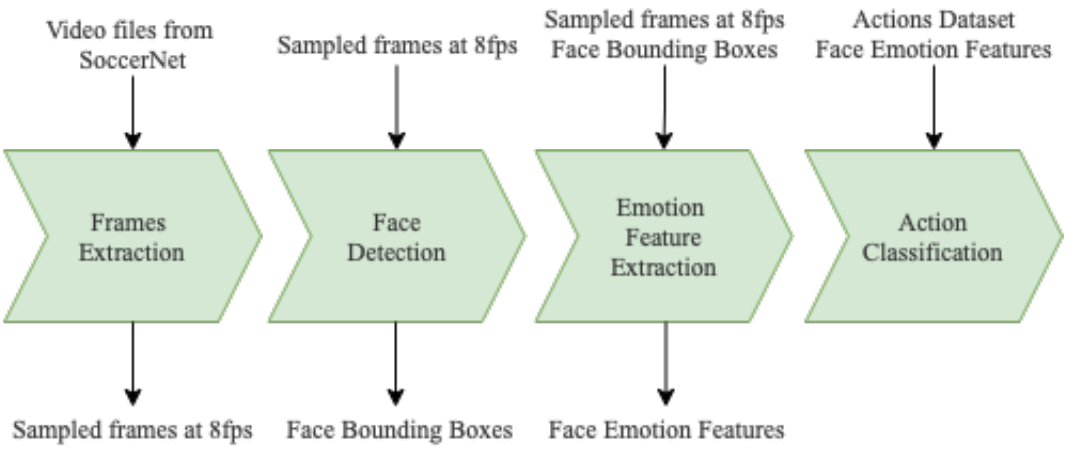


Figure 4.1: Parts of the Proposed Methodology

4.1 SoccerNet Dataset

The first step to create our action classifier is to determine the dataset to use and what information it provides us with. In this section, we explain why we have decided to use the SoccerNet-v2 Dataset [11], which advantages, and which type of information gives us.

There are a wide variety of datasets for action classification, constructed from images and videos coming from movies, cooking, and sports in general. Nevertheless, there are very few that provide us with good visual football data for action classification tasks [11]. The most famous datasets are SoccerNet [11], SoccerNet-v2 [7], and SoccerDB [14], that is based on the SoccerNet dataset, and in terms of quality, quantity, and reliability of the information, these are the best ones available.

Although SoccerDB is a good candidate to use in action classification projects because it provides us with 346 matches, from which 37,709 annotations have been tagged, we decided to use SoccerNet-v2. Soccernet-v2 is the enhanced version of SoccerNet. Both make available 500 soccer matches (a total of 764 hours of duration), from the 5 major European leagues plus the UEFA Champions League matches, which were played in the seasons between 2014 and 2017.
The first version of this dataset had a total of 6,637 annotations, with 3 distinct classes: Goal, Yellow Card, Red Card, and Substitution. The second version has up to 300,000 annotations among all matches, which give us information on 17 different classes, as listed in *Figure 4.2*.

As expected, not all these classes are repeated with the same frequency throughout the 500 games, as many actions (such as Ball out of play, shots on/off-target, or Throw-in) are much more common than others (such as penalty, red and yellow card), as can be seen in *Figure 4.2*. This is a fact that we consider when training the classifier model later in the last part of the methodology.

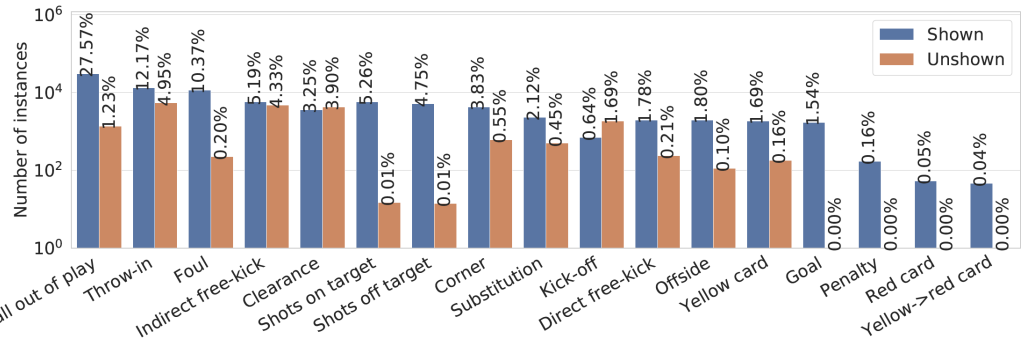


Figure 4.2: SocceNet annotations distribution [11]

In addition, SoccerNet offers action and replay images, replay grounding labels, calibration data, and re-identification data [38], but the information we are interested is the labels list, to obtain a detailed list of all the matches annotations. We obtain one list per match, with a list of dictionaries where each dictionary is one annotation containing the gametime (half and minute where the match happens), label (to which class does the action belong), position (time in milliseconds when the action happens), team (to which team does the action refer, if it is home or away), and visibility (if the action is shown or not).

4.2 Part 1: Frames Extraction

The goal is to extract the frames of each video efficiently and with an adequate sampling frequency to be able to extract quality features.
On one hand, the higher the sampling rate, the more likely it is to have faces with good quality and sharpness, to facilitate the process of detecting faces and extracting the emotional features correctly and reliably. On the other hand, the higher the sampling frequency, the more frames we will get per game, and therefore, the more time it will take to extract the features from all the frames. Therefore, we must find a suitable frame rate, which meets our quality objectives, but without extracting too much information and slowing down the feature extraction process.

Finally, after observing the results of sampling the games with different frequencies, we decided to use a sampling frequency of 8 frames per second. With this frequency, we obtain frames of sufficient quality to see clear faces in most of the frames where they appear. Some are still blurred, but this is an unavoidable constraint. With this, we will also get much more data per face. The more frames per second we use, the more information per face we will have.

In *Table 4.1* we can see a comparison between the quality of the images of the faces between sampling at 2 frames per second and 8. On the left, we can see the two frames sampled at 2 fps in a specific second. On the right we see 2 frames of the 8 that have been sampled in the same second. As it is observed, the ones on the right have a better image quality, which gives us a higher level of detail.

