UNIVERSITÀ DEGLI STUDI DI CATANIA
Dipartimento di Matematica e Informatica
Corso di Laurea Triennale in Informatica

Antonio Finocchiaro

Temporal Video Segmentation of Calisthenics Skills based on Body Pose Analysis

Relazione Progetto Finale

Relatore:
Prof. Antonino Furnari

Anno Accademico 2022 - 2023

Dedicated to my grandparents
and the rest of my loving family

Contents

Abstract

Calisthenics is a rapidly expanding fitness discipline that involves using bodyweight exercises to develop strength, flexibility, and coordination. The skills branch of calisthenics focuses on mastering dynamics and static movements partially inherited from artistic gymnastics. Skills competitions are held in Calisthenics, but unlike many other sports that have developed advanced tools to assist referees, Calisthenics relies only on the judges' eyes to count the holds of the skills. This project investigates an innovative approach to recognize and track the skills performed during a Calisthenics competition. The approach primarily focuses on 2D human pose estimation, classification and temporal reasoning, with the aim to detect and temporally locate a set of eight commonly practiced skills in the discipline. For the experiments, a dataset of 573 videos has been built by following predetermined criteria, which were set in advance to ensure that the dataset met specific requirements and standards necessary for conducting the research. To enable temporal reasoning, an algorithm has been created to reconstruct a video timeline, starting from classified frames. To evaluate the effectiveness of the proposed approach, a comparison was conducted against a previous method for temporal video segmentation based on probabilistic reasoning. The achieved results demonstrate a good accuracy in skill classification and show the promising performance of the proposed algorithm in detecting skill video segments. These findings are extensively presented and discussed, examining the model's effectiveness and identifying its strengths and weaknesses. By providing a dataset and an experimental evaluation, this thesis aims to provide an initial step towards the design and implementation of automated tools which can be useful for automatic assessment in competitions or while practicing at home.

Keywords: Pose recognition, Sports video analysis, Video segmentation.

Chapter 1

Introduction

The discipline of Calisthenics is composed of various branches, namely endurance, strength, and skills. Over the past few years, each of these branches has noticed significant growth in the main competition arenas. Among these categories, the skills branch holds a particularly influential position in the international stage, owing to the impressive strength requirements of the performed poses. Calisthenics contains a wide number of skills, including their diverse variations. However, for this specific project, a carefully selected subset of skills has been chosen based on detailed considerations, which will be explained later. This project investigates method for recognizing and classifying Calisthenics skills from video footage, providing an accurate timeline with correctly segmented skill sequences. The purpose of this work extends to various applications, such as monitoring athletes' skill execution and holds during competitions or serving as a training tool to simulate a judge's role. To accomplish the ultimate goal of the project, a series of sequential steps must be undertaken to successfully complete various subtasks.

The process begins with the selection of skills to recognize and the creation of a video dataset. Next, a pre-trained pose estimator is employed to extract spatial information about the athletes' joints in the videos. Subsequently, a dataset is built from the frame-level body pose tracking results, and a multiclass classifier is designed and trained to classify skills on a frame-by-frame basis. Based on the model's predictions, a segment reconstruction algorithm is applied to construct the video timeline from individual predictions. Fig. 1.1 illustrates the input frame of a video and showcases the expected output, which represents the segmentation of the timeline.

Figure 1.1: Above: An example frame from the input video. Below: The corresponding expected output.

1.1 Thesis Outline

The subsequent chapters will provide a comprehensive account of the entire process, starting with the establishment of prerequisites. Chapter 2 will offer a detailed explanation of the tools and models employed, including *OpenPose*, a pose estimator used in this project to capture spatial information from the athlete in the video. Additionally, the multilayer perceptron, a model employed to solve the classification task, will also be thoroughly discussed. Following that, Chapter 3 will delve into the creation of the video dataset, describing the criteria designed for its construction. The heart of the project resides in Chapter 4 which describes two separate pipelines which have been explored to address distinct objectives. The first pipeline will delve into the process of extracting the keypoints of the body pose an athlete. The second pipeline will focus on the testing phase, highlighting the creation of the test dataset, the raw results obtained, and showcasing the algorithm proposed for reconstructing video segments based on the raw model labels. Chapter 5 will showcase the analysis of the outcomes, offering comprehensive graphical and numerical comparisons among various elements of the project. This will include a comprehensive comparison between the proposed method and the current state-of-the-art in video segmentation [3]. Lastly, Chapter 6 will discuss the project's limitations and areas for improvement, as well as explore possible ways to enhance and achieve better results in the future.

Chapter 2

Prerequisities

This chapter will offer an introduction to the essential concepts and tools utilized in our project, providing an explanation of their mechanisms and a reference to their usage throughout the project. We will start by introducing the machine learning models involved in this project.

We will first dig into the *Multilayer Perceptron* (MLP), which is the prediction model used to classify skills. After that, we will provide a brief overview of *Convolutional Neural Networks* (CNNs), which are deep learning models used in *OpenPose*. Lately, we will discuss two probabilistic models that were proposed in previous works and are relevant to our project's inference section. Firstly, we will provide an explanation of the *Hidden Markov Model* (HMM), which is a statistical model used to describe the probabilistic transitions between hidden states that generate observed data. Secondly, we will introduce the *Viterbi algorithm*, which is a dynamic programming algorithm used to find the most likely sequence of hidden states that corresponds to a given sequence of observations. We will explain how the *Viterbi algorithm* works and how it can be used to decode and analyze the outputs of HMMs. Therefore, we will explain how *OpenPose* works and how it is used in our project. Then, we will shift our focus to the programming language employed in our project, which is *Python*. After that, we will give a brief overview of some of the libraries we used in our project, including *PyTorch*, *Pandas*, *NumPy*, *Matplotlib*, and *Scikit-learn*. Moving on, we will discuss some of the video tools that were used in our project. Firstly, we will introduce *Kdenlive*, which is the video editing software that we utilized to process and edit our videos; additionally, we will explore the *FFmpeg* library, which is a powerful command-line tool used for manipulating and converting video and audio files.

2.1 Machine Learning models

In this section, we will introduce two models that are used in this project. The first model is employed for the classification task, while the second model is incorporated into the architecture of the pose estimation system.

2.1.1 Multilayer Perceptron

The Multilayer Perceptron [4] is a type of artificial neural network that expands on the simple perceptron model, one of the earliest and most essential neural network models, by integrating multiple layers of interconnected neurons arranged in a feedforward network structure. The following Fig. 2.1 illustrates an MLP.

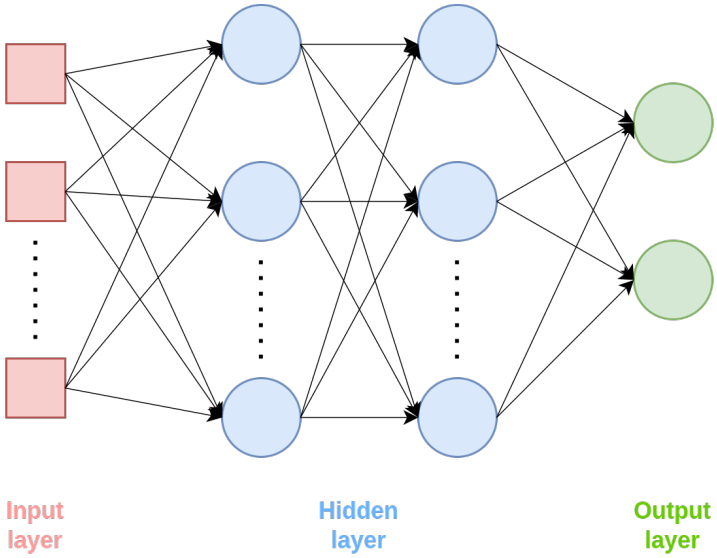

Figure 2.1: An MLP is composed of an input layer, an hidden layer, computing a latent representation of the input, and an output layer, which is used for the end prediction.

MLPs have been shown to be highly effective at addressing a wide range of tasks, including classification, regression, and prediction, by utilizing multiple layers of neurons and non-linear activation functions. In contrast to the simple perceptron, which is only capable of handling linearly separable problems, MLPs can successfully solve complex and non-linear problems. Its basic structure consists of an input layer, one or more hidden layers, and an output layer. The input layer receives data as input, and each neuron in the layer corresponds to a single input feature. The output of each input neuron

is transmitted to the neurons in the first hidden layer. The hidden layers are where the MLP performs most of its computation. Each neuron in a hidden layer receives inputs from all the neurons in the previous layer and produces an output based on a weighted sum of those inputs. The weights on the connections between the neurons are adjusted during training to optimize the performance of the network. The output layer produces the final output of the network. In a classification task, for example, each neuron in the output layer corresponds to a different class, and the neuron with the highest output value indicates the predicted class.

During the training process, the network learns to approximate a function that maps inputs to outputs. To achieve this, the network is trained using a supervised learning algorithm called backpropagation. *Backpropagation* involves feeding the network with a set of input-output pairs, and computing the error between the predicted output and the true output. The weights on the connections between the neurons are then adjusted iteratively using an optimizer, based on the error between the predicted output and the true output. The optimizer uses this error to compute the gradient of the loss function with respect to the weights, which determines the direction and magnitude of the weight updates. The process of feeding forward the input through the network, computing the error, and then propagating the error back through the network to update the weights continues for a number of iterations (or epochs), until the error is minimized and the network achieves a satisfactory level of accuracy.

Later in the project, different experiments will be conducted using various optimizers such as *Adam* [4], *SGD* [4], and *RMSprop* [4] to compare their performance in training the MLP. These experiments will aim to improve the model's performance by adjusting hyperparameters such as the learning rate, regularization strength, and activation function. The effects of these changes will be then observed by measuring the accuracy and speed of convergence of the loss function. MLPs can be used for a wide range of tasks, including image and speech recognition or natural language processing.

2.1.2 Convolutional Neural Network

A convolutional neural network (CNN) [4] is a type of artificial neural network that is commonly used in image recognition and computer vision tasks. CNNs are designed to recognize patterns and features in images by using a process called convolution.

The basic structure of a CNN (see Fig. 2.2) consists of several layers, in-

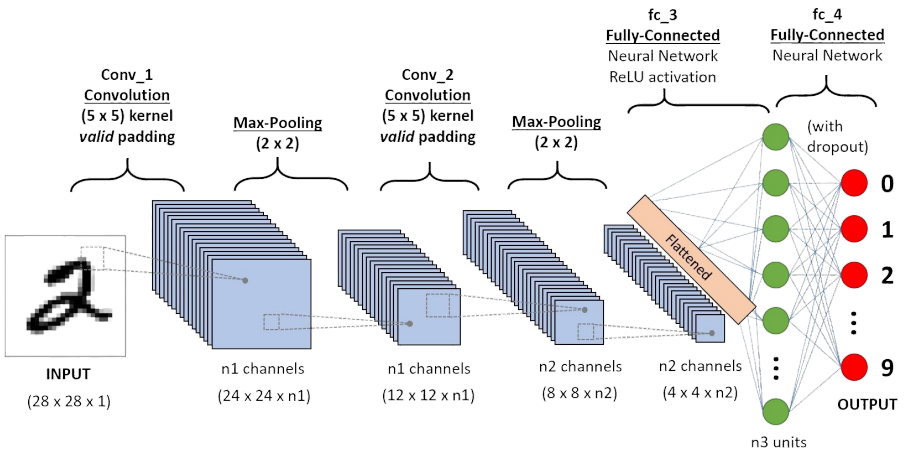

Figure 2.2: A CNN architecture consists of convolutional layers, responsible for extracting meaningful features from the input, followed by pooling layers that downsample the representations, and fully connected layers that perform the final prediction.[1]

cluding convolutional layers, pooling layers, and fully connected layers. The convolutional layers are the core of the network, where the image is processed by convolving it with a set of learnable filters, or kernels, to extract meaningful features. Each filter is typically a small square matrix that is slid across the image, computing a dot product between the filter and the corresponding region of the image. The output of each convolutional layer is passed through an activation function, which applies a non-linear transformation to the output values. The resulting feature maps are then fed into a pooling layer, which down-samples the feature maps by reducing their spatial dimensions. This helps to reduce the number of parameters in the network and make it more efficient. The fully connected layers are used to make the final predictions, mapping the learned features to the output classes or values.

During training, the weights and biases of the filters and the fully connected layers are adjusted using backpropagation, where the error between the predicted output and the true output is propagated backwards through the network, and the weights are updated accordingly. The training process continues until the network achieves a satisfactory level of accuracy. CNNs are widely used in many applications, including image classification, object detection, and segmentation, and have achieved state-of-the-art performance in many benchmarks. They are also used in other domains, such as natural

[1]Image source: `https://www.analyticsvidhya.com/blog/2020/10/what-is-the-convolutional-neural-network-architecture/`

language processing and speech recognition, by converting the input data to a 2D image-like representation (e.g., spectrograms or co-occurrence matrices). One application of convolutional neural networks is in the field of human body pose estimation, which is the task of detecting and locating the joints and body parts of a person in an image or video. OpenPose is a popular pose estimation system that uses a variant of CNNs known as "convolutional pose machines" (CPMs) to perform this task.

In summary, convolutional neural networks and their variants, such as CPMs, have revolutionized the field of computer vision and enabled the development of sophisticated systems such as OpenPose for human pose estimation.

2.2 Probabilistic models

In this section we will discuss about two probabilistic models used in [3]. The authors of [3] propose a method for segmenting egocentric videos based on the locations visited by the user who recorded the video.

The method allows the user to specify which locations they want to keep track of and can account for negative locations using an effective rejection method. The authors collected a dataset of egocentric videos containing 10 personal locations of interest to perform experimental analysis, and the results show that the proposed method is accurate and outperforms existing state-of-the-art methods. To achieve these results, the authors make use of Hidden Markov Models (HMM) and the Viterbi algorithms, which are presented in the following sections.

2.2.1 Hidden Markov Model

A Hidden Markov Model (HMM) [1] is a statistical model that can be used to describe sequential data, where the underlying state is hidden and can only be inferred from the observed data. It is a type of generative model, meaning that it models the probability distribution of the observed data as well as the underlying hidden states that generate the observed data. HMMs are based on the Markov assumption, which states that:

$$P(y_i|y_{i-1} \dots y_1) = P(y_i|y_{i-1})$$

In other words, the future state is conditionally independent of the past states given the present state. This assumption simplifies the modeling of sequential data, and allows the use of dynamic programming algorithms such as the *Viterbi algorithm* and the Forward-Backward algorithm to compute

the likelihood of the observed data and to infer the most likely sequence of hidden states.

An HMM is based on two types of probability distributions: the transition probabilities and the emission probabilities. The transition probabilities, denoted as T, describe the probability of moving from one state to another. Mathematically, T[i][j] represents the probability of transitioning from state i to state j. The emission probabilities, denoted as E, describe the probability of observing a particular output given the current hidden state. E[i][k] represents the probability of emitting output k when in state i. The transition probabilities and the emission probabilities are typically represented as matrices, with each element in the matrix corresponding to a particular transition or emission probability.

In a typical HMM application, the goal is to infer the most likely sequence of hidden states given the observed data. This is known as the decoding problem. The Viterbi algorithm is a dynamic programming algorithm that can be used to efficiently compute the most likely sequence of hidden states.

2.2.2 Viterbi algorithm

The Viterbi algorithm is a dynamic programming algorithm that is commonly used with HMM. In HMMs, there are two types of states: observable states and hidden states. The Viterbi algorithm is used to determine the most likely sequence of hidden states that could have produced a given sequence of observations.

Algorithm 1 Viterbi algorithm pseudocode

Initialize:
for each hidden state s **do**
    $g[1, s] \leftarrow B(s, o_1) \cdot A(s_0, s)$
for $t = 2$ to T **do**
    for each hidden state s **do**
        $g[t, s] \leftarrow \max_{s_0} g[t-1, s_0] \cdot A(s_0, s) \cdot B(s, o_t)$
        $h[t, s] \leftarrow \arg\max_{s_0} g[t-1, s_0] \cdot A(s_0, s) \cdot B(s, o_t)$
Follow $h[t, s]$ **to find** $s_T^*, s_{T-1}^*, ..., s_1^*$. **Starting at** $t = T$
$s_T^* \leftarrow \arg\max_s g[T, s]$
for $t = T - 1$ to 1 **do**
    $s_t^* \leftarrow h[t + 1, s_{t+1}^*]$

The algorithm aims to find the most likely sequence of hidden states based on a given set of observations. It accomplishes this by calculating probabil-

ities at each step and making decisions based on the highest probabilities. The process begins by initializing the algorithm. For each hidden state, an initial probability is calculated, which combines the emission probabilities of the observed state and the transition probabilities from the initial state. Next, the algorithm iterates through each time step, starting from the second step and going up to the last step. At each time step, it considers every possible hidden state and calculates the probability of reaching that state. This is achieved by taking the maximum probability from the previous time step, multiplying it by the transition probability from the previous state to the current state, and multiplying it again by the emission probability of the observation corresponding to the current state. Additionally, the algorithm keeps track of the most probable previous state for each current state. Once all time steps have been processed, the algorithm determines the most likely sequence of hidden states. It starts at the final time step and chooses the state with the highest probability. Then, it traces back through the sequence of states by selecting the most probable previous state at each step until reaching the initial state. This way, the algorithm identifies the most likely sequence of hidden states.

2.3 OpenPose

This open-source library is the first available realtime system for multi-person 2D pose detection, including body, foot, hand, and facial keypoint; it is developed by CMU Perceptual Computing Lab [2]. OpenPose uses a bottom-up representation of association scores via *Part Affinity Fields* (PAF) to learn to associate body parts with individuals in the image. Its bottom-up approach brings various advantages instead of using a top-down approach that has two common issues:

1. It is known to fail when multiple people are in close proximity or the person is occluded, making the entire process fail.

2. The runtime is proportional to the number of people in the image.

The developers have proven first that the PAF refinement is crucial for maximizing the accuracy. They increased the network depth but removed the body part refinement stages. Second, they created a foot dataset that has been publicly released. They show that a combined model with body and foot keypoints can be trained preserving the speed of the body-only model while maintaining its accuracy.

2.3.1 Multi Person Pose Estimation

The multi-person pose estimation of OpenPose is based on a bottom-up approach, where each keypoint detection is independently performed by a network. The keypoints are then linked together to form a human pose. OpenPose uses a deep neural network to estimate the human pose, and it is trained on a large dataset of annotated images. The network takes as input an RGB image and outputs the positions of the keypoints. The keypoints include body joints such as the head, neck, shoulders, elbows, wrists, hips, knees, and ankles. The network is able to estimate the poses of multiple people in real-time, making it suitable for various applications, such as sports analysis, healthcare, and robotics.

2.3.2 Network architecture

The architecture of the OpenPose multi-stage CNN (see Fig. 2.3) is designed to predict both the PAFs (Part Affinity Fields) and confidence maps.