In soccer, each half lasts 45 minutes plus extra time, which is different in each half and in each match. Thus, for each half, we get at least 21,600 frames (45min * 60 s/min * 8frames/s = 21600 frames).

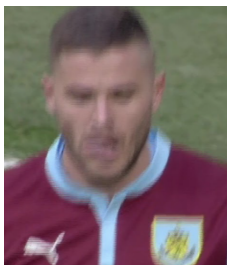
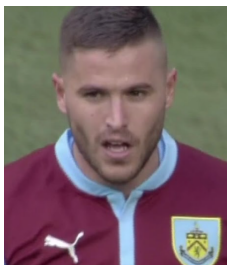
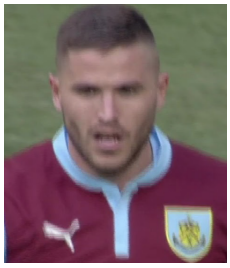
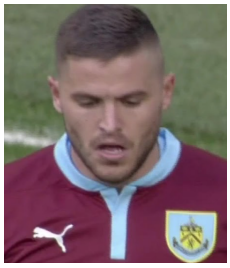
2 FPS	8 FPS
	
	

Table 4.1: Quality comparison between sampling at 2 and 8 frames per second.

4.3 Part 2: Face Detection

Once we have prepared the dataset with the annotations and their respective labels as described in the previous section, our priority is to use a face detector that performs a rigorous detection, and that fails as few times as possible, since this detection will affect the rest of the methodology and results.

In this section, we will decide which face detector to use (observing how they behave in different frames of our dataset) and we will explain in detail how it works. We will also explain the considerations we must take into account, and the conclusions we reached after detecting the faces.

4.3.1 Face Detectors Comparison

As we have seen briefly in the state-of-the-art section, we can use different face detectors, such as OpenCV [30], RetinaFace [9, 22], SSD [16], or MTCNN [44]. As we know, OpenCV and SSD are the most efficient in terms of optimization and time, but they are not as reliable and accurate in detecting. That is, they are more likely to detect faces where there are none, or not detect when they are not very clearly visible.
In contrast, MTCNN and RetinaFace are the most robust and reliable, but they are deeper models (they have a more dense architecture, with more layers and more calculations to be made) and take longer to process.

For our final objective, we think that the robustness and accuracy of the detector are the highest priority since it depends on it to get good emotional features, and if the face is wrongly detected, we could have very confusing and erroneous features, and it is something that would negatively impact our model.

To make the decision, we have compared their performance in several frames from different actions of our dataset. *Figure 4.3* shows an example of the results obtained in this comparison. In this Figure, the frames of three different actions that contain faces are depicted.
For each action frame, we show the face detection results of using all the evaluated models from our comparison. Looking at the *Figure 4.3*, we can affirm what we said before about the robustness of every face detector. OpenCV is a good face detector, but it fails on many occasions, and in moments where there are many faces, it leaves many undetected.

Figure 4.3: Comparison between Face Detectors.

On the contrary, RetinaFace and MTCNN have not presented any false positives in any tested frame, and their accuracy is very similar, but the RetinaFace performance when there are many faces to detect is significatively better. That is the reason we have chosen it as the face detector that we are going to use.

4.3.2 RetinaFace

RetinaFace is a robust single-stage face detector, which performs pixel-wise face localization using extra-supervised and self-supervised multi-task learning. In other words, RetinaFace provides very accurate face detection and localization carrying out both regression and classification tasks at the same time in both branches, and it extracts different information features for every face [9].

The learning of the model is based on four main tasks: face classification, face box regression, facial landmarks regression, and dense face regression. The output of the model provides us with the result of every task, such as a face score (probability of being a face), a face box (frame coordinates of the box surrounding the face), five facial landmarks (two for the eyes, one for the nose and two for mouth), and 3D vertices of the face projected on the plane [9].

The authors apply a multi-task loss with four main parts, one part for each task of the model (face classification loss, face box regression loss, facial landmark regression loss, and dense regression loss). As we can see in Equation1, the losses of each part are added and each one is multiplied by a learning factor, to determine the importance of each part in the learning process [9].

The model is trained using the WIDER FACE Dataset, with more than 30k images and almost 400k bounding boxes, with high randomness in terms of expression, occlusions, position, and rotation. In addition, they have annotated facial landmarks in more than 100k faces, including faces with a very poor resolution, to help the model to detect faces even when the resolution is very poor [9].

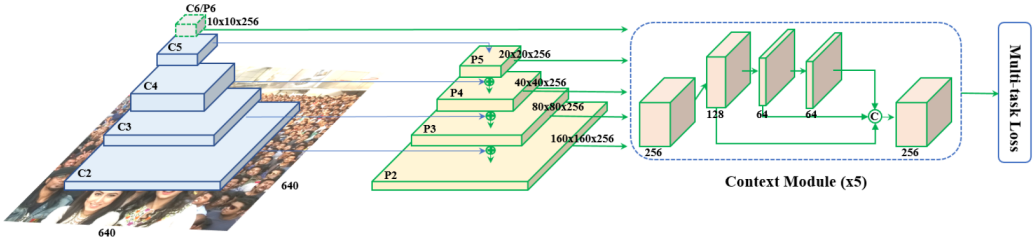


Figure 4.4: RetinaFace Architecture [9]

As it is shown in *Figure 4.4*, the RetinaFace model is a Combination of a Feature Pyramid Network (FPN) with a Deformable Convolutional Network block (explained in *Chapter 2*), and they called it Context Module. The FPN consists of 5 levels, where the bottom-up pathway is computed with a pre-trained network for classification (ResNet-152), and the top-down levels are calculated depending on the output of the previous level and the residual of the ResNet. Then, for each of the 5 levels of the FPN, they create independent Context Modules, and each module is a Deformable Convolutional Network (DCN) [9].