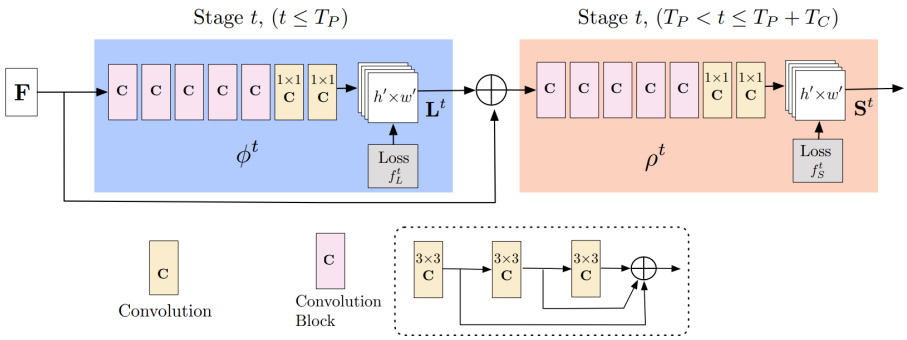

Figure 2.3: OpenPose multi-stage CNN architecture. [2]

The first set of stages in the CNN is responsible for predicting the PAFs, which are vector fields that encode the location and orientation of body parts relative to each other. These stages take as input the original image and produce an output tensor containing the PAFs. The last set of stages is responsible for predicting the confidence maps, which represent the likelihood that a given pixel in the image belongs to a particular body part. These stages take as input the original image and produce an output tensor containing the confidence maps. The predictions of each stage are concatenated with their corresponding image features and passed as input to the subsequent

[2]Image source : `https://medium.com/analytics-vidhya/understanding-openpose-with-code-reference-part-1-b515ba0bbc73`

stage. This allows the network to refine its predictions at each stage by incorporating information from previous stages. In the original approach, convolutions of kernel size 7 were used in the network. However, in the improved version, three layers of convolutions of kernel size 3 are used instead. These layers are concatenated at the end, which allows the network to capture more fine-grained details and improve its accuracy.

2.3.3 Pose estimation models

There are three main models used in OpenPose:

COCO (see Fig. 2.4) is a pre-trained deep learning model used in the OpenPose framework for multi-person pose estimation. This model is trained on the COCO (Common Objects in Context) dataset, which contains over 200,000 images with more than 250,000 labeled people. The model is capable of detecting up to 18 key points in each person. These keypoints can be used to estimate the pose of each person in the image, including their joint angles and body orientation. However, despite its large use, it has not been chosen for this project due to its lower accuracy and fewer keypoints compared to other available models.

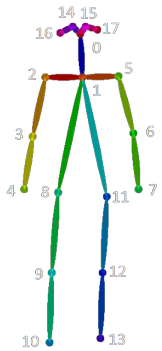

Figure 2.4: Coco keypoints.[3]

BODY_25 It can accurately detect and locate up to 25 keypoints on the human body, including the head, torso, arms, and legs. *BODY_25* is the default model used in OpenPose, and it is trained on a dataset of around 25,000 images. The model is trained on the COCO dataset and uses a combination of convolutional neural networks (CNNs) and part affinity fields to estimate the keypoints of a person in an image.

BODY_25B The *BODY_25B* model is an updated version of the *BODY_25* model with higher accuracy. This model includes some extra PAF (Part Affinity Field) channels and additional body keypoints, specifically the MPII head and neck keypoints. The *BODY_25B* model was trained for the Single-Network Whole-Body Pose Estimation paper and is known to have higher accuracy than the default *BODY_25* model; however, it operates at a slower

[3]Image source : `https://www.researchgate.net/figure/18-keypoints-estimated-by-the-OpenPose-Model_fig2_343232689`

speed compared to it. We opted to this model in our project due to its superior performance in accurately tracking body animation.

Fig. 2.5 showcases different body representations of the last two models.

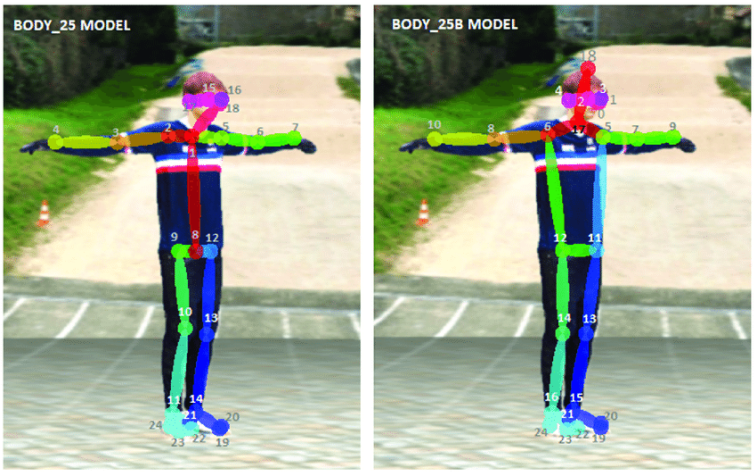

Figure 2.5: Unlike the *BODY_25* model, the neck and middle hip keypoints are not artificially created in the *BODY_25B* model. Instead, the neck keypoint is determined by the middle point of the shoulders, and the middle hip keypoint is determined by the middle point of the hips.[4]

BODY_25 is the default and most commonly used model, while COCO is faster and more efficient for real-time applications. *BODY_25B* is the most accurate and robust model, but it may be slower and require more computational resources.

2.4 Python and related libraries

Python is a high-level, object-oriented, and interpreted programming language (version used: 3.10). It is very popular in the software world and can be used in many fields, including web development, data analysis, artificial intelligence, and machine learning. Thanks to its wide range of libraries and frameworks, Python offers developers many resources to create sophisticated and high-performance applications. Lots of its libraries have been used in this

[4]Image source : `https://www.researchgate.net/figure/The-experimental-body-25b-OpenPose-model-is-more-accurate-than-the-default-body-25-one_fig5_355221981`

project, including PyTorch in particular, that played a significant role in the recognition of skills section.

2.4.1 PyTorch

PyTorch is an open source library primarily developed by Facebook AI Research. Based on the Python programming language, it offers a wide range of features for building machine learning models, including deep neural networks, optimization algorithms, loss functions, data visualization tools, and much more. PyTorch provides functionality for processing multidimensional arrays, similar to what is provided by Numpy, but with a more comprehensive and powerful library. Its particular strength lies in processing tensors using GPU acceleration, which allows for significant performance improvements over CPU processing. PyTorch distinguishes itself from other machine learning libraries for its flexibility and ease of use; it is divided into modules, each of which offers specific functionality for creating machine learning models.

Some of the PyTorch modules used in the project include:

- **torch.nn**: This module provides tools for creating neural networks, including linear, convolutional, and recurrent layers, activation functions, loss functions, and much more.
- **torch.utils.data**: This module offers tools for managing and manipulating datasets, such as creating batches, dividing into training and validation sets, and transforming data.
- **torch.optim**: This module provides optimization algorithms for tuning the parameters of machine learning models, such as stochastic gradient descent.

Thanks to its flexibility and high performance, PyTorch represents a valuable resource for developers and researchers in the field of machine learning and deep learning.

2.4.2 Other libraries

Numerous other libraries were employed for various purposes throughout this project. A comprehensive list of these libraries is presented in the following paragraphs.

Scikit-learn is a free, open-source library for Python that provides tools for data mining, data analysis, and machine learning. It is built on top of other scientific computing libraries such as NumPy, SciPy, and matplotlib, and provides a simple and efficient way to implement many popular machine learning algorithms. Specifically, we utilized it for splitting our dataset into training and testing sets, and for generating a confusion matrix as one of the evaluations metrics.

Pandas is a data manipulation library that offers a variety of powerful tools to quickly and efficiently analyze, clean, and transform data. It provides functionalities such as merging, reshaping, indexing, slicing, and grouping, which are essential for data manipulation. Its primary data structure, DataFrame, is a flexible and robust data structure that allows users to work with data in a tabular form, similar to a spreadsheet. In our project, Pandas has been extensively used to work with CSV files and to manipulate data in some algorithms.

Numpy is a numerical computing library for Python. It provides a multi-dimensional array object, along with a set of mathematical functions to operate on these arrays. It is widely used in scientific and data-intensive computing because of its ability to perform complex mathematical operations efficiently and effectively. It provides a high-performance array computing functionality and tools for working with them, such as linear algebra or random number generation. In our project, NumPy has used to perform various operations on arrays, including computing statistical measures such as mean, and handling data related tasks like saving the output classes to an external file.

Matplotlib Matplotlib is a Python library used for creating static, animated, and interactive visualizations. It offers a range of customization options to make graphs, histograms, scatterplots, and other visualizations. Matplotlib is widely used in scientific computing, data analysis, and machine learning to explore data patterns and relationships.

2.5 Video tools

In this section, we will introduce two essential video tools that are involved in handling various tasks throughout the entire process.

2.5.1 Kdenlive

Kdenlive is a free and open-source non-linear video editor for the KDE desktop environment. It features a multi-track timeline that allows for overlapping of multiple videos and audio tracks, as well as the ability to add custom effects and transitions. Additionally, rendering is done through a separate and non-blocking process, meaning that it is possible to interrupt, pause, and resume rendering while performing other operations. This feature has proven particularly useful in the video processing phase.

Kdenlive supports a variety of audio and video file formats, including AVI, QuickTime, MPEG, MP3, MP4, WMV, MOV, and many others. Furthermore, thanks to its integration with FFmpeg, Kdenlive is able to handle video and audio file conversion, allowing for import and export in various formats. Thanks to its ease of use and numerous included features, Kdenlive is a viable alternative to paid video editors such as Sony Vegas Pro, Adobe Premiere Pro, and Final Cut Pro.

2.5.2 FFmpeg

FFmpeg is an open-source software based on the libavcodec library. It was originally developed for Linux, but can be compiled and run on any operating system. FFmpeg allows users to perform a wide range of audio and video processing tasks, such as converting audio or video files to different formats and codecs, extracting audio or video streams from multimedia files, merging multiple audio or video files into a single file, and adding subtitles or watermarks to videos. It has a command-line interface for user interaction, and supports a wide range of audio and video formats and codecs, including popular ones like MP3, AAC, H.264, and MPEG-4.

Chapter 3

Dataset creation

In this chapter, we will showcase the dataset that was created and delve into the dynamics involved in its development. We will begin by discussing the selected skills, followed by an explanation of the mechanism used to retrieve the videos. Additionally, we will provide an analysis of the dataset.

3.1 Skills selection

To track and recognize calisthenics skills, we first created a dedicated dataset. Given that in Italy, calisthenics primarily consists of static skills, we carefully selected the specific skills that needed to be included before proceeding with video scraping. We chose a limited set of skills, which are as follows:

1. **Back lever** (bl)
2. **Front lever** (fl)
3. **Human flag** (flag)
4. **Iron cross** (ic)
5. **Maltese** (mal)
6. **One arm front lever** (oafl)
7. **One arm handstand** (oahs)
8. **Planche** (pl)

Each skill has a different difficulty level and requires a certain level of strength to be performed. Now, we describe each skill, providing an overview and exploring the reasons why it has been included in our dataset.

Back lever (see Fig. 3.1) is one of the most common static skills in calisthenics, as it requires relatively low strength compared to other skills. Its accessibility makes it a great skill to start with, and it is included in every competition skills rule book. In this pose, the arms must be straight, and the body connections between the foot/pelvis and pelvis/neck should create a single, unique line. The variant included is the "full" one.

Figure 3.1: Full prone back lever.

Front lever (see Fig. 3.2) is the most popular pull element. It requires a good back strength. There are lots of variants of it, decreasing/increasing its difficult; in this project, the only variant considered is the "full" and the "one arm" (later explained). The line reference of the back lever, is here present too, but the body is directed to the top.

Figure 3.2: Full front lever.

Human flag (see Fig. 3.3) is a popular skill performed by athletes from various disciplines such as pole dancers and gymnasts. It involves the athlete gripping a vertical pole and holding their body parallel to the ground, with their torso oriented perpendicular to it. This skill can be performed on various objects, such as vertical poles or street signs, and is included in our project due to its wide range of applications.

Figure 3.3: Human flag.

Iron cross (see Fig. 3.4) is a skill that calisthenics has imported from artistic gymnastics. It is performed on rings and requires an enormous amount of strength and tendon conditioning to be performed. In this skill, the body has a vertical orientation and the elbows must be straight. For a perfect execution, the line linking the height of the hands with the shoulders should be horizontal. It has been chosen for its prominent role in rings skill sets in calisthenics competitions.

Figure 3.4: Iron cross.

Maltese (see Fig. 3.5) is one of the hardest skill in the game. Similar to the iron cross, it is imported from the artistic gymnastics. It is a push element that requires and extreme body condition and specific strength. In this pose, the body is held parallel to the ground, with the arms extended straight out to the sides and the palms facing downward. The legs are lifted towards the chest and kept straight, while the torso is leaned forward to maintain the horizontal position. It can be performed on various object like: floor, rings, parallel bars etc... It has been included in our project due to its significant usage in calisthenics skills competitions and its score in them.

Figure 3.5: Full maltese.

One arm front lever (see Fig. 3.6) is an asymmetric skill that involves holding a front lever position with only one arm. Considered one of the most difficult pulling elements, this variant is included in the skill sets of athletes who have achieved an extreme level of proficiency in calisthenics.

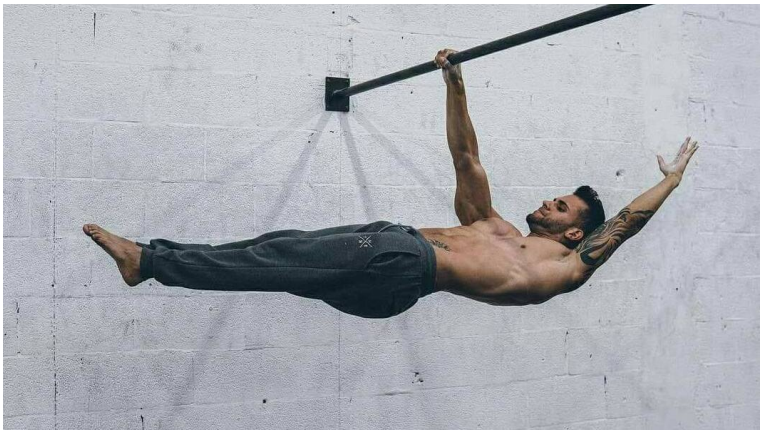

Figure 3.6: One arm front lever.

One arm handstand (see Fig. 3.7) is an asymmetric skill too, and it is considered a more challenging variant of the standard handstand. Unlike some other skills that require power and explosiveness, the one arm handstand demands exceptional balance, body control, and coordination. In this skill, the body is held in a vertical position, supported by only one arm while the other arm is typically extended out to the side or overhead. It is performed by a large range of athletes from various disciplines including hand balancers, gymnasts etc...

Figure 3.7: One arm handstand.

Planche (see Fig. 3.8) is a well-known and difficult pushing element in calisthenics that has many variants. In this project, only the "full" planche variant is being considered, which requires an enormous amount of strength. It is performed similarly to a maltese, but with the hands placed closer together, making it slightly easier. The planche is a primary skill in calisthenics, and it is included in this project due to its significance in the discipline and its presence in almost every competition.

Figure 3.8: Full planche.

3.2 Scraping considerations

We began by identifying a set of skills, and then proceeded to gather videos from various sources on the internet. The main ones have been: *YouTube*, *Instagram*, *TikTok*. All the videos collected have a few requirements:

1. **Number of people**: this number varies in the whole dataset, with almost every video featuring only one athlete. However, there are a few videos in which more than one person appears, although the athlete still covers the largest area of the video compared to the others.

2. **Perspective**: for better recognition, we have included multiple perspectives of every skill. However, it should be noted that not all perspectives have been covered, with a preference given to the frontal and lateral views.

3. **Video resolutions**: even if all video resolutions will be normalized in the next step, video with really low quality due to poor light or weak camera sensor have been cut off the dataset.

4. **Skills' variants**: the main focus is on the skill variants specified in the previous step. However, there are some videos in the dataset that show skills being performed in a more subjective way. For example, some Iron Crosses are held with an "l-sit". This approach tries to compensate for the lack of included variants.

3.3 Elaboration

All the scraped videos have different resolutions and different framerate. In the current step some normalization concepts have been applied. All videos have been converted to a resolution of **960x540(qHD)** and a framerate of **24 fps** without being stretched. The chosen resolution is commonly known as qHD (or quarter HD). It is a step up from standard definition (480p), and offers a 16:9 aspect ratio. While it is not as high as full HD (1080p), it is a good trade-off between quality and file size, making it a perfect choice in our case. The framerate has been chosen due to its standardization and its high versatility in the inference part. To modify resolution and framerate we have been used the Kdenlive software. Fig. 3.9 shows the user interface of Kdenlive:

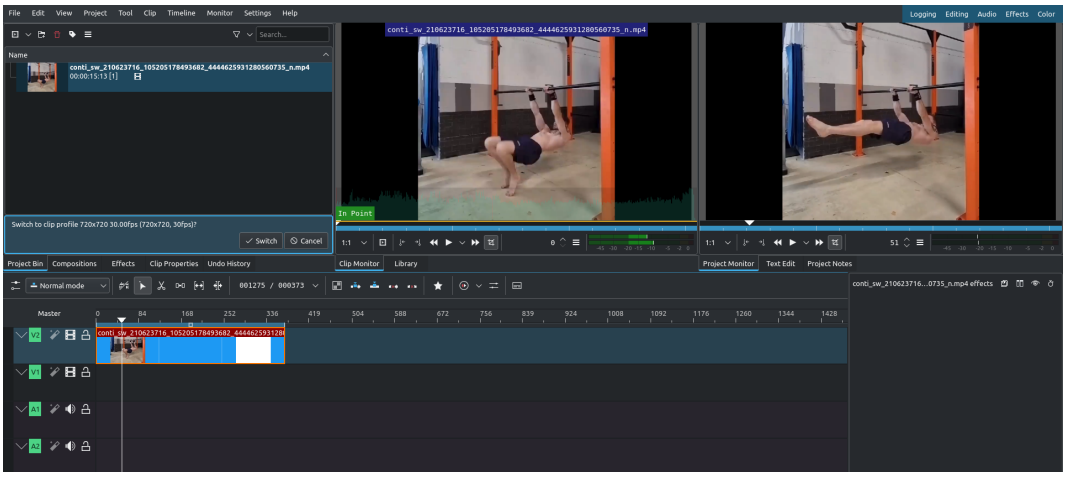

Figure 3.9: Kdenlive user interface.

In order to elaborate a video, the firts step is to create a new composition with the desidered resolution and framerate. Then for each downloaded video, the following operations have been performed:

1. The video is imported and moved to the video track in the timeline.
2. Video and audio have been ungrouped and the audio track has been deleted.
3. Video is cut to the main part where the skill is performed, including some pre/post skill movements.
4. The first skill's frame and the last one are then manually extracted.
5. Lastly, the video has been rendered to a proper folder.

As a result, each video features a single skill, with the option to include a 'none' portion to handle cases where skills are absent. Fig. 3.10 illustrates the potential structure of a video.

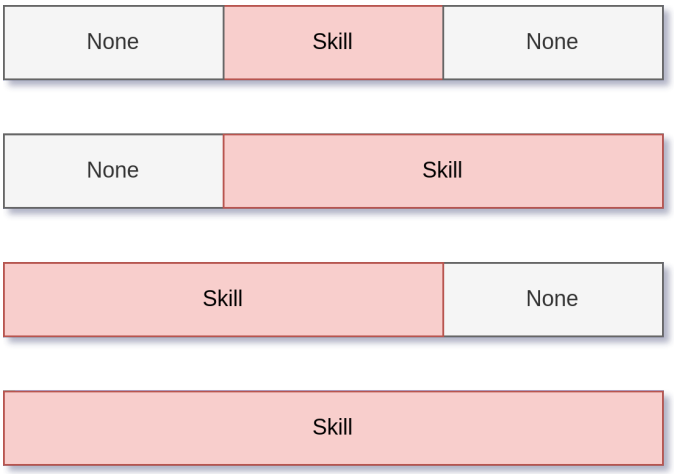

Figure 3.10: Possible video structure combinations.

Every rendered video, is considered "ready to work with" and its filename is structured as follows:

$$\{id\_skill\} + \{progressive\_number\} + ".mp4"$$

For istance, the twenty-fourth front lever on the dataset, is named: *fl24.mp4*.

3.4 Video dataset structure

We stored all the information about the videos in a table composed of 7 columns, as describing in the following:

- **hash_video** is a parameter that uniquely identifies the video. Considering the filename, it is a redundant field. It has been used to verify the integrity of the file and ensure that the content of the file has not been modified or corrupted. More specifically, it has been computed with MD5sum, a checksum function that generates a unique hash value based on the content of a file.
- **start_skill** tracks the start of the skill in the *seconds:frame* format.
- **end_skill** tracks the end of the skill in the *seconds:frame* format.
- **start_skill_frame** tracks the starts of the skill in the *frame* format.
- **end_skill_frame** tracks the end of the skill in the *frame* format.

- **id_skill** is the identifier of the skill, corresponding to the string used to represent the labels, in which the values are (pl, fl, oafl, oahs, mal, ic, flag, bl) + 'none' not written within the dataset video but subsequently integrated through a script.
- **id_video** is the video identifier, which will be used in the next phases to reference a particular video.

Fig. 3.11 illustrates a small portion of the dataset.

hash_video	start_skill	end_skill	start_skill_frame	end_skill_frame	id_skill	id_video
604cb2fd7ba98898f6dac368196b0767	00:06	05:00	6	120	oahs	oahs1
69587f17bf293fb295850d631598efb5	00:00	03:03	0	76	oahs	oahs2
9182446017b96389061b852ef34528af	00:22	08:15	22	207	oahs	oahs3
52de460b587e2ab486ca31c4f1a147f8	00:09	05:13	9	133	oahs	oahs4
53402beb50f9ef000698c89dfd5d380d	00:00	02:03	0	51	mal	mal2
91181ba79e8d9199f243eb1f9a8bddb3	01:17	03:12	41	84	pl	pl2
3843604fb9e44e75a78d05d9081865f3	00:14	02:02	14	50	pl	pl3
084342e0e2694770ba80ad78f5af4766	01:18	02:18	42	66	pl	pl4
9c28ece71dae6997712ed8f7b28dc3f9	02:17	06:01	65	145	pl	pl5
e38e98c5f2bb524e8752c634194043ae	01:03	03:08	27	80	mal	mal1
a0412179814461d9126f7b8ac993c73b	00:18	01:11	18	100	fl	fl1
914103e2904bb37fd1b3f183f71c667a	00:22	03:16	22	88	pl	pl1
6b6f2c42c5171f77ab56c145a84a0952	00:08	01:16	8	40	pl	pl6
7a6146cdf9e9b86e12502d8300b846a5	00:13	03:11	13	83	mal	mal3
370d9bce28aea6634cbb935ac7863d67	00:07	03:13	7	85	pl	pl7
10e8fe247c1bb05d1f5c493d69af1f8e	00:00	06:02	0	146	pl	pl8
c3706656aaef20ee19a1abbbb9cd8d30	00:00	03:01	0	73	ic	ic1
1a1449cc4d6cf8bd15a866102d61bcfa	00:00	04:04	0	101	pl	pl9
bcc670774e20f0b6f246981c556df551	00:09	02:03	9	51	mal	mal4
e944f7d5d899d5f7c2a232372abc8505	00:00	03:12	0	84	ic	ic2
34d0cf2fd42c387b9f7a11cb5c6a2618	00:00	03:10	0	82	ic	ic3
38245d627f51f12cda711c1a4823244f	00:16	05:20	16	140	pl	pl10
84f085ef4a928af173a848e40e779f4c	00:14	04:07	14	103	pl	pl11
ba9ca0fb7f77e497c592fde080d080ac	00:00	05:07	0	127	pl	pl12

Figure 3.11: Dataset of videos.

3.5 Dataset analysis

The whole dataset counts an amount of 573 videos. Till this point, the only classes considered are the 8 skills presented before plus a background class denoting the absence of any skill. The Table 3.1 reports statistics on the dataset.

	Skills								Total
	bl	fl	flag	ic	mal	oafl	oahs	pl	8
Video occurrences	65	105	24	53	95	61	82	88	573
Frame occurrences	9823	10177	2866	8596	9001	6231	11405	9629	68088
Percentages	11.34%	18.32%	4.18%	9.24%	16.57%	10.64%	14.31%	15.35%	100%

Table 3.1: Occurrences at the video and frame levels in the video dataset.

Fig. 3.12 represents a pie chart about the numbers of occurrences:

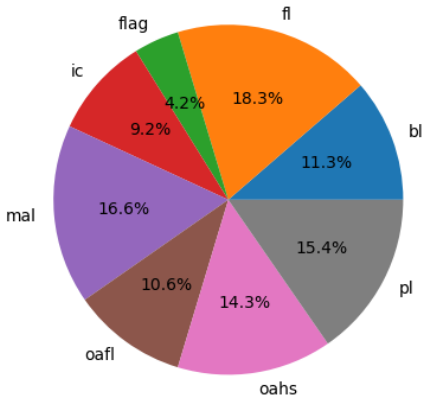

Figure 3.12: Pie chart illustrating the percentage distribution of video skill occurrences in the video dataset.

Some notes about the numbers:

- Flag is the least frequently occurring skill among the others. However, this is not a concern because flag is an easy skill for OpenPose to track, and it only requires one frontal perspective. This compensates for the relatively low number of videos that contain this skill.
- The same observation applies to the iron cross. Even though it appears at a low percentage, it can be observed from only two perspectives and is invariant to horizontal mirror transformation.

- Other skills, such as planche, maltese, back lever, or front lever, are more frequently observed due to their main role in the discipline, as well as their greater number of commonly used perspectives. Moreover, these skills are difficult to track. This last point will be better discussed in the next chapter.

Fig. 3.13 illustrates the occurrences of the videos in correlation with their durations.

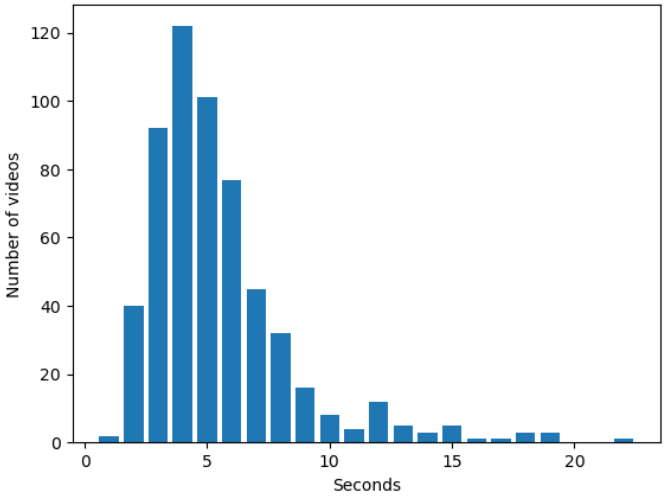

Figure 3.13: Histogram of video durations in seconds.

As can be seen, the largest amount of video has a duration that goes from three to six seconds, covering the 60% of the total videos. The video with the shortest duration is: *oahs45* with 0.833 seconds. While, the longest one is: *bl51* with 21.5 seconds.

Furthermore, Table 3.2, shows the average durations in seconds for each class.

	Skills							
	bl	fl	flag	ic	mal	oafl	oahs	pl
Average duration in seconds	6.29	4.03	4.97	**7.04**	3.94	4.25	5.79	4.55

Table 3.2: Average skill video durations in seconds.

Some considerations about the durations:

- The iron cross has the highest average value, primarily due to its origin as a skill in gymnastics competitions, where it is held for extended periods of time. Moreover, unlike other skills that involve dynamic movements related to them, the iron cross is characterized by its static position. Consequently, when executed, it has a longer duration of sustained holding.

- The maltese shows the lowest duration value, primarily attributed to the difficulty of the skill. As one of the most challenging skills to perform, it is typically held for a shorter duration compared to other skills. Furthermore, it is worth noting that the strongests athletes can incorporate dynamic movements such as Maltese press, Van Gelder, Zanetti, or dead maltese, this decrease the duration of the isometry.

- Front lever has the second lowest duration, but unlike the maltese, the reason behind this is the frequent combination of the skill with various dynamic elements. These additional elements include front lever pulls, front lever pull-ups, victorians, front lever raises, front lever to back lever transitions, and more. The addition of these dynamic movements decrease the duration of the isometry usually held.

Chapter 4

Proposed method

In this chapter, we will present the training and testing pipeline proposed in this thesis.

4.1 Training phase

This phase (see Fig. 4.1) involves the necessary steps to tackle the skill recognition task. We will explore the integration of OpenPose into our project to extract numerical data from video frames. Subsequently, we will utilize this dataset to develop an architecture specifically designed for skill classification.

Figure 4.1: Training pipeline.

4.1.1 OpenPose keypoints extraction

In order to extract spatial information about human body joints, we have utilized OpenPose on the video dataset. As previously mentioned, we chose to use the *BODY_25B* model which offers a well-designed joint connection layout and a great reliability. This model provides a total of 25 identifiable joints, which as described in Fig. 2.5.

For each joint, OpenPose gives a set of three values:

- **X-Coordinate**, the horizontal position of the joint within the image,
- **Y-Coordinate**, the vertical position of the joint within the image.

- **Confidence**: a confidence value indicating the reliability or certainty of the joint's detection. It represents the likelihood that the joint's estimated location is accurate. The confidence value is typically normalized between 0 and 1, where a higher value indicates a more reliable detection.

In this project, we have chosen to normalize the coordinates, constraining them to the range [0, 1]. The origin (0, 0) corresponds to the top-left corner, while the bottom-right corner is represented by (1, 1). By normalizing the coordinates, we ensure consistency and facilitate comparisons across different resolutions.

In the OpenPose configuration file, the parameter that allows to obtain the normalization is: *-keypoint_scale 3*. Two other relevant parameters that have been tuned are:

- ***number_people_max*** refers to the maximum number of individuals that OpenPose will attempt to detect and track. In our case it has been set to 1. This configuration ensures that OpenPose captures and tracks only one body in the video, even if multiple individuals are present. In the case of multiple people within the video, OpenPose typically prioritizes tracking the person who is most prominent or closest to the camera.

- ***net_resolution*** refers to the resolution at which the neural network processes the input image or video frames. It accepts a string value with the desidered resolution in format *width* $\times$ *height*.

  The first value '-1' indicates that the width of the input image or frame will be automatically adjusted while preserving the aspect ratio. After conducting several graphical comparisons, the second value has been set to 208, representing the desired height of the input image or frame. It is important to note that this value must be a multiple of 16.

The OpenPose processing generates two main outputs:

- A video file in '.avi' format, preserving the original resolution and framerate of the input video. Fig. 4.2 illustrates an instance.

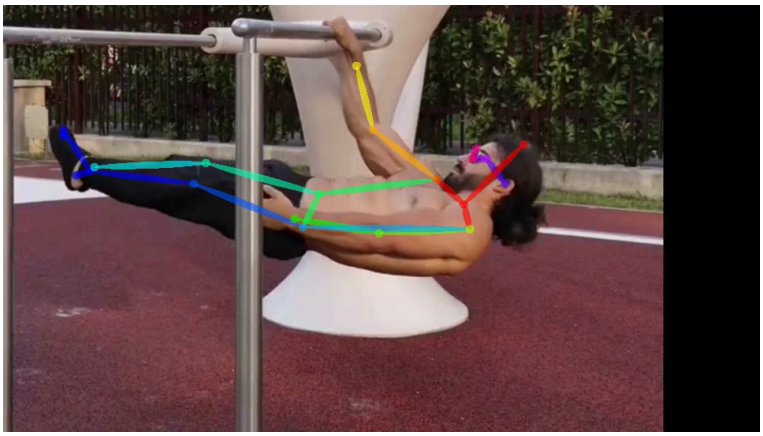

Figure 4.2: An instance of a frame in the output of OpenPose.

- Multiple JSON files, with the number of files equal to the number of frames in the video.

While the first component provides a visual representation of the processing, the JSON files serve as crucial information sources for further stages of the pipeline. Within each JSON file, two primary keys of the dictionary are of particular interest:

- ***person_id***: This field contains an integer value that serves as an identifier for each person detected in the video. If no joints of any person are recognized, this value will be '-1'.

- ***pose_keypoints_2d***: This field consists of 75 values ranging between 0 and 1, representing the spatial coordinates of each joint. These 75 values describe the x, y coordinates and the corresponding confidence level for each joint. All the values of the joints not found are set to zero. In the absence of detected joints, this field will be empty.

All the JSON files of every video are placed in separate folders, identified by the *video_name*.

4.1.2 Keypoints dataset

In this subsection, we will delve into the process of creating the numerical dataset that is used as input for the neural network to perform the classification task.

4.1.2.1 Dataset building

During this stage, we are effectively building the numerical dataset that will be utilized in the following phases. The dataset we aim to create will comprise a total of 78 columns, representing the following components:

- The 75 values which are derived from the OpenPose algorithm (25 joints x three values per joint).
- The name of the corresponding video. This identifier is assigned during the initial video dataset creation process and remains consistent throughout the succeeding stages of data processing and analysis.
- Another column includes the progressive number of the current frame being processed.
- Lastly, the dataset incorporates the *skill_id*, which serves as the class or label for the associated skills. This attribute is crucial for subsequent steps.

All the rows will contain every single frame of all videos. To achieve this objective, we have developed a Python script. The algorithm implemented in this script consists of several steps, which are explained in the following:

1. First of all, a CSV file is generated by compiling all the column features.
2. For each video in the dataset, the script iterates through all the JSON files, extracting the keypoints for each frame. These keypoints are then added to a temporary dataset, populating each row accordingly. If a JSON file contains an empty list of keypoints, indicating the absence of detected keypoints, the corresponding frame is excluded from the dataset. From the filename of each JSON file, the video name and current frame number are extracted.
3. To populate the *skill_id* field, a comparison is made with the previously built dataset. The process is simple: for each row in the current dataset, the script searches within the previous video dataset for the row with the same video name. Once a match is found, the current frame is compared with the start and end frame.

- If the frame falls within the range, the label is set to the *skill_id* of the video in the video dataset.
- If the frame falls outside the range, it indicates that the skill is 'none'. This is because each video represents only one skill (except for the 'none' category).

4. After writing each row of a specific video to the temporary dataset, a brief statistical analysis is conducted on the actual video. This analysis involves calculating the number of keypoints with a value of zero in the video and comparing it to the total number of keypoints.

   - If the percentage of the keypoints not tracked in the current video exceeds a predefined threshold, the video is discarded from further processing.
   - On the other hand, if the video has a relatively low number of zero keypoints, it indicates that it has a sufficient amount of keypoints for the nexts steps.

   The threshold for exclusion has been set at 50%. Videos that are excluded from the dataset are intentionally retained within the video dataset. This decision is based on two reasons. Firstly, future advancements in pose estimation may lead to the development of more accurate body tracking algorithms, capable of detecting a greater number of keypoints in these videos. Furthermore, by retaining these videos, we open up the possibility of their inclusion by lowering the threshold.

5. Once the video passes the previous check, all the rows representing frames are inserted into the keypoints dataset, which serves as the effective dataset used for training the model.

Here is a brief example of the script's output:

```
Processing the video... :  fl24
Total zero keypoints :  966
Total keypoints :  6300
Video frames :  84
The current video has been added to dataset!

Processing the video... :  mal18
Total zero keypoints :  2181
Total keypoints :  11325
```

```
Video frames :  151
The current video has been added to dataset!

...

Processing the video... :  oahs16
Total zero keypoints :  2145
Total keypoints :  13050
Video frames :  174
The current video has been added to dataset!

Total frames processed:  61397
Total videos processed:  573
Total excluded videos:  15
Excluded videos:  ['fl41', 'fl84', 'oahs1', 'oahs47',
'oafl13', 'fl81', 'oahs59', 'flag6', 'mal63', 'oafl7',
'fl6', 'oahs10', 'oahs44', 'flag17', 'oahs65']
Excluded videos frames:  804
```

Table 4.1 illustrates the number of videos pre and post processing. Table 4.2 reports the video and frame occurrences after the processing phase, while Fig. 4.3 reports proportions in a pie chart.

	Skills								Total
	bl	fl	flag	ic	mal	oafl	oahs	pl	
Pre processing occurrences	65	105	24	53	95	61	82	88	573
Post processing occurrences	63	102	22	53	93	59	73	88	553
Discarded videos	2	3	2	0	2	2	9	0	20

Table 4.1: Video dataset occurrences pre-post processing.

	Skills									Total
	bl	fl	flag	ic	mal	none	oafl	oahs	pl	
Video occurrences	63	102	22	53	93	-	59	73	88	553
Frame occurrences	7465	7362	1282	7069	6932	12852	4699	6491	7403	61555
Percentages	12.12%	11.96%	2.08%	11.48%	11.26%	20.87%	7.63%	10.54%	12.02%	100%

Table 4.2: Video and frame dataset occurrences after processing.

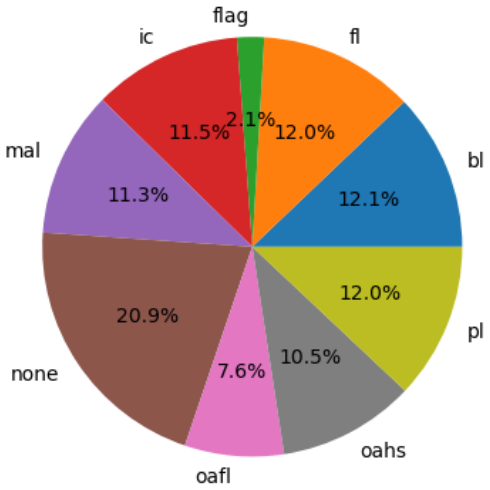

Figure 4.3: Pie chart illustrating the percentage distribution of frame skill occurrences in the dataset.

The chart clearly illustrates a significant number of 'none' frames, which is expected as there are numerous body positions both pre-skill and post-skill that should be identified as the absence of skills. This observation highlights the importance of recognizing these non-skilled positions in order to provide comprehensive skill detection.

On the other hand, the dataset contains a relatively small percentage of flag videos, which is not a significant concern. As mentioned earlier, flag detection is an easy task for OpenPose, ensuring optimal performance in recognizing this particular skill.

4.1.2.2 Dataset split

After discussing the creation of the keypoints dataset that will be used to train our model, we proceed with two dataset operations, starting with the splitting of the dataset into two smaller subsets. This is an essential step in machine learning model development, where the original dataset is divided into distinct training and testing sets. An important observation is made during the split process. When performing the dataset split, data has been divided based on the *video_name* rather than on a frame level basis. This choice allows to avoid potential issues related to overfitting the model due to the presence of similar neighboring frames, which would lead to an "overlap" between training and test set. By splitting based on *video_name*, we ensure that frames from the same video are present in either the training or test set but not both. The dataset splitting operations are performed using the Scikit-Learn library. Scikit-Learn provides the *GroupShuffleSplit* function, which allows us to split the dataset into training and test sets based on the previous strategy illustrated.

The sizes are set as follows: Training Set: 80% and Test Set: 20%. The training set offers a larger volume of data for the model to learn patterns, while the test set serves as a benchmark to evaluate performance on unseen data, making it smaller in size.

4.1.2.3 Data augmentation

To prevent overfitting, we perform data augmentation, a technique commonly used in machine learning to artificially increase the size of a training dataset by applying various transformations to the existing data. More specifically, we introduce new samples by applying operations such as rotations, translations, scaling, cropping, flipping, and adding noise or distortions to the original data, which are assumed to keep the class of the processed frame intact. In our project, a custom function has been developed to perform two specific transformations on the data:

- **Horizontal mirroring**: The first transformation, involves flipping all elements in the dataset horizontally. For every row in the dataset, the x-coordinate of every joint is modified by subtracting it from 1 (1 - x). This operation is not applied to the y-coordinates or the confidence values. If a coordinate has a missing value (0), it remains unchanged.