4.3.3 Model Implementation

The objective of our code is that given a list of match halves to process, extract the information of all the faces we are interested in, together with the frame to which they belong. In other words, we will obtain a dictionary of frames, where each frame will be a dictionary of the faces detected in that frame, and each face will have the information extracted from the RetinaFace model, which is another dictionary. The information we are interested in is the bounding box of the face, but we will also keep the other information (score and landmarks) in case we are interested in the future.

To speed up the detection process, we only try to detect faces in the frames that are around some action. To do this, we have to define how many frames we must consider if they belong to an action or not. That is, we must define how long an action lasts, or in other words, from the moment the action is annotated, how many seconds before and after are part of it.
The duration of the actions is called the time window parameter (T). In the state-of-the-art of action classification, [42] and [20], different time windows are used. In [20] they use a time window of 15-20 seconds, and in [42], they do a comparison between time windows of 20, 30, and 60 seconds.

Due to the large number of actions that we have per game, we think that the time windows of 30 and 60 seconds are too large as they could cause too much action overlapping. For example, and as we have seen observing the matches, after taking a Corner, many different actions can happen. For example a Goal, Shot off target, Shot on target, Clearance, or even another Corner. The corner action and one of these actions will overlap partially even if we use a Time Window of 10-15 seconds, but if we use a 30-60 seconds one, the actions will totally be overlapped. Also, with frames sampled at 8 frames per second, we would have too many frames inside the window, and the window size (W) would be too large.
Finally, we decided that the time window would be at most 20 seconds. This means that we consider that the action occurs at most from 10 seconds before to 10 seconds after the moment when the action is annotated. Hence, the window size (W) is at most 161 frames long (10s * 8fps before and after the action plus the central frame). For now, we are going to use this T, but we must keep in mind that we have decided to use this value to do the feature extraction, but in the future when we define and train the action classification model, there would be no problem if we change this parameter to fewer seconds.

To implement the model, we have implemented the RetinaFace code[1], which is a TensorFlow implementation of the original RetinaFace [9].
This library offers a function to detect the faces, that given a frame, gives you the information of the faces that appear, but it is not prepared to parallelize the process in batches of frames, that is, to process different frames at the same time. Hence, we have made a re-implementation of the code to suit our needs.

As we have described at the beginning of this section, the result of this part will be a dictionary of frames, where each frame will have a dictionary of faces with the information of each face, and if any face is not detected in a concrete frame, the frame will be an empty dictionary.

4.3.4 Section Results

Apart from obtaining the information of each face and the data we are interested in, which is the location of the face, we can obtain other information that will be useful for our experiments, such as how many faces we have per action, how many times each action appears during the match, and how the faces are distributed during the time window of the action. Hence, we have done an experimental analysis about these results.

In *Table A.1* we can see two types of graphs for 3 different matches.
The left graph is composed of 3 graphs that indicate: the number of total faces that are detected in the match depending on the action, the number of times that each action is repeated throughout the match, and the third graph is the division between these two, which shows the average number of detected faces each time the action appears. As we can see, the first two plots are more similar to each other than the third one, since the first two have much more faces than the other one.

The first conclusion that we can draw is that there are 5 classes that have much more faces than the others, which are *Ball out of play*, *Clearance*, *Throw-in*, *Indirect free-kick*, and *Foul*, and it is not surprising because these are the most repeated actions among the Dataset, as it is shown in *Table 4.2*.

We can also point out that, although there are actions that are not repeated too often, they have a high ratio of faces per appearance, such as *Substitution*, *Goal*, or *Corner*.

The plots on the right show the temporal distribution of the faces throughout the action. Each face detected in the action is assigned a relative position to the

[1]GitHub: https://github.com/serengil/retinaface

central frame, which is the number 0. For example, if a face is detected one second before the action is annotated, this face will have the value -8, because it is 8 frames (1 second sampled at 8 frames per second) before the central frame.
This plot provides us with information that can be used to decide the time window of the actions. As we can see, the average distribution of the faces throughout the action never exceeds ± 60 frames. This means that we might have to use a smaller T when training the model, to avoid outlier faces. We can also observe some interesting data from the actions, for example, in the goal action, most of the faces are detected after 5 seconds (40 frames) since the action is scored, which makes sense because it is when the players and fans are celebrating.

4.4 Part 3: Emotion Feature Extraction

After detecting faces in the frames that belong at least to one action, the next step of our proposed methodology is to perform the extraction of emotional features from them. These emotional features are the information that we are going to provide to our action classifier model. Therefore, making a good extraction of emotional features is also a very important part of our methodology, since it will depend on it whether our classifier learns correctly.

As we have seen in the state-of-the-art section, and as we can see if we do some research about good face emotional classifiers, there are many models that we can use, such as [17] and [36], but the easy implementation and simplicity of the model provided by DeepFace [35] , together with the good results it gives, makes us choose it to be our face emotional classifier.
Hence, the objective of this section is to describe DeepFace, to explain the model we have implemented in order to extract the emotional features, how we do this extraction and what kind of features we obtain.

4.4.1 DeepFace

One of the best libraries for facial recognition and analysis is DeepFace[2]. This is a very complete library that uses many state-of-the-art methods to do facial recognition, verification, and attribute analysis, such as VGG-Face [25], Facenet [28], OpenFace [1], Facebook DeepFace [39], DeepID [27], Dlib [8], and ArcFace [33]. In our case, we are not interested in recognition and verification, because the identities of the people on the scene would not give us information about what action is happening. In contrast, the face analysis module of this library is

[2]GitHub: https://github.com/serengil/deepface

very useful to our purposes. It predicts the age and gender of a person, using a classification model with 101 classes for age, and 2 classes for gender, applying transfer learning and a VGG-face model [25, 32], and a facial expression predictor [31], made with a Convolutional Neural Network (CNN) model that can classify the face expression in 7 different emotions.

The model we are most interested in is the DeepFace facial expression predictor [31]. This model has been trained with the Fec2013 Dataset [15]. It consists of 28k labeled images for training, and 3k for validation, classified into 7 emotions (anger, disgust, fear, joy, sadness, surprise, and neutral). As mentioned, this model is based on a CNN, implemented with TensorFlow and Keras. As it is observed in *Figure 4.5*, this CNN is based on 3 convolutional layers, with max-pooling and two average poolings, respectively. Then, they use a flatten layer, which converts the result of the last convolutional layer into a 128-dimension vector, and finally, three fully connected layers for the classification part. All layers (both convolutional and fully connected) use ReLU as the activation function, except the last layer of the model, which uses a SoftMax to perform the classification. The loss function used to train the model is the cross-entropy loss function, as commonly used in multi-class classification models [31].

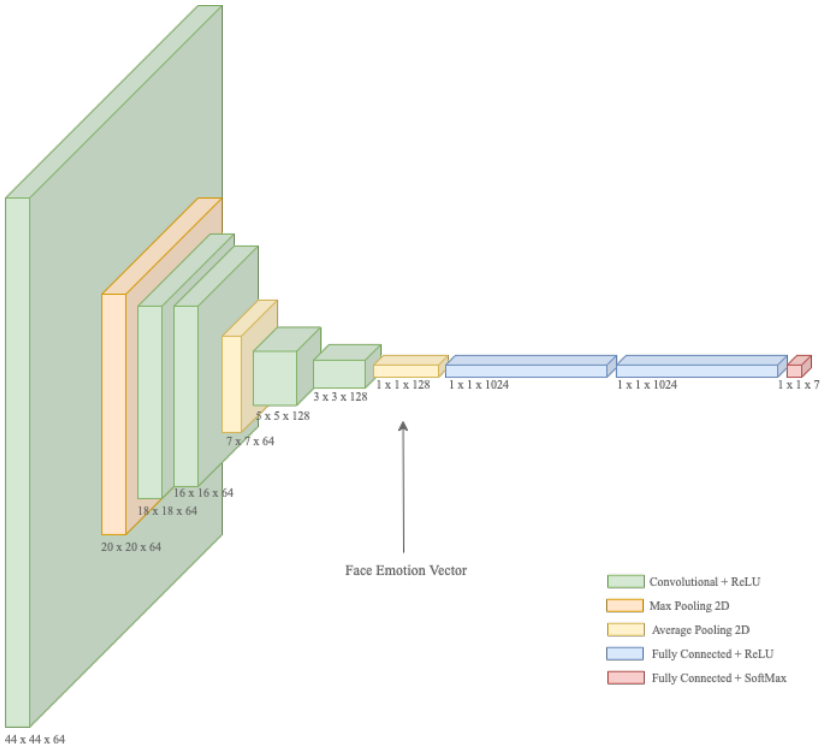


Figure 4.5: DeepFace Facial Expression Recognition architecture

4.4.2 Feature Extraction

As we have explained, this model is implemented as part of the DeepFace library and it predicts the probability that a face belongs to each of 7 different emotions. The main reason that the prediction is a probability rather than a hard classification is that a face can show multiple emotions at the same time. For example, a person might be feeling angered and sad simultaneously and express both feelings through his face. Therefore, using the output of the SoftMax layer as a face emotional feature for our classification model could be a good idea, because it can tell if a face displays one clear emotion or multiple emotions at the same moment.
However, we did not choose this option, because although this way we get what or which emotions each face belongs to, we are not interested in the emotion itself, we are interested in using emotion features to predict an action. We think that a vector of length 7 will not be useful to describe in detail the emotion of the face and that we would need a larger vector, to have more detailed information, and not something as generic as just the emotion probability vector.

Finally, the information that we use to define the emotion of the face, and therefore, the one that our action classifier uses as a feature, is the output vector of the flatten layer that has a 128 dimension. This vector may contain more latent information about the emotion than the prediction itself. This layer is located between the convolutional layers and the fully connected layers of classification, as we can see in *Figure 4.5*. We consider that since this vector has enough information to distinguish between emotions through a network of fully connected layers, it can also be used as input to a neural network that predicts actions given an arbitrary number of these vectors, depending on the faces that we detect in each action.

As stated in the DeepFace paper [31], the DeepFace model works better if we already pass to it detected faces instead of whole frames. That is why, using the bounding box of each face extracted earlier in Part 1, we pass to the model the trimmed faces of each frame and not the whole frame.

After this, we obtain a dictionary with similar information as in Section 4.3.3. This dictionary contains the emotion vector of each face that appears in all the frames of a soccer match. Additionally, we also store the size of the bounding box of each face. It is information that can be useful in the future, in case we want to discard the smaller faces, or if we want to give more weight to the larger ones.
4.4.1

4.5 Part 4: Action Classification

Once we have extracted the facial emotion features of the matches, the last part of our research is to develop the action classifier.

This model and the rest of the code has been developed in Python. Although the Neural Networks we described in the previous two sections are made with a TensorFlow framework, we have implemented the model in PyTorch, which is another Python framework created for Neural Networks models, with a lot of stability, support, and information available.

In this section we explain the proposed model, the training process, and present the problems we have found during training, and their solutions.

Let us recall that for each match, we have a vector that defines the emotion of each face, which appears in at least one frame located in at least one action. As we know, each action consists of a fixed number of frames, depending on the window size (W), each frame with a variable number of faces, and each of these faces has a facial emotion vector. Hence, the input of our model will be all the facial emotion vectors of all the faces that appear in a specific action, and the objective will be to classify to which action they belong.

4.5.1 Model Scheme

As we can see in *Figure 4.6*, our model consists of two initial pooling layers, followed by two fully connected layers. The pooling layers are intended to summarize the content of the initial features and end up having only one vector per action, and the fully connected layers are the ones in charge of the classification.

As we have described before, each action contains a fixed number of frames that will depend on the window size (W). Each frame has a variable number of faces, and each face has a face emotion vector of length 128.

The first pooling is implemented with the objective of obtaining a single vector of length 128 that summarizes the content of the emotions of all the faces of a frame, having a variable number of face emotion vectors. If a frame has no faces, we assign it a vector with values very close to zero. With that, we avoid the variability of the size of our model. Therefore, what we obtain from the first pooling is W vectors of length 128, one for each frame.