- **Shifting** This second one, introduces small displacements to the keypoints, simulating variations in their positions. It consists by generating a random value within the range of (-0.10, +0.10); this value is then

applied to all the x and y coordinates in the dataset. The confidence value remains the same. The shift factor is added to the respective coordinates, and if the resulting value exceeds 1 or falls below 0, it is set to 0 to ensure valid coordinate values.

The choice of which transformation to apply is determined randomly with a 50% probability. The transformation is applied to each row of a duplicate of the training set and then merged with the original training set. By applying these data augmentation techniques, we effectively double the size of our training set and introduce variations in the keypoints' positions. This helps our model generalize better and learn robust representations of the keypoints, leading to improved performance during training and inference.

4.1.3 Training of the Multilayer Perceptron

In this subsection, we present the architecture of the multilayer perceptron used to classify each frame, providing a comprehensive examination of all its design details.

4.1.3.1 Network architecture

The multilayer perceptron (MLP) implemented for this project is designed specifically for skill classification, leveraging a set of 75 input keypoints. The model architecture is shown in Fig. 4.4 and consists of three hidden blocks, each comprising a linear layer followed by an activation function. The output layer is composed of nine neurons, representing the different skill classes to be classified, including the background class.

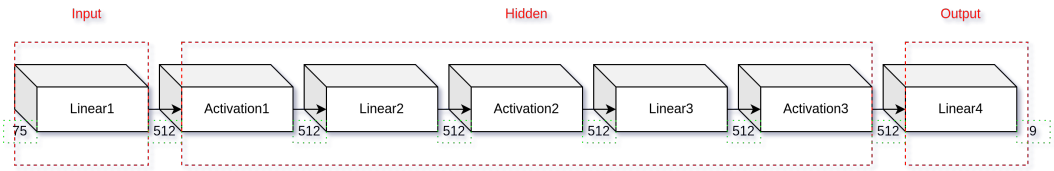

Figure 4.4: Multilayer Perceptron architecture composed of an input layer, three hidden blocks, and an output layer.

The input layer of the MLP receives as input the 75 keypoints of a given example from the dataset. These keypoints serve as the fundamental signal from which subsequent layers of the MLP can learn and identify the patterns within the data. In the hidden layers, the linear layer of the MLP performs a matrix multiplication operation between the input keypoints and a set of

learnable weights. This process is followed by the addition of biases, which further contributes to the learning process. The output of the linear layer is then passed through an activation function, specifically the *LeakyReLU* [4] in this architecture, represented as a diagonal line with a slope of 1 for positive inputs, and a smaller slope (defined by a small positive constant) for negative inputs, as shown in Fig. 4.5.

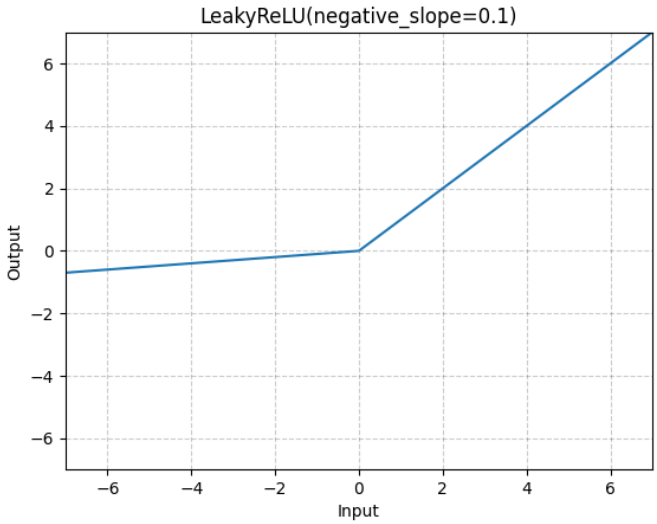

Figure 4.5: LeakyReLU's activation function representation.[1]

This activation function helps alleviate the "dying ReLU" problem where neurons can become stuck in a state of inactivity due to negative inputs. By allowing a small gradient for negative values, it enables better information flow during training, especially in cases where the ReLU function might be too aggressive in zeroing out negative inputs. Furthermore, it introduces non-linearity to the network, allowing it to model more complex and nonlinear relationships within the data. To enhance the capacity of the MLP to extract features, each hidden layer is composed of 512 neurons. By utilizing hidden layers with a significant number of neurons, the model becomes more capable of learning and representing complex patterns and relationships within the dataset. The MLP employs three hidden blocks, each consisting of a linear layer followed by the *LeakyReLU* activation function. Finally, the output layer of the MLP consists of nine neurons, matching the number of skills to be recognized (including the background class). The output layer applies a

[1]Image source: `https://pytorch.org/docs/stable/generated/torch.nn.LeakyReLU.html`

linear transformation to the activations received from the last hidden block. This transformation enables the model to produce probabilities for each skill. The class with the highest probability is then predicted as the output of the model.

4.1.3.2 Optimizer and Loss function

During the training process, the MLP utilizes the *Adam* optimizer with a learning rate set to 0.0001. The Adam optimizer combines the benefits of both the *AdaGrad* and *RMSProp* optimizers by adapting the learning rate based on the gradient's first and second moments. This adaptive learning rate contributes to faster convergence and improved generalization of the model. *Adam* also incorporates a mechanism for bias correction to address initialization biases. These corrections ensure accurate estimations, particularly in the early stages of training. To prevent overfitting and encourage generalization, the Adam optimizer employs *weight_decay* regularization which is set to $1e-4$ and it introduces a penalty term to the loss function, discouraging large weight values and reducing model complexity. The loss function utilized in this MLP is the *Cross-entropy* [4]:

$$L_{CE} = -\sum_{i=1}^{n} t_i log(p_i)$$

for n classes, where t_i is the truth label in the form of a one-hot vector and p_i is the Softmax probability for the i^{th} class.

The *Cross-entropy* loss measures the dissimilarity between the predicted probability distribution and the true distribution of the target labels. By minimizing the *Cross-entropy* loss, the model learns to assign higher probabilities to the correct skills and lower probabilities to incorrect ones. This enables the MLP to effectively classify and recognize the desired skills based on the input keypoints. The utilization of the *Cross-entropy* loss function is a common and effective approach for training classification models.

4.1.3.3 Model's training

During the training process, the model utilizes a DataLoader to load the training data in batches. This batch processing strategy enhances the efficiency of the model by performing computations on multiple data samples simultaneously. The size of each batch is set to 512 data samples, allowing for efficient parallel processing. The model makes weight and bias updates by computing the average loss for each batch. This method ensures more stable gradients and accelerates convergence during training. By repeatedly

processing multiple batches and adjusting the model's parameters, the MLP becomes more accurated in recognizing patterns and extracting meaningful features from the input keypoints. The MLP has been trained for 100 epochs, where an epoch denotes a complete iteration through the entire training dataset. Within each epoch, the model handles individual batches, calculates gradients, and updates weights and biases accordingly. This iterative process repeated over 250 epochs enables the MLP to refine its parameters and optimize performance.

4.2 Testing phase

In this section, we will explore the test pipeline, which consists of the steps illustrated in Fig. 4.6:

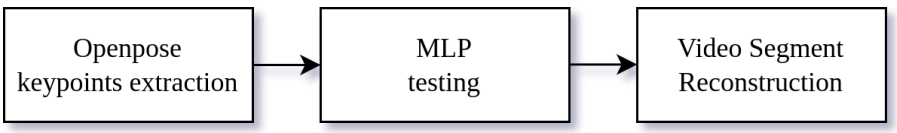

Figure 4.6: Testing pipeline.

First, we will incorporate OpenPose into this pipeline. Then, we will proceed to the testing phase of the model. Finally, we will discuss the development of an algorithm to reconstruct the video timeline using the obtained frames.

4.2.1 Keypoints extraction and test set building

During this test phase, the same process as the training phase is applied. To begin, OpenPose is employed to extract the keypoints from all the frames of the videos achieved exclusively for the test set. Once the keypoints are extracted, the testing set is constructed using the same methodology as in the training phase. The 75 keypoints extracted from each frame serve as input features for the model. Additionally, the *skill_id* column, is utilized as the label for the testing phase.

By incorporating the features and mapping them to their respective labels, the testing dataset is prepared for evaluation. This enables the model to make predictions based on the input keypoints and assess its performance in accurately identifying and classifying the desired skills or actions. By following this process, the test dataset captures the relevant information from all the frames in the test video, enabling the model to make predictions and

evaluate its performance in an accurate way. The uniformity between the construction of both the training and testing sets is essential to ensure fair and unbiased evaluation of the model's performance.

By employing analogous data preparation methodologies, we decrease the risk of introducing any biases or unwanted factors that may affect the model's accuracy and generalizability.

4.2.2 Multilayer Perceptron testing

After training the model with a specific architecture and the selected parameters and preparing the test set, the testing phase is conducted to evaluate the model's performance on unseen data.

To begin, the test set is loaded using the same method as in the training phase, employing a DataLoader to process the data in batches. However, a slight modification is made to the batch size. In this case, the batch size is set to half the size of the original batch used during training (256). During the testing phase, the model is applied to the input data, and the corresponding output is obtained to calculate the loss function. The model's predictions are extracted using PyTorch functions such as *torch.max*.

Several analyses are conducted during the testing phase, they will be further elaborated and discussed in an upcoming chapter, where the model's performance will be examined and evaluated.

4.2.3 Video Segment Reconstruction algorithm

So far we have primarily focused on classifying individual video frames. However, to reach our goal of reconstructing the video timeline and rectifying inaccurate predictions within a sequence, we need an algorithm that satisfies these specific requirements. For this purpose, we have designed an algorithm that can be subdivided into three essential parts, listed below and discussed in detail in the following subsections.

1. **Sliding Window Mode Extractor**
2. **Filtering and noise removal**
3. **Timeline reconstruction**

4.2.3.1 Sliding Window Mode Extractor

This is the first step of the algorithm. Its input is a DataFrame containing predictions. During the testing phase, a list of string-type labels is retrieved, with a length equal to the number of frames in the video. The list is converted into a DataFrame. This conversion is done to take advantage of the efficiency of data manipulation, filtering, and analysis using built-in functions and methods. This allows for faster and more intuitive operations.

An important hyperparameter, called *window_size*, is tuned in this step. To apply this algorithm, the total number of frames in the video shall be greater than the *window_size*. If the total number of frames is less than the *window_size*, it is automatically set to the total number of frames. The value of the *window_size* has been determined through analysis that will be discussed in the next chapter.

Consequently, all label values within the range of *[i, window_size-1]* with $i = 0$ *initially*, are taken into consideration. The algorithm follows these steps:

- If there is no dominant value, meaning that all the labels have the same frequency, the *window_size* is increased by one, and this step is repeated by recalculating the mode for the new set. This process is iterated until a dominant value is found.
- If the dominant value is found, a new list is built with the following data:
  - **current_mode**: The string value of the mode.
  - **min_index**: The first occurrence of the value in the window, with global indexing.
  - **max_index**: The last occurrence of the value in the window, with global indexing.

  Therefore, this temporary list will looks like:

  [current_mode, min_index, max_index]

  And it is inserted inside a *modes* list. Then, the algorithm is iterated increasing the counter variable adding a *sample_step* to it: $i = i +$ *window_size - 2* and restoring the *window_size* to the starting value. This logic is repeated for the rest of the labels in the list, having all the mode values being stored in the modes list.

At the end of this step, a list of lists is returned, which has a smaller length than the original list. It is important to highlight that this operation is lossy, as it reduces the length of the data and creates an approximation of the original information.

The following code, illustrates the function that implements the described algorithm.

```
#Default window_size value set to 13
def windows_mode(raw_predicted, window_size=13):
    modes = []
    temp_mode = []
    ws = window_size
    i = window_size
    sample_step = window_size-2
    while i <= len(raw_predicted):
        values = raw_predicted[i-window_size:i].iloc[:, 0]
        current_mode = values.mode()[0]
        if len(set(values)) == window_size:
            window_size += 1
            i+=1
        else:
            min_idx = values[values==current_mode].index.min()
            window_size = ws
            i += sample_step
            max_idx = values[values==current_mode].index.max()
            if i >= len(raw_predicted):
                max_idx = len(raw_predicted)-1
            temp_mode = [current_mode, min_idx, max_idx]
            modes.append(temp_mode)
    return modes
```

4.2.3.2 Filtering and Noise Removal

The second step involves cleaning the input list, *modes*, obtained from the previous step. This step is based on the understanding that within a sequence of frames, it is unlikely for a pose to change multiple times. If such a situation occurs, it suggests that there is an incorrect pose value in the sequence. The algorithm leverages the observation that different poses should have multiple contiguous elements in the list. To identify and remove these incorrect pose values, an iterative algorithm has been developed.

The algorithm handles five distinct cases, each addressing a specific index in the list. However, the underlying logic remains the same across all cases. The algorithm identifies sequences of a certain skill and checks if they are interrupted by some skills that does not repeat in the subsequent elements. This interruption is considered noise and is removed from the list. By ap-

plying this filtering mechanism, the algorithm aims to eliminate inconsistent pose values and improve the overall accuracy of the reconstructed timeline.

An important assumption in this step is that the length of the modes list should be at least 4. If the length is lower, the list is passed to the next step without applying the filter procedure.

Similar to the previous step, an iteration variable, denoted as p, is used in this step. The elimination of a certain noise skill, is performed by replacing that value with the mode of a restricted range around the current value. There are some index position to handle carefully:

1. If p is the first element of the list, the value at index p is replaced with the mode of the first four elements in the list.

2. If p is the second element of the list, the current value is replaced with the mode of the values at indices *p-1, p, p+1* and *p+2*.

3. If p is the last element of the list, the current value is replaced with the mode of the last four values in the list.

4. If p is the second-to-last element, the current value is replaced with the mode of the values at indices *p-2, p-1, p* and *p+1*.

5. For any other index p in the list, the value at index p is replaced with the mode of the four closest elements (two on the left and two on the right) in addition to itself.

This step returns the *modes* list, cleared and ready for reassemblage. The length of the list remains unchanged. Here the code of the *filtering* function:

```
1 def filtering(pointer, patch_mode, modes, index1, index2,
      index3, index4, index5=None):
2     patch_mode.append(modes[index1][0])
3     patch_mode.append(modes[index2][0])
4     patch_mode.append(modes[index3][0])
5     if len(set(patch_mode)) == len(patch_mode):
6         patch_mode.append(modes[index4][0])
7         if(index5 != None):
8             patch_mode.append(modes[index5][0])
9     moda_ = mode(patch_mode)
10    modes[pointer][0] = moda_
11    return modes
```

And the code of the *noise_removal* function:

```
def noise_removal(modes):
    if len(modes) > 3:
        for p in range(0, len(modes)):
            patch_mode = []
            if p == 0:
                filtering(p, patch_mode, modes, p, p+1, p+2,\
                p+3)
            elif p == 1:
                filtering(p, patch_mode, modes, p-1, p, p+1,\
                p+2)
            elif p == len(modes)-2:
                filtering(p, patch_mode, modes, p-1, p, p+1,\
                p-2)
            elif p == len(modes)-1:
                filtering(p, patch_mode, modes, p-2, p-1, p,\
                p-3)
            else:
                filtering(p, patch_mode, modes, p-1, p, p+1,\
                p-2, p+2)
    return modes
```

4.2.3.3 Timeline Reconstruction

This last step has the goal of returning the final list with a length equal to the skill segments present in the video. It also returns a second list that is equal to the first one but with the lengths of the skills in seconds. The mechanism behind the reassemblation of the various segments is the merge between the same elements in the *modes* list cleaned. Considering the *modes* list as a list with sublists in it, we have the following type of elements:

$$
\begin{aligned}
&[[\text{skill}_0, \text{start\_frame}_0, \text{end\_frame}_0],\\
&[\text{skill}_1, \text{start\_frame}_1, \text{end\_frame}_1],\\
&\ldots\\
&[\text{skill}_n, \text{start\_frame}_n, \text{end\_frame}_n]]\\
&\\
&\text{with } n = \text{length of the modes list}
\end{aligned}
$$

We considered the following merging criteria:

1. If two adjacent elements have the same *skill_id*, it means that a merge can be done. This involves extracting the first *start_frame* when the skill occurs and the last *end_frame* where the skill appears in sequence,

while discarding the intermediate values. This merging process is iteratively repeated for the entire list, merging all consecutive elements that have the same *skill_id*.

2. If two sequential elements have different *skill_id*s, it indicates that they belong to different segments. The *end_frame* of the first element is considered as the final frame reference for the first skill, while the *start_frame* of the second element represents the beginning point of the second skill.

   There may be a case where the difference between $(start\_frame_{n+1})$ and (end_frame_n) is greater than 1. This indicates the presence of a noise gap between two skill segments, potentially caused by incorrect predictions from the machine learning model. As we have no concrete skill reference in that gap, we need to fill it in an artificial way. The method used is as follows:

   Considering the nth skill and the *(n+1)*th skill: If the first reference of the second skill occurs at frame x, it means that before that frame, the skill wasn't recognized. Thus, it would be inappropriate to fill the gap with the *skill_id* of the $n + 1$ one. Therefore, we fill that gap with the *skill_id* of the nth skill, based on the fact that a few frames belong to it, observing the segment based on the previous actions.

This leads to the creation of a temporary list: *[skill, start, end]*, which is then appended to the final list. At the same time, the duration of the segment is converted to seconds and added to a separate list that contains the durations of all the segments in seconds. The output of the model will be the list of segments in frames and their durations in seconds.

The following code describes the current phase.

```
def timeline_reconstruction(modes):
    output = []
    output_s = []
    j = 0
    breakp = False
    while j < len(modes)-1:
        skill = modes[j][0]
        if j == 0:
            start = 0
        else:
            start = modes[j][1]
        while j < len(modes)-1 and modes[j][0] == skill:
            j += 1
            if j == len(modes)-1:
                end = modes[j][2]
                breakp = True
        if breakp == False:
            end = (modes[j][1]-1)
        output.append([skill, start, end])
        seconds = ((end+1)-start)*(1/24)
        output_s.append([skill, seconds])

    return output, output_s
```

4.2.3.4 Instance of use

To execute the algorithm, it is necessary to invoke a function that encapsulates all the previously discussed phases.

```
def vsr_algorithm(raw_predicted, windows_size=12):
    #Input automatically converted in Dataframe
    if type(raw_predicted) == list:
        raw_predicted = pd.DataFrame(raw_predicted)
    #Insufficient frames case handler
    total_frames = len(raw_predicted)
    if total_frames < windows_size:
        windows_size = total_frames
        return raw_predicted
    modes = windows_mode(raw_predicted, windows_size)
    modes = noise_removal(modes)
    output, output_l = timeline_reconstruction(modes)
    vsr_predicted = []
    for i in range(0, len(output)):
        for j in range(output[i][1], output[i][2]+1):
            vsr_predicted.append(output[i][0])
    vsr_predicted = encoding(vsr_predicted)
    return vsr_predicted, output, output_l
```

Invoking this function, a third element is returned. This element is a list of the same length as the original input list. It contains all the labels that have been processed by the algorithm, converted in integer values. To do this, an additional function called *encoding* has been implemented. This function converts the string labels into corresponding integer values, maintaining the same order as the labels used during the training and testing phases of the model. The effectiveness of the steps listed is illustrated in Fig. 4.7. The algorithm has been applied to a video and the steps' results are then reported.