Given a face emotion vector per frame, the objective of the second pooling is to obtain an emotion vector per action, which summarizes the emotions of the

whole action. That is, to go from W vectors per action to only one.

These two pooling layers can be *average* or *max pooling*. We have decided to use average pooling since if we use max pooling, the first layer will be more likely to collect erroneous vectors. For example, if we have four happy faces and one sad face in a frame, and we use average pooling, that sad face will not affect much in our resulting vector. In fact, we are interested that it affects our vector (because it has important information about the frame) but that it does not have too much effect. On the contrary, if we use the max pooling, that sad face can affect a lot in our resultant vector, since if the features that the sad face has are bigger than the others, we are going to keep the ones of the sad face. This does not interest us, since it would give a big weight to the face that in this case is an exception.

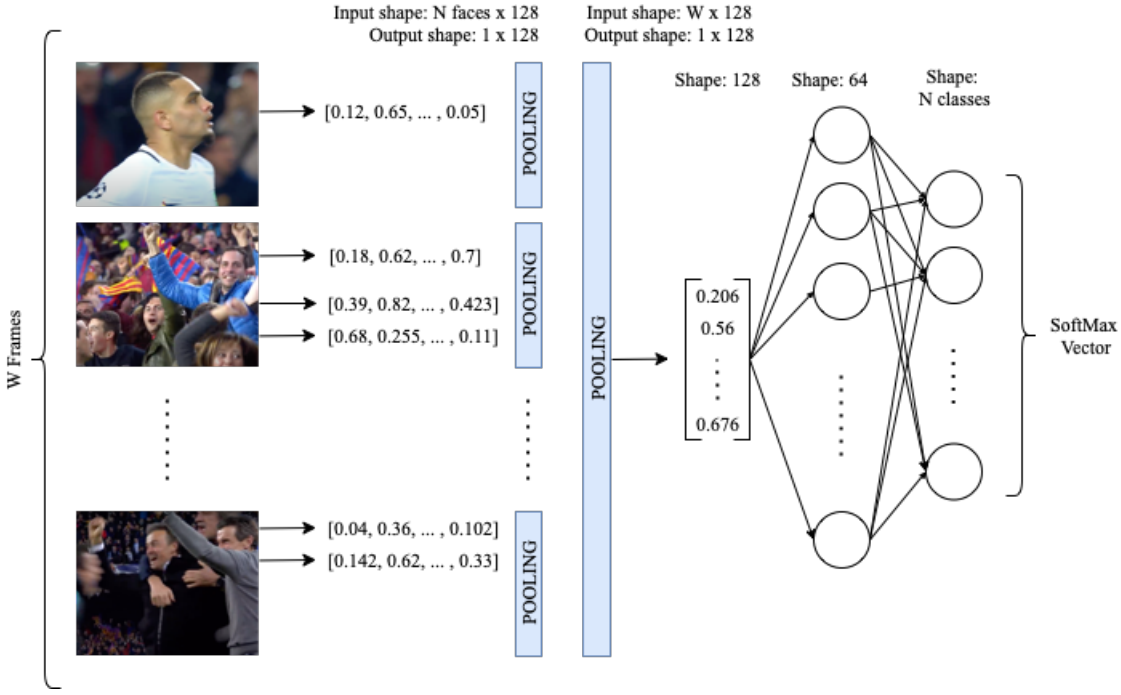


Figure 4.6: Diagram of our proposed model

As mentioned above, the fully connected layers are in charge of classifying the average vector of the action resulting from the pooling layers. Therefore, they will be the layers that will learn during the training of the model. As we are implementing a classification model, the activation of the second layer is a SoftMax, which will determine the probability that the action belongs to each of the 17 possible actions.

4.5.2 Dataset Preprocessing

Due to the high computational cost of detecting faces with the RetinaFace model, we have not been able to process the entire SoccerNet dataset but we have processed 135 complete matches. These matches are split in training, validation and test sets considering that the actions in the different matches have the same distribution. Although we have not processed so many matches compared to all those that SoccerNet provides, we have a fairly large number of actions to train our model: 23,097 for training, 5,552 for validation and 4,042 for test.

Before training the model, we have to preprocess the dataset to eliminate the information we do not need, eliminate erroneous data or data that may cause confusion, for example:

- Frames that have no faces: 60% (949,104/1,562,545) of the training set frames that belong to unless on action are faceless. As we have indicated above, when a frame has no faces, it is assigned a vector of numbers very close to zero as the output of the first pooling. If there are many frames without faces, this will be a drawback, since it will significantly affect the result of the second average pooling, reducing its values. To solve it, we only consider the resulting vectors from the frames that have at least one face to do the second pooling and avoid all zero vectors.

- Actions that have no faces: 5.7% (1,290/22,327) of the training set actions are faceless. When an action has no faces, the result of pooling layers is also a vector of zeros. This may confuse our model, as there will be different actions with the same vector. To fix it, we must not consider the actions in the training, validation, and test set.

4.5.3 Model Training

During the training process, we used the SGD as the optimizer, with a momentum of 0.9. We have used SGD and not Adam, because, although Adam is faster, SGD usually generalizes better [24]. This means that, with SGD, the model will be less likely to just memorize the data, rather than learn how it behaves.

As we have said before, the activation function of the second layer is a *SoftMax*, as is typical for classification models. Therefore, we have used the *Cross-Entropy Loss* as the loss function. Another consideration to keep in mind when training a model is how long to train it, or when to stop training it. If we train not enough, the model will be underfitting the train set, but if we train it too much, the model will overfit the train set (if it is not big enough) and will have a lower performance in the test set [3]. Hence, to avoid overtraining the model, we have added an early stop of 15 iterations.