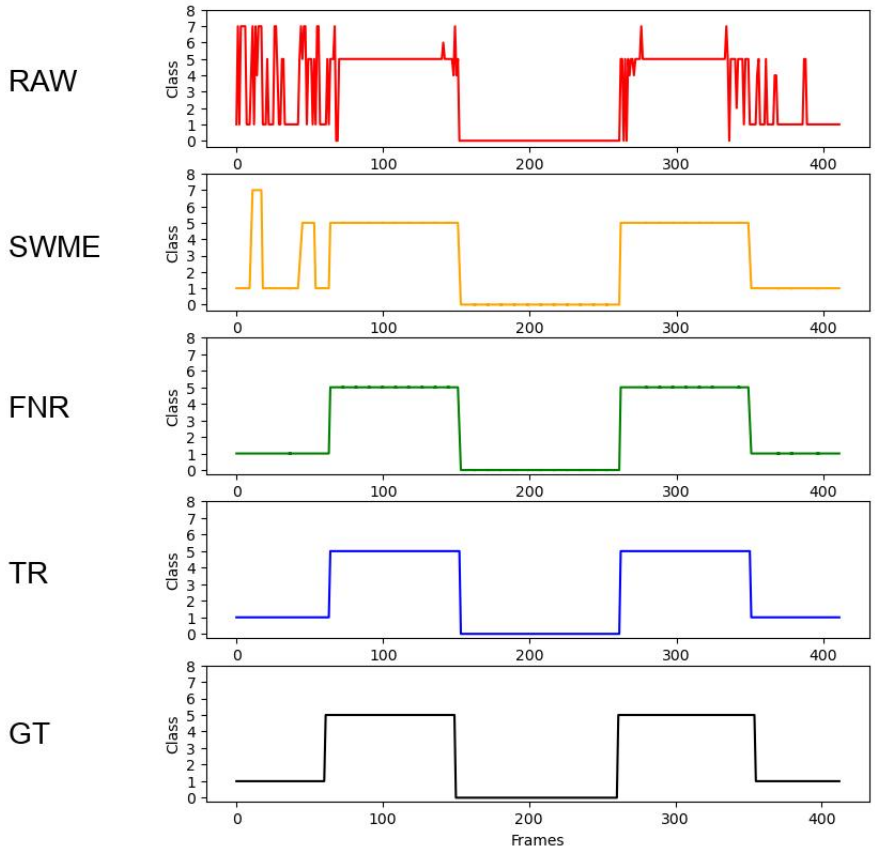

Figure 4.7: The representation of the VSR algorithm steps is as follows: The first plot illustrates the predicted raw labels, the second one shows the application of the SWME step, the third one demonstrates the effect of the FNR, the fourth one displays the reconstructed timeline. Finally, the ground truth timeline is presented at the end.

Chapter 5

Results and Analysis

In this chapter, we present and analyse the results of the experiments, providing detailed explanations, visual plots, and images to motivate the choices made throughout the process.

We will start by exploring OpenPose and analyzing its strengths and weaknesses in terms of human pose tracking. Next, we will delve into the multilayer perceptron (MLP) structure, focusing on the final configuration and conducting benchmark tests by varying different architectural aspects. This in-depth examination will show the performance and effectiveness of the MLP in this project. Furthermore, we will perform a brief analysis to evaluate the accuracy achieved. Following that, we will introduce the video segment reconstruction algorithm developed within this project and evaluate its performance by varying the *window_size* parameter. We will proceed to compare the results of the proposed approach with the relevant works discussed in [3]. This comparative analysis will offer further valuable insights into the performance of our algorithm.

5.1 OpenPose analysis

In this section, we will examine the OpenPose configuration selected for constructing the dataset. Subsequently, we will conduct a comprehensive analysis of the various scenarios in which the pose estimator performs effectively or encounters limitations, and comment on those cases where the accuracy of the model is not satisfactory.

5.1.1 Net resolution comparison

This parameter was introduced in the previous chapter, with a specific value of '208' determined through visual inspection. To identify the optimal value, a bash script was developed to test various resolutions ranging from 64 to 528 with a step size of 16 in a subset of the dataset videos. Fig. 5.1 illustrates the type of analysis performed on a particular video, namely *pl26*.

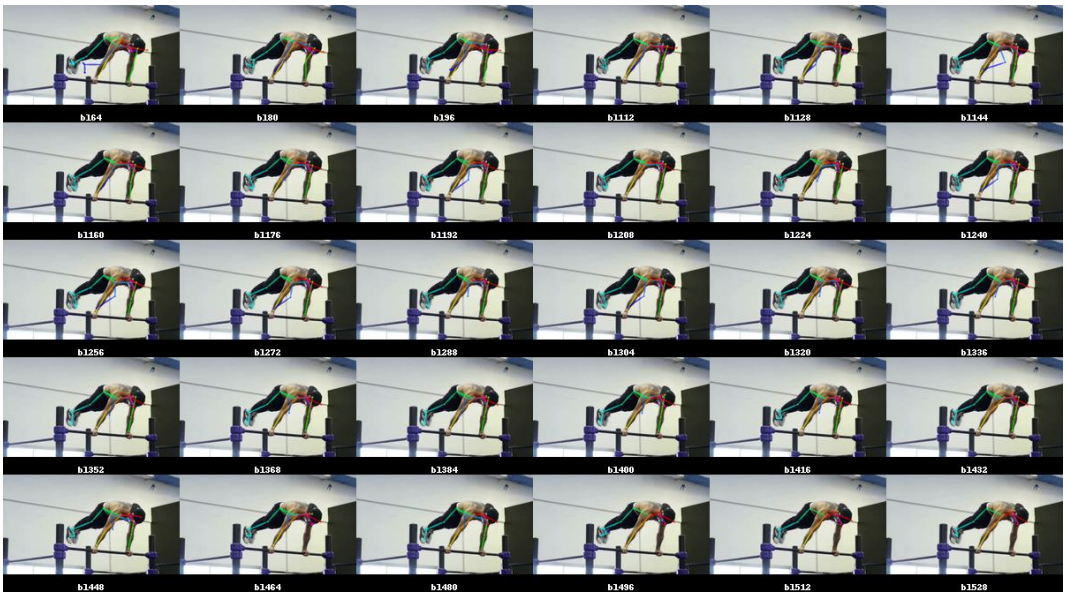

Figure 5.1: The net resolution matrix consists of tracking results obtained by gradually incrementing the *net_resolution* value over rows and columns.

After conducting a closer examination, it was observed that the optimal values fall within the range of [192, 224]. Consequently, greater attention was directed towards these values. A more detailed analysis was performed on these specific values.

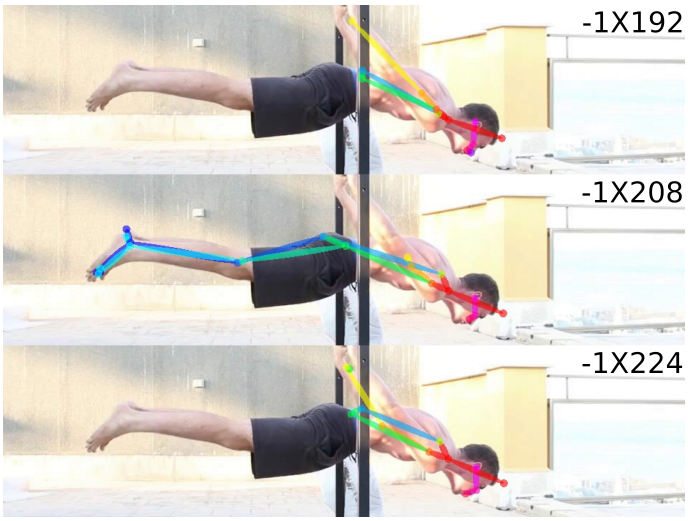

Figure 5.2: Restricted net resolution values comparison.

Fig. 5.2 illustrates an example of different parameter values applied to the same frame. As observed, the middle frame has better results compared to the others. In the second image, both the legs and feet are fully recognized, whereas they are not correctly estimated in the other two frames. Although the first image shows an higher precision on arm estimation; overall, the 208 value demonstrates better performance across the analyzed video samples.

5.1.2 Body pose estimation considerations

In this subsection, we will delve into visual considerations that shed light on the factors influencing the performance of OpenPose body pose estimation. Two crucial elements are examined, namely:

1. The varying levels of difficulty in recognizing the pose of different types of skills and their association with specific body parts that are relatively easier or harder to track.

2. The impact of body-background color contrast and the influence of the lighting conditions present in the scene.

By exploring these elements, we aim to gain insights into the performance dynamics of OpenPose and its sensitivity to various visual factors.

5.1.2.1 Skills and body recognition

Starting from the skills, we will examinate examples from every class, making some observations about the body positions and the joints.

Back lever has a good pose recognition rate, the whole instances cover approximately 11% of the entire dataset. The most common viewpoints used to film these skills have been included, while others such as frontal, top-down view, or rear view are excluded due to their infrequent usage in videos. In this subset, various analyses have been conducted to draw positive and negative observations:

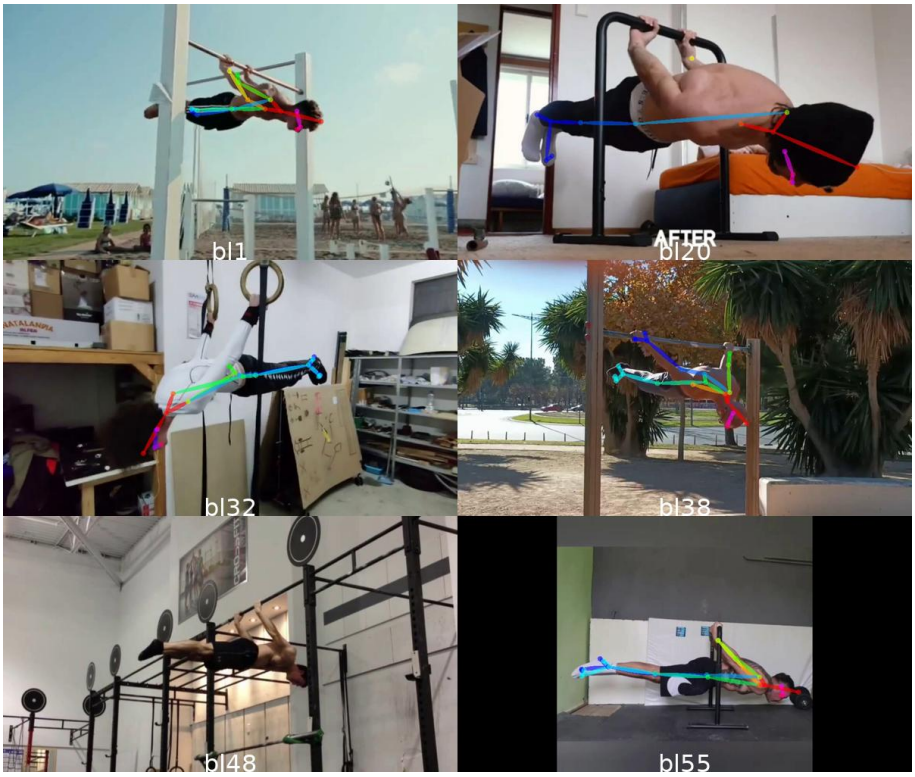

Figure 5.3: Back lever istances.

In Fig. 5.3, we can observe several positive examples such as *bl32*, *bl38*, and *bl55*, where OpenPose achieves a high level of tracking precision. However, in the case of *bl1*, there is a vertical bar that causes an interruption in the body, resulting in untracked feet. It is important to note that this is not a general representation of the entire video, as in neighboring frames, the connection between the legs and feet is established. Furthermore, *bl20* demonstrates a challenging tracking scenario due to the underexposed right arm and its perspective, which poses a difficulty for OpenPose. On the other hand, *bl32* represents an average situation where the body swings around the axis of the rings, causing some flickering. This is partly attributed to the poor contrast between the t-shirt and the wall. We will delve into this is-

sue further in a separate paragraph. Lastly, *bl48* illustrates a situation where OpenPose fails to recognize any keypoints. This can be attributed to missing limb or head information, which makes it impossible to establish connections with other body parts.

Front lever is the skill with the highest occurrence in the dataset, covering 18% of the whole dataset. It has a better level of recognition than back lever, due to the inverted body position, exposing essential body part like the head. The same consideration made for the back lever are valid here too. The subset extracted for this analysis is shown in Fig. 5.4.

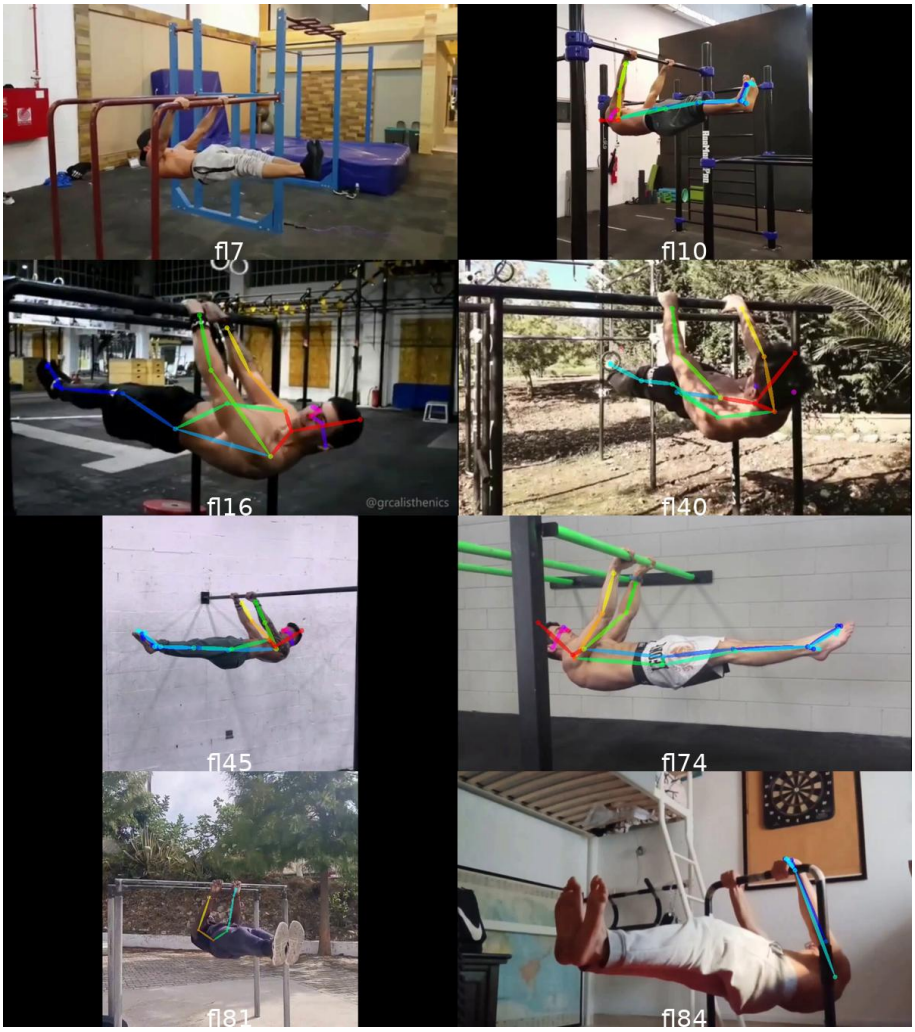

Figure 5.4: Front lever istances.

Among the positive examples, *fl16*, *fl40*, *fl45*, and *fl74* show successful body pose estimation despite differences in perspective. In the case of *fl10*, even if there is a well-tracked frame, there is noticeable flickering in the head and left arm, suggesting a connection between the two. When either of these body parts is not accurately localized, it obstruct the PAF principle's ability to establish connections. Two instances of poor tracking that further emphasize this concern are *fl81* and *fl84*, where the face is obscured by an arm, causing issues for OpenPose. Finally, *fl7* serves as evidence that when both parts are not visible, no keypoints can be effectively tracked.

Human Flag covers 4% of the whole dataset, despite the low occurrences of this skills, it is well tracked. It has two involved viewpoints, the frontal and the rear view. Both gives good results, but the frontal one performs better.

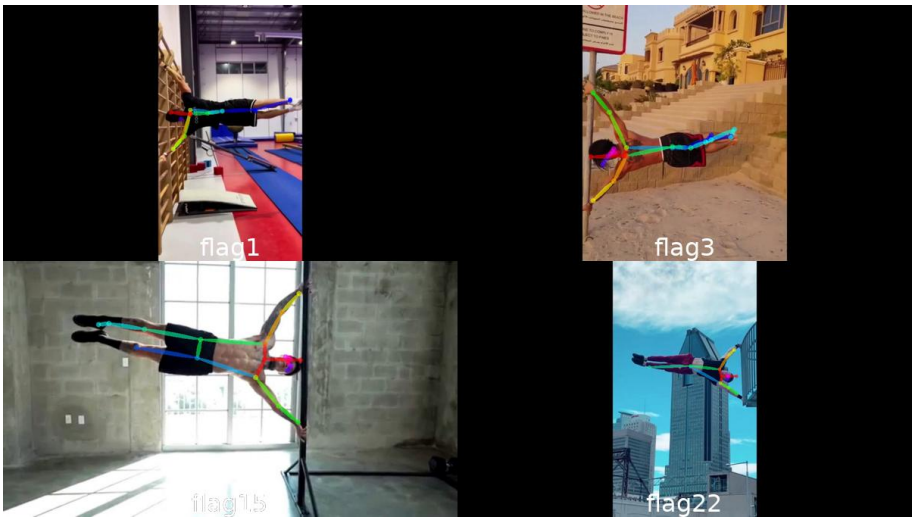

Figure 5.5: Human flag istances.

As can be noted in Fig. 5.5, *flag3* and *flag15* exhibit a high level of tracking accuracy, representing positive samples in our analysis. On the other hand, *flag1* and *flag22* demonstrate tracking performance that covers most of the body, except for certain limbs. In the first case, the absence of face information may be the cause for the incomplete tracking of the arm, while the second one benefits from the availability of face information, resulting in an overall better tracking.

Iron cross covers approximately 9% of the entire dataset. It is the best tracked skill. Its exceptional accuracy can be attributed to several key factors. Firstly, the pose has only two commonly used perspectives: the frontal view and the rear view similar to the previous one. These perspectives allow for a optimal analysis of the body's joints ensuring clear visibility without any conflicting elements. However, the same considerations have been made for the human flag pose, where OpenPose shows worse tracking performance.

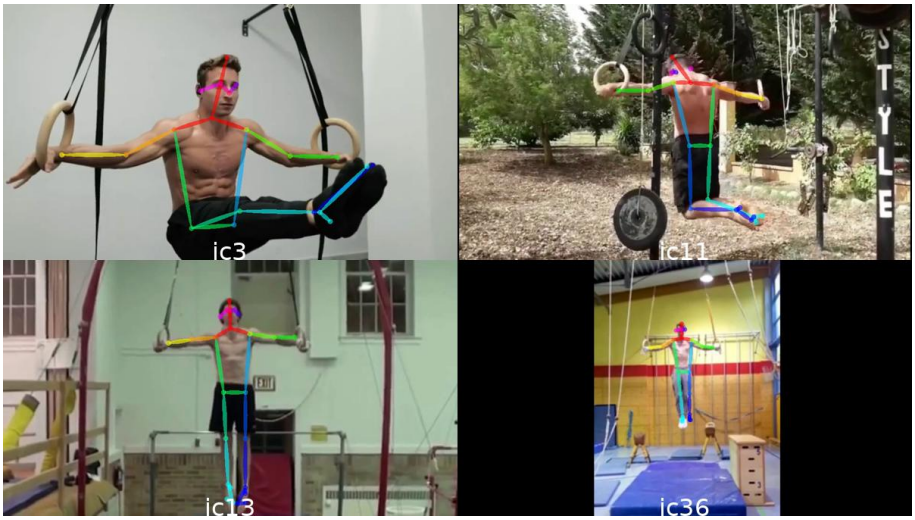

Figure 5.6: Iron cross istances.

The analysis of the subset showed in Fig. 5.6 reveals that the skill being considered, including variants such as the l-sit or the half lay position of the legs, is accurately tracked by OpenPose. From this analysis, two important observations can be made:

- OpenPose exhibits a stronger predisposition to tracking vertical human body positions, particularly those that are not upside down. This can be attributed to the higher frequency of vertical poses used during the model training phase. The model's familiarity with such poses contributes to performance by accurately tracking them.
- It appears that OpenPose achieves greater accuracy when the individual joints of the body exhibit a well-distributed spatial arrangement within the image. This favorable distribution facilitates the work of Part Affinity Fields, enabling better identification of nodes and subsequent linking of body parts.

Maltese holds the second highest frequency within the dataset, accounting for 16% of the dataset. Tracking this particular skill presents a considerable challenge due to its heavy reliance on perspective. The Maltese is commonly observed and captured from various angles, including frontal, side, rear, and bottom-up views. To delve deeper into the aspects of this skill, a subset of videos has been extracted for some visual analysis and discussion.

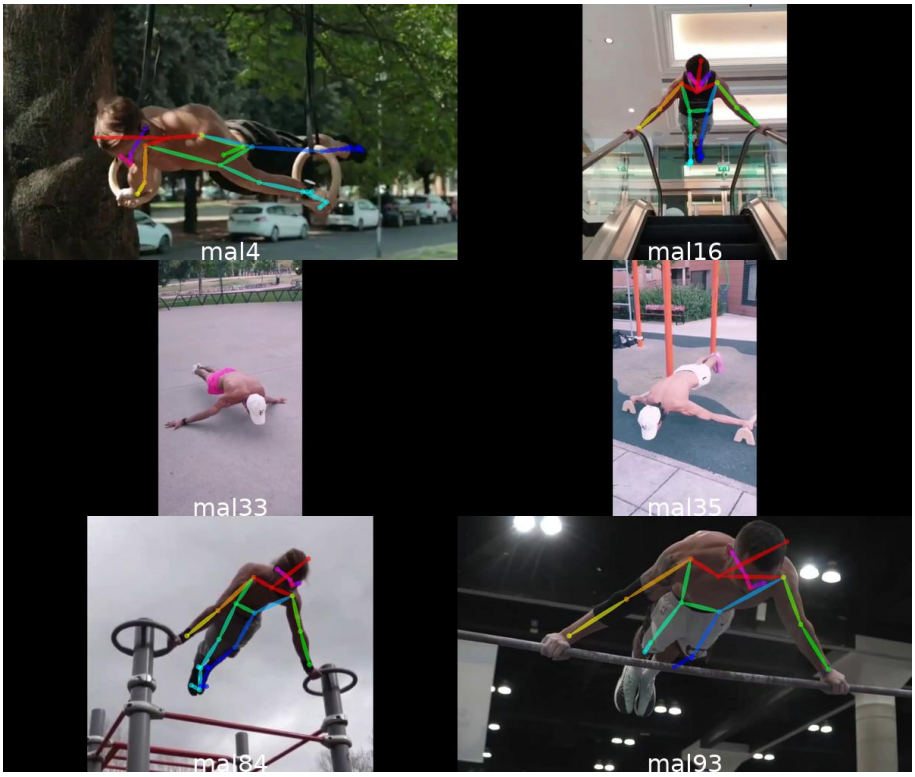

Figure 5.7: Maltese istances.

Based on the Fig. 5.7, several observations can be made. Firstly, we can identify positive examples such as *mal4*, *mal16*, and *mal84*, which demonstrate frames that are perfectly tracked. However, in the case of *mal93*, although the tracking is overall excellent, an interruption between the thighs and feet is noticeable due to the presence of the bar. Additionally, frames like *mal33* and *mal35* exhibit zero recognized joints. A noticeable common factor is specifically the absence of head information. This observation leads us to consider another influential factor: in the lasts two considered frames, the athletes use a hat. It appears that OpenPose struggles to detect crucial joints, particularly the upper ones, when athletes wear hats or something else on his hair. This limitation renders OpenPose "blind" to the presence of

these essential joints, consequently resulting in poor tracking performance.

One arm front lever comprises 10% of the total dataset. It shares similarities with the front lever in terms of body position and the perspective used for capturing it. To conduct a visual analysis of this specific skill, a subset of videos has been created.

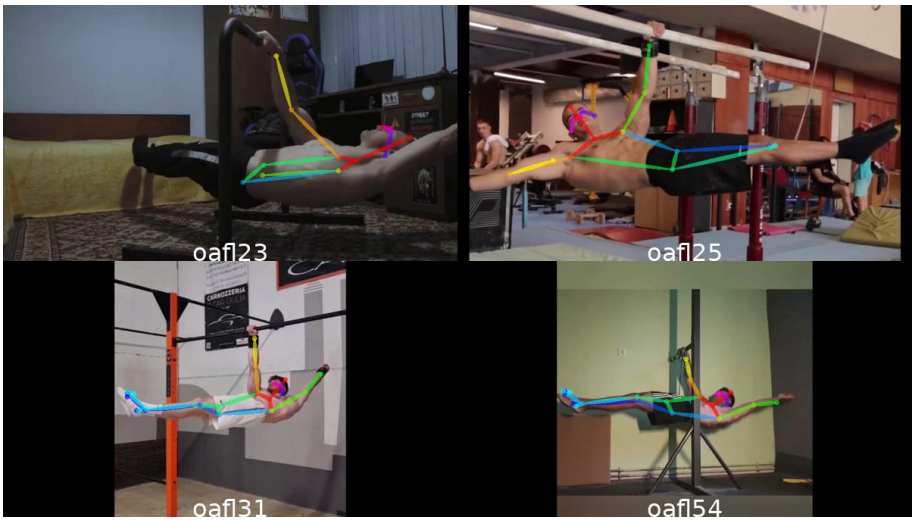

Figure 5.8: One arm front lever istances.

Fig. 5.8 highlights the great tracking capability of OpenPose with this skill. Notable positive examples include *oafl25*, *oafl31* and *oafl54*, where the skill is tracked exceptionally well. However, in the case of *oafl23*, a situation arises where the upper body is accurately tracked, but the lower body is not recognized. This discrepancy can be attributed to two potential causes: the presence of a vertical bar obstructing the view and the lighting conditions within the room. This last point will be discussed later on.

One arm handstand represents approximately 14% of the dataset. This skill has two primary perspectives, similar to those of the human flag or iron cross. It involves a vertical orientation but in an upside-down position. Despite the limited number of views included in the dataset, it is highlighted as one of the most challenging skills for OpenPose to track accurately. Now, we delve into a few graphical instances to further illustrate this difficulty.

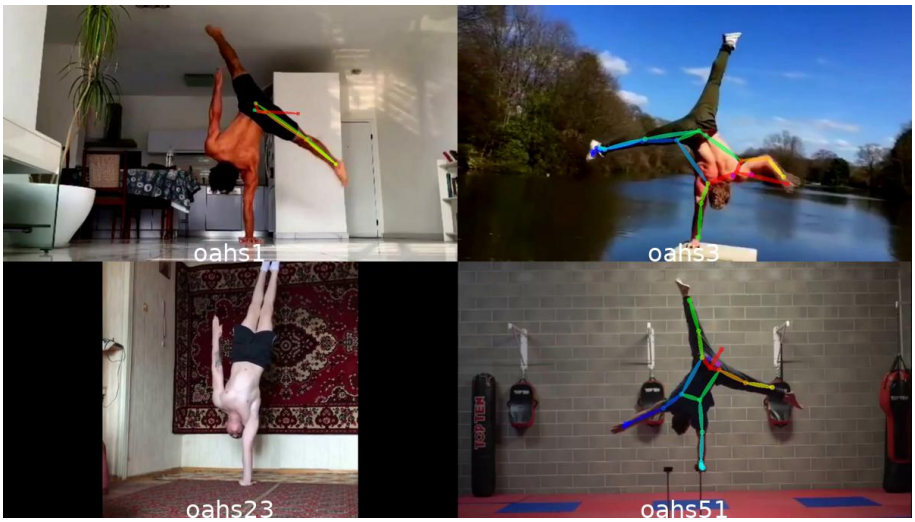

Figure 5.9: One arm handstand istances.

From Fig. 5.9, it is evident that the instances considered do not demonstrate optimal tracking. The only positive example that has been reported is *oahs51*, while *oahs3* exhibits partial tracking. Differently, the remaining two examples, *oahs1* and *oahs23*, clearly present significant challenges for OpenPose.

They can be attributed to two primary reasons:

- OpenPose has been trained with a limited number of upside-down poses, which affects its ability to accurately track such instances.
- Many of the videos within the dataset lack essential head information, resulting in a missing connection between the head and other body parts.

Planche represents approximately 15% of the total videos in the dataset. It shares common perspectives with the front lever. OpenPose demonstrates impressive capabilities in tracking this particular pose. Now, we dig into the analysis of some frame instances to further understand its tracking performance.

Figure 5.10: Planche istances.

Fig. 5.10 showcases a generally high level of tracking accuracy. The only frames exhibiting slightly lower precision are the *pl63* instances, primarily attributed to missing information of the lower body, likely caused by underexposure in that particular region. Overall, it is worth noting that the precision of tracking in these frames may be attributed to the visible upper body information.

None movements In this last paragraph, we analyse some 'none' movements, exctracted from some videos of the dataset. These type of frames are the most present in the dataset, because every skill has a pre and post movement. A small subset of them is displayed in Fig. 5.11 to make some considerations.

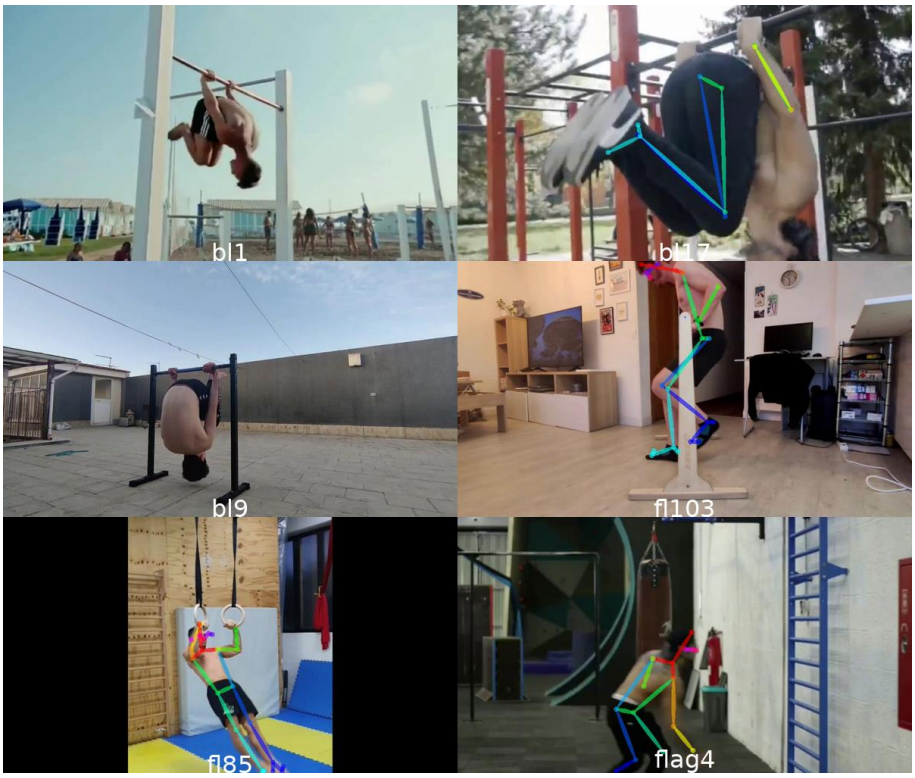

Figure 5.11: None movement instances in body pose estimation.

The frames *bl1*, *bl9*, and *bl17* present a challenging pose for OpenPose to track known as "skin the cat". This pose is commonly used as a transitional movement between a front lever and a back lever skill, and it is also utilized to enter or exit the back lever position. In skin the cat, the body is tucked, with the legs closer to the upper body parts.

This configuration poses a significant challenge for OpenPose to track due to the previously mentioned reasons. This skill combines an upside-down and horizontal position, creating a particularly demanding scenario for OpenPose's tracking capabilities. On the other hand, the frames *flag4*, *fl85*, and *fl103* showcase OpenPose's excellent tracking precision. This can be attributed primarily to the vertical position of the body and the availability of visible upper body information, facilitating accurate tracking by OpenPose.

5.1.2.2 Background and light influence

After having conducted some considerations per class, we will briefly discuss some other aspects that can have a negative or a positive influence on the body pose estimation challenge for OpenPose.

Background influence The background is a fundamental factor that can help or obstruct the OpenPose's performance. Challenging conditions can be due to poor contrast between the athlete and the scene, or to the confusing environment that makes discrimination difficult.

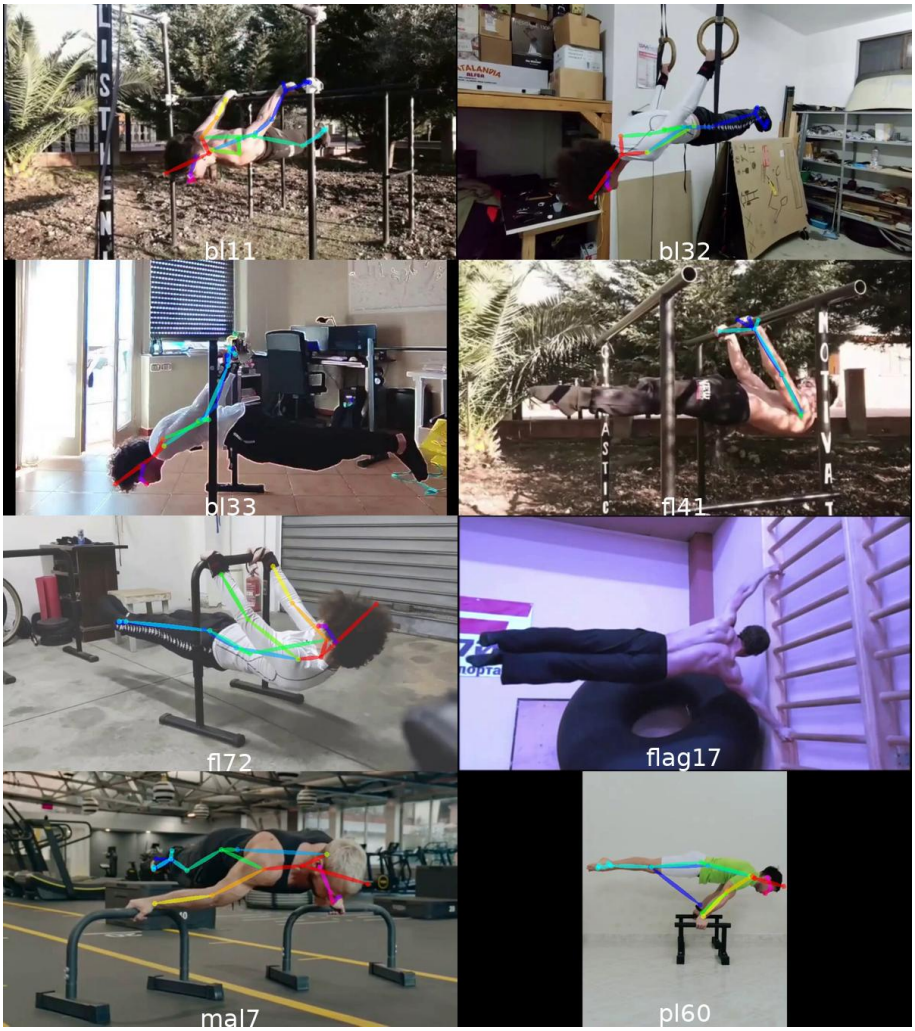

Figure 5.12: Background influence in body pose estimation.

From Fig. 5.12, a brief distinction between two main cases can be made.

- The first scenario can be observed in frames *bl11* and *fl41*, where an outdoor environment with a natural scene background introduces high variability in shadows. This variability in lighting conditions and background detail can pose difficulties for OpenPose.

- The second critical situation is exemplified by the remaining frames. In these frames, poor color variations between the clothes worn by the individuals and the background objects are evident. For instance, in *bl32*, the shirt blends with the wall, while in *bl33*, the trousers do not contrast well with the chair in the background. Similarly, in *fl72* and *mal7*, the color of the socks matches that of the objects behind them. In *flag17*, a peculiar automatic camera color correction issue arises, causing the trousers to blend with the wheel behind, obstructing the detection of upper body. Finally, in *pl60*, the tracking is only partially accurate, correctly identifying the bottom body connections.

Light level observation In the following paragraphs, we will delve into the final considerations regarding the lighting conditions in the environment. The first case we analyse, discusses cases of overexposure.

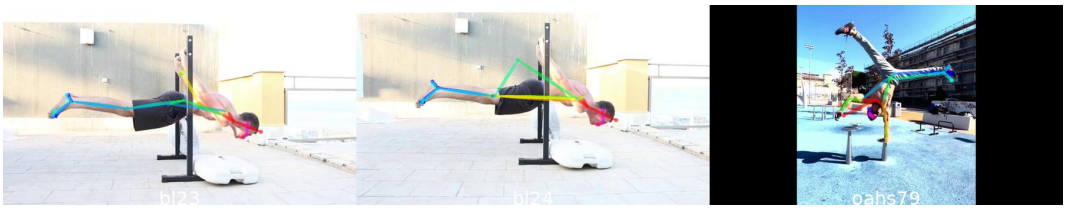

Figure 5.13: Overexposure influence in body pose estimation.

From Fig. 5.13, it is evident that in the first scenario, the lighting conditions do not appear to pose any issues for body pose estimation. However, in the second scenario, the connections between the hips and shoulders, as well as the one between the hands and shoulders, are compromised. In the last image, there are instances of connection mismatches, which can be attributed to the lighting. These lighting-related issues result in connection problems while still enabling the recognition of certain body joints. Nevertheless, it appears that this slightly diminishes the tracking process compared to the impact caused by underexposure.

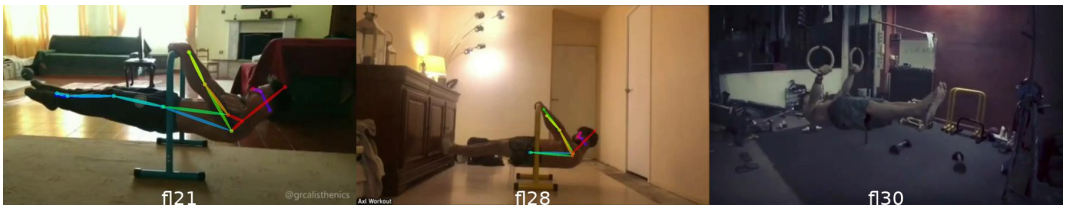

Figure 5.14: Underexposure condition instances.

Fig. 5.14 shows that the tracking precision depends on how much underexposure affects different body parts. In the first scenario, both upper and lower body information appear to be adequately preserved despite the lighting conditions. However, in the second image, only half of the body is recognized, indicating a partial loss of tracking due to underexposure. The last case represents the most challenging situation, where the light conditions are extremely unfavorable. It is noteworthy that the impact of underexposure differs from that of overexposure. In the case of underexposure, fewer joints may be recognized; however, the ones that are detected exhibit well-connected tracking.