4.5.4 Model Constraints and Considerations

The first consideration is that, as we can see in Figure 1, our data is highly imbalanced. Some actions predominate over others and can cause training problems, such as the model only learns and predicts classes among the most repeated ones [19]. There are different techniques to avoid data imbalance, such as changing the performance metric, resampling data, generating synthetic data, or testing penalized models [2]. Oversampling and undersampling can create new issues, such as to cause overfitting or stopping the model from learning certain information that it could have learned if we had not undersampled the data [37]. We have tried to reduce the effect of imbalanced data using the weighted loss technique. This consists of giving more weight to the loss function encountered in the classes with fewer repetitions, and less weight to the predominant classes, depending on how much each class is repeated [37]. Hence, each class will have a weight. There are many ways to calculate the weights depending on the repetitions of each class, and in our approach, we have chosen the following:

- Inverse of Number of Samples (INS) [37]:

$$W = \frac{\frac{1}{\text{Number of samples of each class}}}{\sum(\frac{1}{\text{Number of samples of each class}})} * \text{Number of classes}$$

- Opposite of the Class Ratio [41]:

$$W = 1 - \frac{\text{Number of samples of each class}}{\text{Total number of samples}}$$

The results of using this approach are shown in Section 5.

Another problem is that we have too many classes (17) to classify and it is very difficult to know which classes the model is good at classifying, and which are not. To solve this problem and to know which classes are better differentiated, we have trained and tested the model using fewer classes, and with different combinations among them.

We can see the results depending on the actions we used in Chapter 5. To choose which actions to use, we consider which actions can have more emotions, and we also look at *Table A.1* to see if those actions have enough faces to include them. We will see the results considering different actions and a different number of them in the experimental results section.

Chapter 5

EXPERIMENTAL RESULTS

As we have said in section 4.5.4, we have trained and tested our model considering all the 17 classes of the Dataset, and using fewer classes with different combinations. This section is divided into two parts: The first part presents and discusses the results obtained from testing our model with all possible classes, and the second one the results depending on the different subsets of classes we have selected.

Table 5.1 presents a comparison of the test accuracy given different weight equations and time windows using all the classes of the Dataset. As we can see, if we do not use a pondered loss equation, or we use the Opposite of the Class Ratio equation, the test accuracy is the same among the three tested time windows. That is because using both techniques, our model always predicts the most repeated class, and the test accuracy is the proportion of the majority class between all the classes in the test set. However, if we use the INS equation for the pondered loss, the test accuracy is smaller, but our model learns to distinguish between several classes, because of INS equation is the most restrictive (further penalizes the majority classes). For example, if we assign a time window of $T = 10$, it is able to distinguish between *Clearance*, *Ball out of Play* and *Throw-In*, with 34%, 62.6% and 28.5% of accuracy per class.

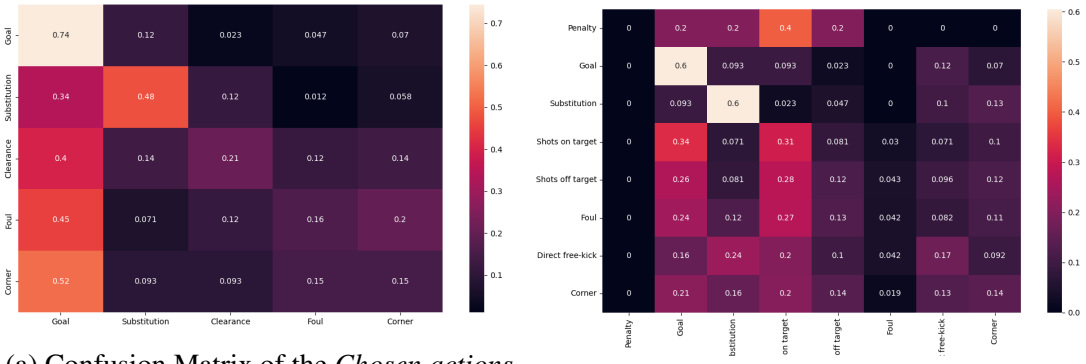
Weight Equation	$T = 10$ ($W = 81$)	$T = 16$ ($W = 129$)	$T = 20$ ($W = 161$)
Not using Weighted Loss	28.7%	28.6%	28.5%
INS	**25.2%**	**18.4%**	**20.9%**
OCR	28.7%	28.6%	28.5%

Table 5.1: Test accuracy for all classes experiment.

Hence, the model can behave in different ways and distinguish different classes with different accuracy per class. This is why we have decided to observe the behavior of the model considering only certain classes, as it is shown in *Table 5.2*.

We have chosen three different sets of classes. For the first set, we have decided to choose the classes that have more faces per action (M.FpA), inspecting at the plots of *Table A.1* and more matches. These actions are the ones that have more input features and, consequently, more information. For the second set, we have chosen the classes that we think that emotion plays an important role (Em). Finally, the last set of actions include the ones that our model differentiates well (Chosen). *Table 5.2* shows the total test accuracy (TA) and the accuracy per class given these three different sets of actions, and depending on the time window and the weight equation. As can be seen, we are not considering a time window of 20 seconds. That is because the model performance decreases, and is possible that is due to the action overlapping. We also do not consider to use a loss function without weights, because without class restrictions the performance also decreases significantly.

As can be seen in *Table 5.2*, the classes that have more faces per action are the ones that present more test accuracy, but in those actions, our model is only able to classify *Clearance* and *Substitution*, and the first with much more accuracy per class than the other. This is another example that our model tends to predict the majority class. For the *Chosen* and *Subj* actions sets, it can be observed that our model learns between more actions, and is able to differentiate between 5 and 7 different ones if we use the INS weight equation and a time window of 16 seconds as it is shown in *Table 5.2* and *Figure 5.1*. These actions are *Goal*, *Substitution*, *Shot on target*, *Shot off target*, *Clearance*, *Foul*, *Direct free-kick*, and *Corner*. The actions that achieve more accuracy per class are *Goal* and *Substitution*, and it makes sense because substitutions have many faces per action, and *Goal* is probably the action that has the clearest emotions. If we use a time window of 10 seconds, we also find remarkable results, differentiating between 4 classes.