5.1.3 General remarks

OpenPose has proven to be a valuable tool for body pose estimation. However, it is important to note that the limitations of OpenPose can impact the overall performance of the system. These limitations include challenges in handling occlusions, inaccurate joint localization in certain poses, and sensitivity to variations in lighting conditions.

Despite these limitations, OpenPose still provides reasonably accurate results for a significant percentage of cases. Based on our evaluation, we estimate that OpenPose performs sufficiently accurately for our intended purposes in approximately 70% of cases. However, it is worth mentioning that in approximately 30% of cases, the body pose estimation fails for various reasons, including factors such as the performed skill and the perspective of the shots. These challenging cases pose difficulties for accurate body pose estimation, leading to less reliable results.

Looking ahead, we expect that future advancements in body pose estimation models will address these limitations and further enhance the performance of such systems. These improvements may include better handling of occlusions, increased robustness to challenging poses, improved adaptability to various lighting conditions as well as a more stable tracking of neighboring frames.

5.2 Architecture and performance comparisons of the MLP classifier

In this section, we will present a comprehensive analysis of the model, highlighting all the evaluations that have been conducted. The results have been obtained using a *Cross Validation* technique with 5-fold splits, ensuring an

adequate estimations of the model's performance. To begin, we will introduce the optimal configuration that has been derived from the analysis. This configuration represents the optimal set of hyperparameters and model architecture that lead the best results. Then, we will delve into a detailed comparison of various parameters, exploring their impact on model's performance. In order to evaluate the model's performance, we have adopted the following metrics:

- **Training Loss**: This metric measures the final loss incurred during the training process, indicating how well the model is fitting the training data.
- **Accuracy Test**: This metric indicated the model's overall correctness in predicting the target labels on the test dataset.
- **Recall**: Measure the fraction of positive examples which have been classified as positive.
- **Precision**: Measures how many of the examples classified as positive are actually positive
- **F1 Score**: Combines precision and recall by taking their harmonic mean, providing a single value that reflects a trade-off between them. It gives equal importance to both metrics and is particularly useful when there is an imbalance between the number of positive and negative instances in the dataset.

By examining these metrics, we can gain a comprehensive understanding of the model's performance and make informed decisions regarding its effectiveness in solving the given problem.

5.2.1 Optimal configuration

The classification model that has been developed for this task is configured with the following specifications:

- Number of hidden unit blocks: *3*
- Size of hidden units: *512*
- Activation function: *LeakyReLU*
- Optimizer: *Adam* with a learning rate of *0.0001*
- Loss function: *Cross-entropy*

Throughout the training process, the model is trained for a total of 250 epochs. An epoch refers to a complete pass through the entire training dataset, where each example is presented to the model for learning. The choice of 250 epochs allows the model to gradually learn from the data over multiple iterations, refining its parameters and improving its predictive capabilities. The batch size used in each iteration is set to 512, indicating the number of samples processed together before updating the model's parameters. These settings and parameters have been carefully chosen to optimize the model's performance in the classification task, aiming to achieve accurate predictions and minimize the loss during training.

Table 5.1 provides a summary of the obtained cross-validation results.

Fold	Training Loss	Test Accuracy	Recall	Precision	F1 score
1	0.0127	72.57%	0.717	0.765	0.730
2	0.0135	**76.07%**	**0.775**	**0.783**	**0.777**
3	0.0120	74.41%	0.752	0.765	0.755
4	**0.0110**	73.32%	0.738	0.745	0.738
5	0.0121	73.24%	0.714	0.774	0.729
Average	0.0123	73.92%	0.739	0.766	0.746

Table 5.1: Optimal model cross-validation outcomes.

An alternative configuration for faster training utilizes a learning rate of 0.001 and 128 epochs, all the remaining settings are unchanged. This configuration exhibits a similar behavior, but the loss function converges more quickly due to the larger learning rate. Subsequently, this second configuration has been adopted for subsequent analyses.

5.2.2 Architecture comparisons

The initial analysis focuses on the structure of the network, with three specific analyses outlined and discussed.

5.2.2.1 Depth factor

In this section, we will examine the model based on the utilization of different hidden blocks. A hidden block, as a reminder, is comprised of a linear layer followed by an activation layer. Through this comparison, our objective is

to discover the optimal choices for constructing a good trade-off between performance and architectural complexity. The other parameters are equals to the optimal configuration ones. The results are then shown in Table 5.2.

Hidden blocks	Training Loss	Test Accuracy	Recall	Precision	F1 score
2	0.0477	73.21	0.734	0.760	0.742
3	0.0109	**73.24**	**0.736**	**0.763**	**0.743**
4	**0.0086**	72.42	0.727	0.756	0.732

Table 5.2: Hidden blocks size comparison.

After a few observations, it is apparent that the configuration with three hidden blocks emerges as the most favorable choice, exhibiting higher accuracy performance while maintaining a moderately shallow network structure. An essential point to highlight is the importance of maintaining a balance between network depth and performance. Indeed, deep networks can pose challenges like vanishing gradients and increased computational complexity.

5.2.2.2 Hidden units

Another structural analysis is conducted, focusing on the number of hidden units. These values represent the input/output size of the hidden layer, describing the network's complexity. Similar to the previous analysis, the objective is to identify an optimal trade-off between performance and complexity. The other parameter are set to their optimal values. Table 5.3 demonstrates that despite the slightly higher accuracy achieved with a hidden size of 1024, the 512 size is preferred due to its superior performance on other metrics and its a lighter network architecture.

Hidden units	Training Loss	Test Accuracy	Recall	Precision	F1 score
64	0.0477	71.80	0.720	0.754	0.728
128	0.0225	72.20	0.720	0.754	0.728
256	0.0124	71.26	0.708	0.759	0.719
512	0.0109	73.24	**0.736**	**0.763**	**0.743**
1024	**0.0105**	**73.31**	0.732	0.762	0.740

Table 5.3: Hidden units size comparison.

5.2.2.3 Activation functions

Finally, this analysis aims to identify the most suitable activation function for our network. Several functions were tested, and the results are presented in Table 5.4. Among the considered options, the *LeakyReLU* stands out as a clear winner, exhibiting higher performance across all columns of the table. The other parameters of the configuration such as the optimizer, the activation function, the hidden units and the hidden blocks are the same of the optimal configuration.

Activation function	Training Loss	Test Accuracy	Recall	Precision	F1 score
LeakyReLU	**0.0109**	**73.24%**	**0.736**	**0.763**	**0.743**
ReLU	0.0109	71.70%	0.722	0.760	0.731
Sigmoid	0.1122	72.67%	0.728	0.739	0.726
Tanh	0.0118	71.69%	0.720	0.748	0.726
SiLU	0.0158	70.51%	0.708	0.745	0.716

Table 5.4: Activation functions comparison.

5.2.2.4 Optimizers

In this subsection, we will provide a rationale for the selection of the optimizer used in this project. To ensure a comprehensive comparison, we have chosen some of the most commonly used optimizers. Table 5.5 shows the results obtained with different optimizers. The other parameters remain the optimal ones.

Optimizer	Training Loss	Test Accuracy	Recall	Precision	F1 score
Adam	**0.0109**	73.24%	0.736	**0.763**	**0.743**
SGD	0.1278	73.86%	0.741	0.746	0.739
RMSProp	0.0173	71.05%	0.709	0.743	0.714
Adagrad	0.1121	**74.14%**	**0.743**	0.751	0.742

Table 5.5: Optimizer comparison.

As it can be noted, the *Adagrad* optimizer allows to achieve a higher accuracy than the others. Despite this outcome, the *Adam* optimizer has been chosen for the following reasons: compared with *Adagrad*, *Adam* minimizes

the loss function better, converges to a lower value with a difference of 10^{-1}. This factor has a greater impact on our analysis, which can be generalized as follows. A model with an optimizer that guarantees a lower loss value is considered better than another one with a higher loss value because the main goal of optimization is to minimize the loss function. By doing this, the optimizer improves the model's ability to make accurate predictions, which suggests that the optimizer has effectively adjusted the model's parameters to reduce errors and improve its ability to fit the training data. Therefore, *Adam* is the chosen optimizer, which also demonstrates higher values in Precision and F1 score.

Fig. 5.15 illustrates the trend of the loss functions when using different optimizers:

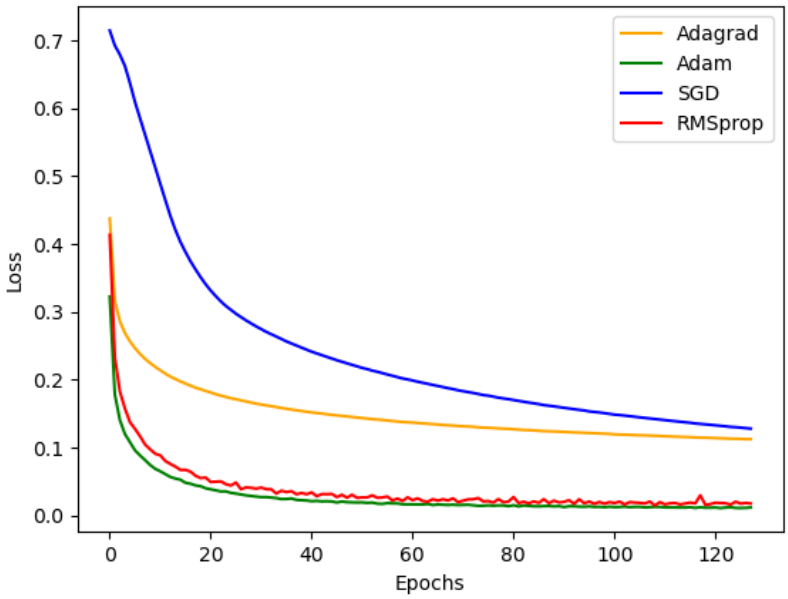

Figure 5.15: Optimizers loss functions comparison.

5.2.2.5 Learning rate

Following the selection of the optimizer, a comprehensive comparison of learning rates was conducted. Specifically for this analysis, the model was trained for 250 epochs to observe the behavior of the loss functions over an extended period. The results are shown in Table 5.6. The other parameters have the same values as the optimal ones.

Adam, lr	Training Loss	Test Accuracy	Recall	Precision	F1 score
0.01	0.0332	72.25	0.718	0.750	0.726
0.001	**0.0082**	72.70	0.733	**0.767**	0.741
0.0001	0.0123	**73.92**	**0.739**	0.766	**0.746**

Table 5.6: Learning rate comparison

Based on the conducted analysis, an interesting observation emerges: the model trained with a lower learning rate value demonstrates higher accuracy on the test set. To provide further insights, we present the curves representing the convergence of the loss functions. These visualizations allow us to access to the convergence time of each model and reach a better understanding of training dynamics.

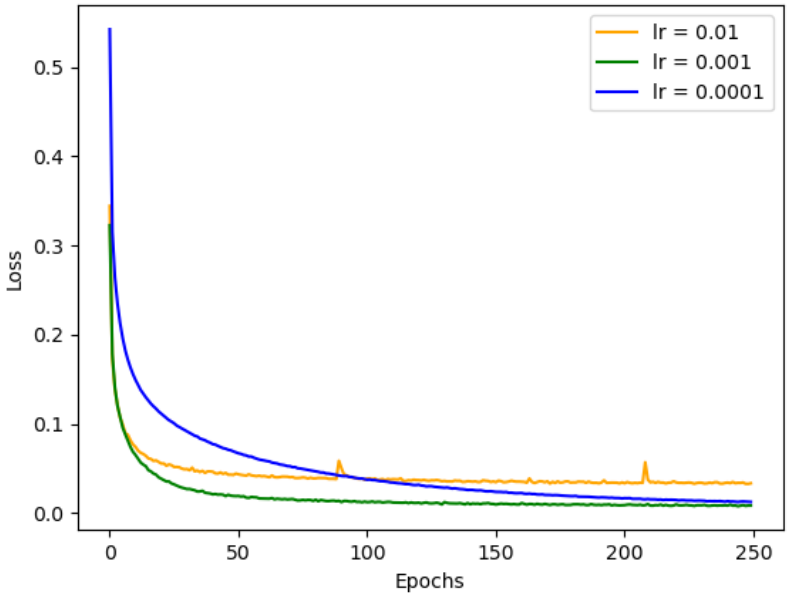

Figure 5.16: Loss functions for different learning rates.

From Fig. 5.16, we can clearly the different convergence behaviors and their impact on the achieved minimum loss values and accuracy.

- The *0.01* value makes the loss function converge to a suboptimal minimum around the 50th epoch. Although it converges relatively quickly,

it cannot minimize the loss function to a lower value compared to the other learning rate values.

- The *0.001* value takes slightly longer to converge, typically in the range of epochs 75 to 100. However, it achieves a better minimum value for the loss function compared to the previous value.
- The *0.0001* value exhibits a significantly longer convergence time, requiring almost all the training epochs to find a relatively optimal value. Despite the longer convergence time, it achieves the highest accuracy value among the three values. This is the value we adopt in this work.

5.3 Testing results

In this section, we present an analysis that includes both qualitative and quantitative aspects. The focus will be on interpreting and visualizing insights from the confusion matrix obtained during the test phase, as well as analyzing the accuracy values.

5.3.1 Confusion Matrix interpretation

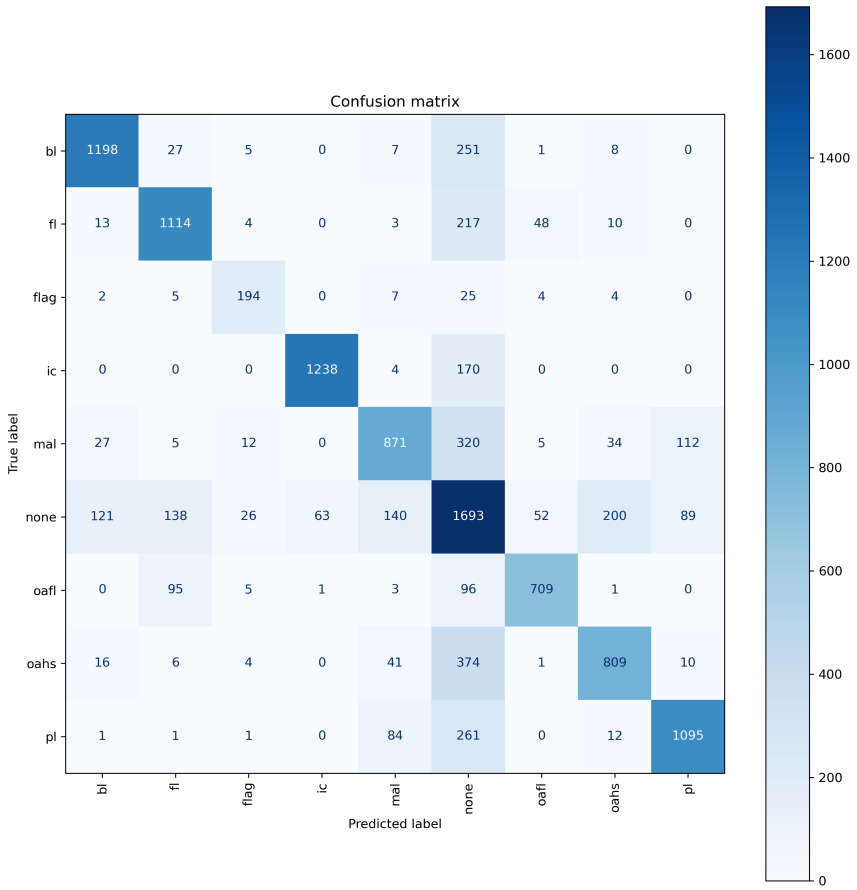

Figure 5.17: Confusion matrix.

5.3.1.1 Analysis of skill predictions

Fig. 5.17 reports the confusion matrix of the MLP component, which reveals the model's excellent ability to distinguish between different skills. To delve deeper into this analysis, we focus on the predictions for all skills except 'none'.

The average error per class is shown in Table 5.7.

The back lever exhibits an average error of *6.85%*, indicating a relatively accurate classification. Similarly, the front lever shows a comparable behavior with an average error of *11.14%*. The human flag also demonstrates good classification performance, with an average error of *3.14%*. In contrast, the iron cross stands out as the most accurately predicted skill, with a small

	Skills							
	bl	fl	flag	ic	mal	oafl	oahs	pl
Average skill-skill error	6.85	11.14	3.14	**0.57**	27.85	15.0	11.14	14.14

Table 5.7: Skill-specific average error % excluding the 'none' class.

number of frames being incorrectly classified. On the other hand, the maltese skill exhibits the highest average error among the classes, *27.85%*. This is likely influenced by over a hundred examples that were misclassified as planche. Conversely, the planche skill also shows a relatively high average error of *14.14%*, primarily due to approximately *84%* instances being wrongly predicted as maltese. As an example, Fig. 5.18 reveals two instances of maltese predicted as planche and vice versa.

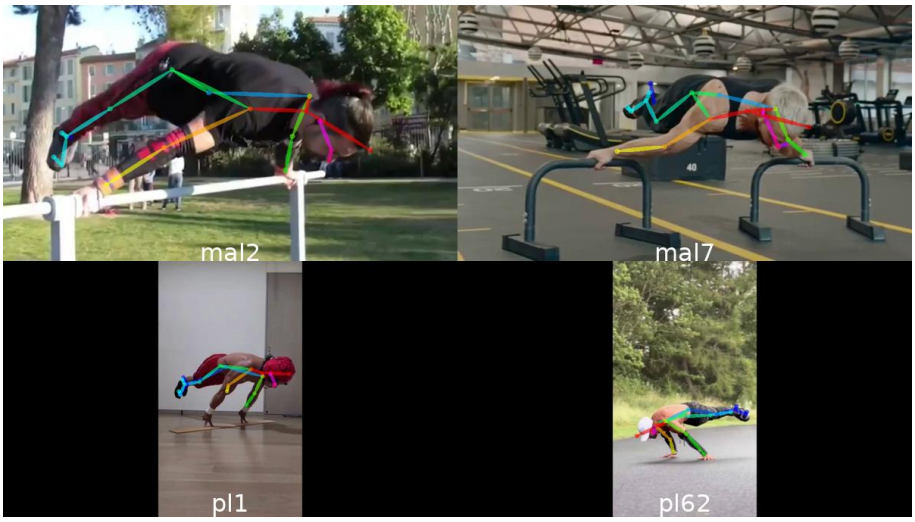

Figure 5.18: Maltese/planche wrong predictions.

The primary reason behind this issue is the similarity between the two skills. In fact, distinguishing between a maltese and a planche can be challenging, even for judges in competitions or the athletes themselves. This difficulty arises because the "wide" planche variant and the maltese performed with enclosed hands, are essentially the same skill. Finally, both the one arm front lever and the one arm handstand exhibit relatively good average error values of approximately *15.0%* and *11.14%* respectively.

5.3.1.2 Skills transition review

Despite the good values observed in the confusion matrix, the model's overall accuracy is not exceptionally high. The primary cause is the incorrect prediction between skills and 'none' body poses. Fig. 5.19 shows some skills recognized as absence of skill.