(a) Confusion Matrix of the *Chosen actions*

(b) Confusion Matrix of the *Emotion actions*

Figure 5.1: Confusion Matrix

Observing both Confusion Matrix in *Figure 5.1* (and knowing that on the *x-axis* we find the actions that the model predicts, and on the *y-axis* the actions that really are) we can observe that in most classes, the diagonal cell has the highest value of its respective row and column. This means that the model is correct in most of the predictions it makes. Although the goal is the action with the highest accuracy, 34% of the shots on target are also detected as a goal. This is because many times these actions will happen at the same time and will be completely overlapped. It also confuses a lot of shots on target and shots off target, detecting 28% of the actions that are shots off target as shots on target, and 26% as goal. It is also an explainable confusion since they are very similar actions that can lead to cause the same emotions.

C	WE	TW	TA	Pen	Ko	Goal	Sub	Son	Sof	Cl	Foul	Dfk	Cor
Chosen	**INS**	**10**	**41%**	-	-	**0%**	**76%**	-	-	**21%**	**64%**	-	**2%**
Chosen	**INS**	**16**	**22%**	-	-	**74%**	**47%**	-	-	**21%**	**16%**	-	**14%**
Chosen	**OCR**	**10**	**50%**	-	-	**0**	**18%**	-	-	**63%**	**70%**	-	**2%**
Chosen	OCR	16	46%	-	-	0%	0%	-	-	48%	74%	-	0%
Em	**INS**	**10**	**18%**	**0%**	-	**0%**	**80%**	**56%**	**13%**	-	**0%**	**0%**	**16%**
Em	**INS**	**16**	**17%**	**0%**	-	**60%**	**60%**	**31%**	**12%**	-	**4%**	**17%**	**14%**
Em	OCR	10	26%	0%	-	0%	0%	0%	0%	-	100%	0%	0%
Em	OCR	16	36%	0%	-	0%	0%	0%	0%	-	100%	0%	0%
M.FpA	INS	10	59%	0%	0%	0%	18%	-	-	93%	-	-	-
M.FpA	INS	16	59%	0%	0%	0%	29%	-	-	89%	-	-	-
M.FpA	OCR	10	61%	0%	0%	0%	22%	-	-	95%	-	-	-
M.FpA	OCR	16	61%	0%	0%	0%	20%	-	-	94% -	-	-	-

Table 5.2: Test Accuracy and accuracy per class given different classes.
Pen: Penalty, Ko: Kick-off, Sub: Substitution, Son: Shots of target, Sof: Shots of target, Cl: Clearance, Cor: Corner

In conclusion, our model differentiates the classes better when using INS as a weight equation and a time window of 16. As we have said, INS is more restrictive and penalizes more the predominant classes, which helps to increase the accuracy of the less repeated ones. The explanation of why a time window of 16 seconds can be found by looking at the *boxplots* in *Figure A.1*. As already mentioned (and as we can see in the figure), most of the faces in the actions are between the 60 frames before the action and the following 60 frames, that is, 7.5 seconds before and after the action, which gives a time window of 15s. Therefore, using a time window of 10 seconds implies a significant loss of information.

Chapter 6

CONCLUSIONS AND FUTURE WORK

In this work, we have been able to implement an efficient feature extractor[1] implementing face and emotion detection models, and we extracted meaningful features. Given these features, we have created an action classification model[1] that predicts the corresponding actions, achieving the main objective of the research.

We have extracted enough features for our model to learn, achieving almost 33k action examples, processing a total of 135 matches of the 500 that provides SoccerNet.

After testing and experimenting with the model, we have reached the conclusion that the best option is to use a time window of 16 seconds, and a weighted loss calculated with the INS equation.

We have not been able to make our model know how to differentiate between all 17 SoccerNet classes (as it is difficult due to the limited number of examples of some classes), but it has learned to differentiate between 7 classes, with a remarkable accuracy per class.

We consider that this work serves as an initial approach for understanding soccer games based on emotions, as we have completed the main objective of detecting football actions using face emotion features, and that in the future it can be greatly improved (and be able to classify between more classes), as we describe below.

[1] GitHub: https://github.com/joseprs/ActionClassification

6.1 Future Work

After studying and implementing all parts of this thesis, and observing how our model behaves, we have identified that we can make some improvements and future investigations.

As improvements, we can try to use Data Augmentation to balance the data by upsampling the less repeated classes, generating synthetic data. It could be interesting to try a less robust face detector, because RetinaFace detects even partially occluded and blurred faces, and a different pooling of the model, in such a way that the result of the pooling layers is not a single vector, but N vectors, referring to different periods within the action.

We can also do different future research, to complement the model created. One of these investigations could be to consider audio emotion features in our model and combine it with what we already have, to make a double check of the action classification. What would also be interesting is to implement cameras that focus on the public all the time, as it would give us much more information than now, and test our model with those images. Finally, one factor that we can consider and investigate in the future is to take into account if the player that does a specific action is from the local or the visitant team because depending on that, the emotions of the fans will be different.

Bibliography

[1] Brandon Amos, Bartosz Ludwiczuk, and Mahadev Satyanarayanan. Openface: A general-purpose face recognition library with mobile applications. Technical report, CMU-CS-16-118, CMU School of Computer Science, 2016.

[2] Jason Brownlee. 8 tactics to combat imbalanced classes in your machine learning dataset, august 2015. Machine Learning Mastery [Online; Last Update 15-August-2020].

[3] Jason Brownlee. A gentle introduction to early stopping to avoid overtraining neural networks, december 2018. Machine Learning Mastery [Online; posted 7-December-2018].

[4] Paolo Caressa. How can ai support football tech staff in technical and tactical analysis and decision making?, November 2021. CodeMotion [Online; posted 10-November-2021].

[5] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. *CoRR*, abs/1703.06211, 2017.

[6] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, 2005.

[7] Adrien Deliège, Anthony Cioppa, Silvio Giancola, Meisam Jamshidi Seikavandi, Jacob V. Dueholm, Kamal Nasrollahi, Bernard Ghanem, Thomas B. Moeslund, and Marc Van Droogenbroeck. Soccernet-v2 : A dataset and benchmarks for holistic understanding of broadcast soccer videos. *CoRR*, abs/2011.13367, 2020.

[8] Jiankang Deng, Jia Guo, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. *CoRR*, abs/1801.07698, 2018.

[9] Jiankang Deng, Jia Guo, Yuxiang Zhou, Jinke Yu, Irene Kotsia, and Stefanos Zafeiriou. Retinaface: Single-stage dense face localisation in the wild. *CoRR*, abs/1905.00641, 2019.

[10] Maheshi B. Dissanayake Dinusha Nuwan Ranaweera Dilanka Sasindu Perera, Tharindu Ekanayake. Ball localization and player tracking using real time object detection. *International Conference on Advances in Computing and Technology (ICACT–2020) Proceedings*, 2020.

[11] Silvio Giancola, Mohieddine Amine, Tarek Dghaily, and Bernard Ghanem. Soccernet: A scalable dataset for action spotting in soccer videos. *CoRR*, abs/1804.04527, 2018.

[12] Jonathan Hui. Understanding feature pyramid networks for object detection (fpn), March 2018. Medium [Online; posted 27-March-2018].

[13] Samuel Hurault, Coloma Ballester, and Gloria Haro. Self-supervised small soccer player detection and tracking. *CoRR*, abs/2011.10336, 2020.

[14] Yudong Jiang, Kaixu Cui, Leilei Chen, Canjin Wang, Chen Wang, Hui Liu, and Changliang Xu. Comprehensive soccer video understanding: Towards human-comparable video understanding system in constrained environment. *CoRR*, abs/1912.04465, 2019.

[15] Kaggle. Challenges in representation learning: Facial expression recognition challenge, 2013. [Kaggle Competition].

[16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[17] André Teixeira Lopes, Edilson de Aguiar, Alberto F. De Souza, and Thiago Oliveira-Santos. Facial expression recognition with convolutional neural networks: Coping with few data and the training sample order. *Pattern Recognition*, 61:610–628, 2017.

[18] Isabel Margarit. Los orígenes del fútbol, October 2018. La Vanguardia [Online; posted 26-October-2018].

[19] Saikat Mazumder. 5 techniques to handle imbalanced data for a classification problem, june 2021. Analytics Vidhya [Online; posted 21-June-2021].

[20] Michele Merler, Khoi-Nguyen C. Mac, Dhiraj Joshi, Quoc-Bao Nguyen, Stephen Hammer, John Kent, Jinjun Xiong, Minh N. Do, John R. Smith, and

Rogerio Schmidt Feris. Automatic curation of sports highlights using multimodal excitement features. *IEEE Transactions on Multimedia*, 21(5):1147–1160, 2019.

[21] Fact Monster. Handling imbalanced classes with weighted loss in pytorch, february 2017. [Online; posted 21-February-2017].

[22] NN. Deep face detection with retinaface in python, april 2021. Sefiks.com [Online; posted 27-April-2018].

[23] University of York. The real world impact of facial detection and recognition, april 2022. [Online; posted 14-April-2022].

[24] Sieun Park. A 2021 guide to improving cnns-optimizers: Adam vs sgd, june 2021. Medium [Online; posted 21-June-2021].

[25] Omkar M. Parkhi, Andrea Vedaldi, and Andrew Zisserman. Deep face recognition. In *British Machine Vision Conference*, 2015.

[26] P.J. Phillips, Hyeonjoon Moon, S.A. Rizvi, and P.J. Rauss. The feret evaluation methodology for face-recognition algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(10):1090–1104, 2000.

[27] Morteza Analouia Sara Shahsavarania and Reza Shoja Ghiass. Deep-id: A novel model for multi-view face identification using convolutional deep neural networks. Technical report, School of Computer Engineering, Iran University of Science and Technology, Tehran, Iranb; Université Laval, Quebec City, Quebec, Canada, 2016.

[28] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. *CoRR*, abs/1503.03832, 2015.

[29] Sciforce. Face detection explained: State-of-the-art methods and best tools, June 2021. [Online; posted 17-June-2021].

[30] Sefik Serengil. Face alignment for face recognition in python within opencv, february 2018. Sefiks.com [Online; posted 23-February-2018].

[31] Sefik Serengil. Facial expression recognition with keras, january 2018. Sefiks.com [Online; posted 1-January-2018].

[32] Sefik Serengil. Apparent age and gender prediction in keras, february 2019. Sefiks.com [Online; posted 13-February-2019].

[33] Sefik Serengil. Face recognition with dlib in python, july 2020. Sefiks.com [Online; posted 11-July-2020].

[34] Sefik Serengil. Large scale face recognition for deep learning, may 2020. Sefiks.com [Online; posted 25-May-2020].

[35] Sefik Serengil. Deepface – the most popular open source facial recognition library, 2021. viso.ai [Online].

[36] Jiawei Shi and Songhao Zhu. Learning to amend facial expression representation via de-albino and affinity. *CoRR*, abs/2103.10189, 2021.

[37] Ishan Shrivastava. Handling class imbalance by introducing sample weighting in the loss function, december 2020. Machine Learning Mastery [Online; posted 17-December-2020].

[38] SoccerNet.

[39] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708, 2014.

[40] Thales. Facial recognition: top 7 trends (tech, vendors, use cases), june 2021. [Online; posted 24-June-2021].

[41] Haritha Thilakarathne. Handling imbalanced classes with weighted loss in pytorch, july 2021. NaadiSpeaks [Online; posted 31-July-2021].

[42] Bastien Vanderplaetse and Stéphane Dupont. Improved soccer action spotting using both audio and video streams. *CoRR*, abs/2011.04258, 2020.

[43] WorldAtlas. The most popular sports in the world, October 2020. [Online; posted 16-October-2020].

[44] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. Joint face detection and alignment using multi-task cascaded convolutional networks. *CoRR*, abs/1604.02878, 2016.

[45] Ruiheng Zhang, Lingxiang Wu, Yukun Yang, Wanneng Wu, Yueqiang Chen, and Min Xu. Multi-camera multi-player tracking with deep player identification in sports video. *Pattern Recognition*, 102:107260, 2020.

Appendix A

TABLE OF PLOTS

Table A.1: Plots that give us an idea of how the faces are distributed, in terms of quantity and time.