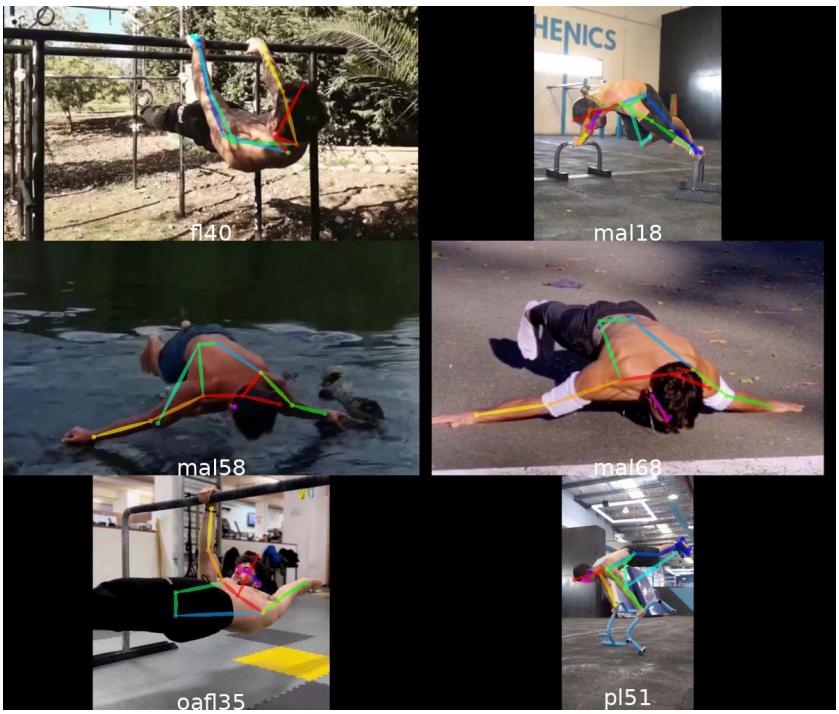

Figure 5.19: Mistakes in classifying none movements instances.

There are two main reasons behind these incorrect label predictions.

- One of the primary reasons is the poor body pose estimation performed by OpenPose, which poses a significant challenge for the accurate classification of skills.
- Another one can be attributed to the inherent uncertainty in distinguishing between skills and none movements. However, it is important to note that even for human judges, identifying the precise moment of transition between a none movement and a skill is a challenging task. During competitions, judges often have difficulties in determining the exact point at which the transition happens. This issue highlights the subjective nature of such decisions. Furthermore, it is worth considering that athletes may perform skills with variations in their body

alignment. While a perfect execution of an element typically involves maintaining a straight line throughout the body, some athletes may customize their movements by reducing the angle formed between the upper and lower body. These variations in body positioning can further complicate the classification process. Despite these challenges, the frame-level analysis we are conducting allows for a certain margin of error. Given the complexity and subjectivity involved in identifying transitions between skills and none movements, it is expected that some degree of uncertainty and variability will exist in the model's predictions.

An additional factor contributing to the challenges faced by the algorithm is the absence of a definitive way for the model to determine whether a skill is truly being performed or not, solely based on the spatial distribution of joints. This issue becomes particularly evident in cases involving pre or post-skill poses. In these situations, athletes may maintain their upper body in a valid skill position while their feet are still in contact with the ground. The current approach utilizing OpenPose does not effectively handle these situations, which can be problematic, particularly when the initial pose is very similar to the desired skill. To validate the significance of the second point discussed, a comprehensive analysis has been conducted.

5.3.2 Edge Segments Accuracy Analyzer

This analysis was conducted on the model's test results using cross-validation to examine the causation or correlation of transitions in the videos. A dedicated script was developed for this purpose, which identifies all edge values, serving as the boundaries between different skill segments, and calculates the accuracy within a specified radius around these edges. In order to validate the hypothesis mentioned earlier, it is important to note an increasing accuracy trend as the distance from the edges increases.

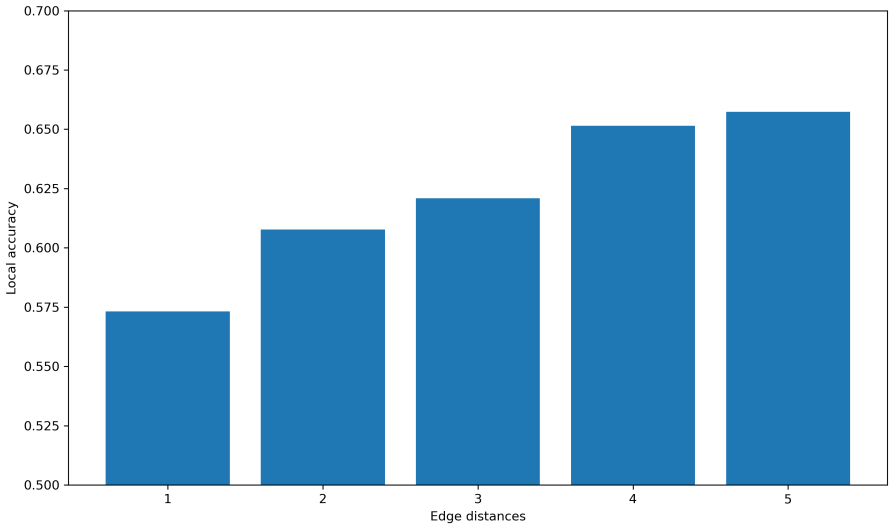

Figure 5.20: Analysis of the correlation between frame distance from the edge and local accuracy.

Fig. 5.20 provides further evidence in support of the hypothesis, as it demonstrates that the model achieves higher accuracy in the presence of more centered values.

5.4 Temporal segmentation algorithms

In this section, we present a concise analysis of the proposed algorithm for segment reconstruction. Additionally, we will compare it with the method presented in the PLS (Personal-Location-Segmentation) work [3], which is based on Hidden Markov Models (HMM). It is important to note that all the results have been obtained using the cross-validation technique.

5.4.1 Evalution metrics

Before discussing the experiments, we will briefly introduce the metrics involved in this section to evaluate the algorithms. The SF1 metric is a threshold-dependent, segment-based F1 measure. It is computed using precision and recall values for a specific threshold.

The formula for SF1 is given by:

$$SF1^{(\gamma)}(t) = 2 \cdot \frac{\text{precision}^{(\gamma)}(t) \cdot \text{recall}^{(\gamma)}(t)}{\text{precision}^{(\gamma)}(t) + \text{recall}^{(\gamma)}(t)}$$

The SF1 measure allows us to plot threshold-SF1 curves, which show the performance of the segmentation method at different tolerance levels. ASF1 (Average SF1) is the overall performance score of the segmentation method. It is computed as the average SF1 score across a set of thresholds $T = \{t \text{ such that } 0 \leq t \leq 1\}$:

$$ASF1^{(\gamma)} = \frac{\sum_{t \in T} \mathrm{SF1}^{(\gamma)}(t)}{|T|}$$

mASF1 (mean ASF1) is another performance measure that represents the average ASF1 scores for all considered classes ($\gamma \in 0, ..., M$). It provides an overall assessment of the method's performance across different classes.

5.4.2 VSR window size analysis

In the previous chapter, we introduced the algorithm and explained its logic. One crucial parameter in this algorithm is the window size, which needs to be manually configured. To determine the optimal window size that ensures superior performance in timeline reconstruction, an analysis has been conducted. The algorithm was executed using 20 distinct values for the window size, ranging from 6 to 26. The results were evaluated using the mASF1 metric, which serves as a reliable performance indicator.

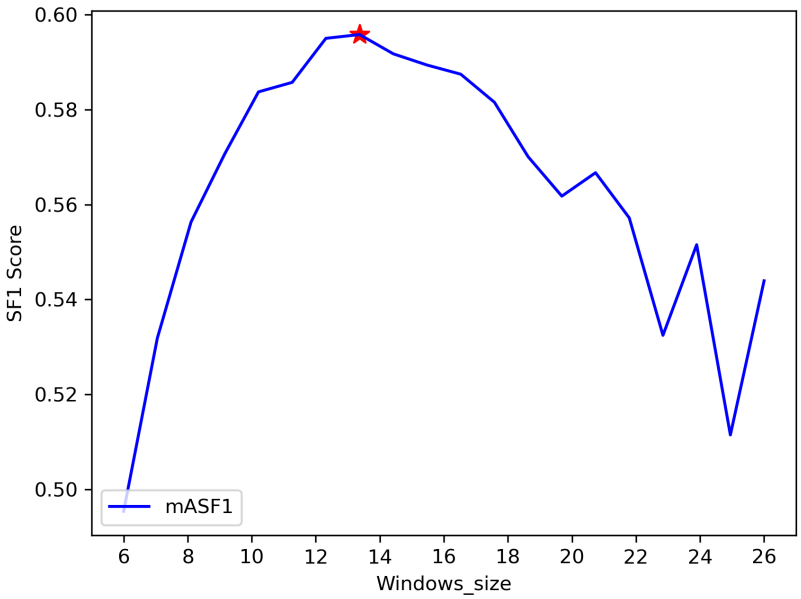

Figure 5.21: Comparison of mASF1 scores for different window sizes.

Fig. 5.21 exhibits distinct monotonic patterns with some pseudo-stochastic behaviors characterized by elevated values. Analyzing the obtained results, it becomes evident that the values 12 and 13 lead to the most favorable outcomes, delivering optimal approximations during the segment discrimination phase. However, manual tuning of this parameter during the inference phase can lead to improved results. The choice of window size can be approximated based on the desired precision. Lower values may lead to higher precision but increase the risk of introducing noise in the timeline. On the other hand, larger window sizes may sacrifice some accuracy but contribute to better generalization of video segments.

5.4.3 Algorithms comparison

In this subsection, we will conduct a comparison between the algorithm developed for this project (VSR) and a probabilistic method for temporal video segmentation. The purpose of this comparison is to assess the performance and effectiveness of our algorithm in relation to existing approaches.

To evaluate the performance, we will use the mASF1 metric, which provides an overall measure of accuracy for segment reconstruction. Additionally, we will examine the ASF1 scores for each individual class to gain insights into the algorithm's performance across different skill categories.

Table 5.8 reports the raw predicted labels. This will allow us to analyze and visualize the accuracy of the predictions made by our algorithm.

Method	mASF1	Bl	Fl	Flag	Ic	Mal	None	Oafl	Oahs	Pl
VSR	0.6	0.66	0.61	0.60	0.77	0.51	0.36	0.64	0.55	0.62
PLS	0.67	**0.70**	**0.72**	**0.64**	**0.85**	**0.61**	**0.42**	**0.77**	**0.61**	**0.71**
RAW	0.1	0.06	0.09	0.05	0.31	0.06	0.08	0.09	0.02	0.11

Table 5.8: Per-class ASF1 scores and related mASF1 measures for all compared methods.

A graphical comparison is shown in Fig. 5.22. Fig. 5.23 illustrates the SF1 scores of the algorithms at different thresholds.

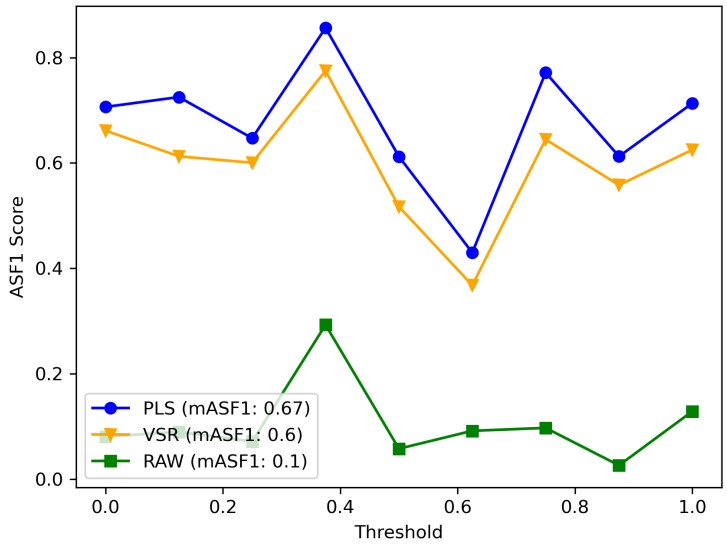

Figure 5.22: ASF1 curves comparing the proposed method with respect to the method presented in [3].

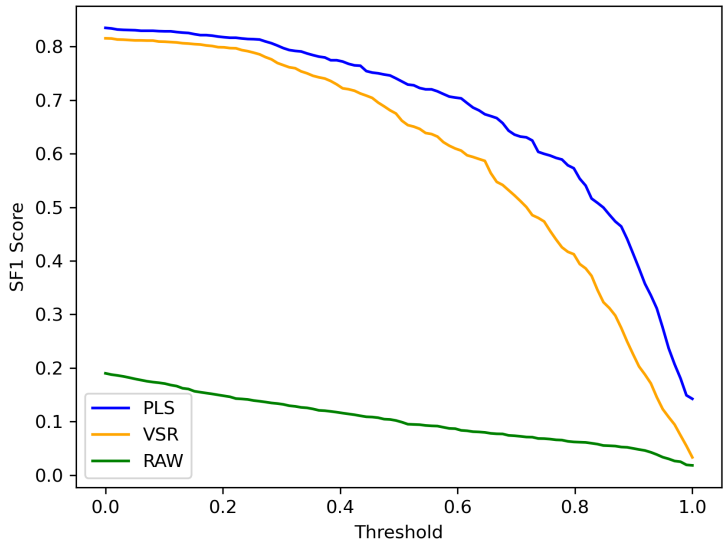

Figure 5.23: Threshold-SF1 curves comparing the proposed method with respect to the method proposed in [3].

Our algorithm performs comparably across all thresholds, displaying a steeper decline in higher thresholds compared to the PLS [3] despite its simplicity.

5.5 Inference phase

In this last section, we will discuss about the inference part, describing the steps used to test the whole architecture. We will first introduce the algorithm built to perform the inference, then we will make some final considerations on the performance of the model.

5.5.1 Algorithm implementation

The algorithm presented here is responsible for integrating various components to create a final result for the entire video processing pipeline. The main objective of this algorithm is to generate a comprehensive numerical and graphical timeline that shows the composition of the analyzed video. The algorithm is composed of several steps to achieve this:

1. Initially, the algorithm reads the video passed as an argument. The video can have any file extension, resolution, or frame rate. By utilizing the *FFmpeg* library, the video is automatically converted to a standardized resolution of about *960x540* pixels and a frame rate of *24* frames per second. It is important to note that this conversion process preserves the original aspect ratio of the video without distorting it.

2. Following the video conversion stage, the algorithm proceeds by processing the video using OpenPose. This step involves processing the video and generating corresponding JSON files of the body pose joint coordinates and confidence scores, which are stored within a designated folder. The OpenPose processing logic employed here is the same as the one discussed in the previous chapter.

3. Once the JSON files have been generated, the algorithm moves forward to construct the dataset, employing the same methodology utilized during the training phase to build the training/test datasets.

4. Then, the algorithm loads the trained model and associated classes. This allows for the evaluation of the current dataset using the optimal configuration of the MLP discussed in the previous sections.

5. After the evaluation, the raw labels predicted by the model are submitted to further processing using the algorithms seen in the previous chapter, namely the proposed one and the PLS.

6. In this step, the algorithm incorporates ground truth labels by manually inserting them. This allows for a direct comparison between the algorithm's predictions and the actual expected outcomes.

7. Finally, the algorithm presents the output of the previous steps by visualizing them through a graphical pipeline specially built for this purpose. The graphical pipeline provides a visual representation of the processed video segments, enhancing the overall understanding and analysis of the video timeline.

5.5.2 Inference instances

After discussing the algorithm's steps, we analyse some inference instances to gain insights into the results and understand the behavior of the proposed algorithm. In addition to the two algorithms, we compared the ground truth timeline with the raw ones to assess the impact of these algorithms.

It is worth mentioning that a window size of 10 was used for these analyses, which appeared to be the most suitable value based on the tested video. It is important to note that this differs from the best value obtained during the testing phase. The variation is attributed to the structure of the considered video. In the testing phase, a batch of videos was analyzed collectively, meanwhile in the following cases, the videos are processed individually.

Figure 5.24: First inference example.

The initial video under analysis in Fig. 5.24 shows a flag pose and demonstrates the model's exceptional performance due to precise OpenPose tracking. In this instance, the PLS algorithm exhibits superior approximation of

the segment's end, accurately capturing the conclusion of the pose. Meanwhile, the proposed algorithm displays a slightly shifted end point for the segment.

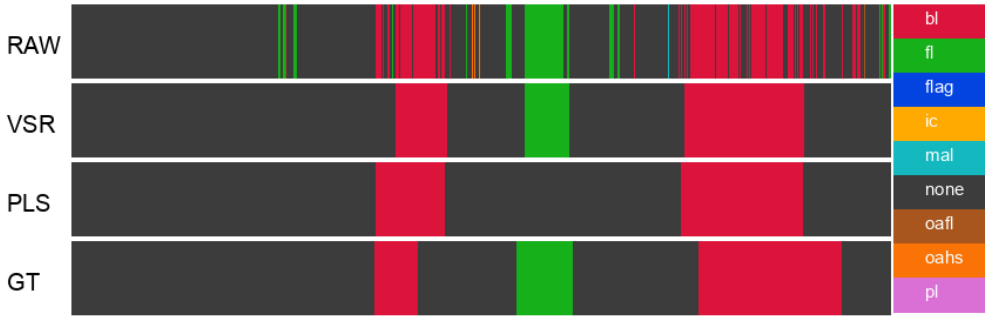

Figure 5.25: Second inference example.

In this second instance illustrated in Fig. 5.25, we encounter a different scenario. The model's performance is noticeably poorer compared to the previous example, primarily because OpenPose makes mistakes in correctly recognizing the middle front lever. As a result, the accuracy of the model's predictions is compromised. When examining the performance of the PLS algorithm, we observe that it exhibits better performance in the initial segment. However, it misses in reconstructing the middle segment, which is tracked by the VSR algorithm.

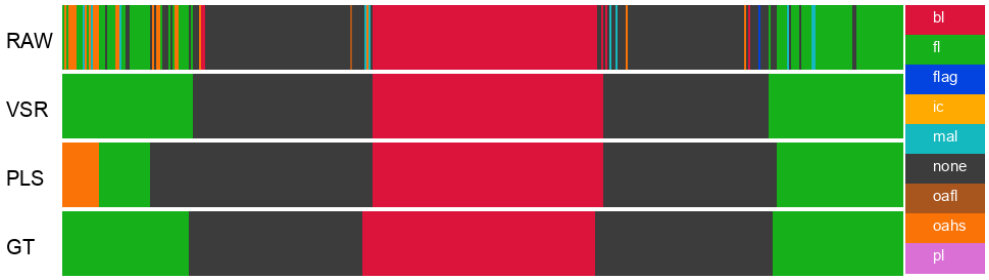

Figure 5.26: Third inference example.

This example, shown in Fig. 5.26 highlights a scenario where the model exhibits high uncertainty in the starting segment. It is in this test that the VSR algorithm demonstrates its superior performance, showing one of its key strengths. The proposed algorithm effectively manages noise and accurately recreates the ground truth segment with remarkable precision.

This outcome is the result of the carefully designed logical steps incorporated into our algorithm. In the remaining two segments, the performance of both algorithms is nearly the same, resulting in an optimal timeline reconstruction.

Figure 5.27: Fourth inference example.

In this fourth experiment (see Fig. 5.27), we encounter another scenario where the PLS reconstruction method fails to recognize an entire segment, highlighting a clear advantage of the VSR algorithm. Furthermore, our algorithm demonstrates superior performance in recognizing the second segments. However, it struggles to handle the noise near the third segment and mistakenly identifies a segment that is not present in the original footage. It is important to note that this issue arises because the model faces additional challenges in dealing with uncertainties in the middle of the timeline. These uncertainties are caused from the athlete changing the perspective of the skill, vertically mirroring their body by 90 degrees. This change in perspective poses a significant challenge for OpenPose, challenge that is well handled by both algorithms.

Figure 5.28: Fifth inference example.

In the analysis shown in Fig. 5.28, we examine the testing of a maltese skill. Initially, the raw timeline exhibits an incorrect segment, which can

be attributed to the pre-isometric pose of the arms recalling the position of the arms in planche. However, both algorithms correctly discard this erroneous segment. This particular case illustrates the identical behavior of both algorithms, where there is a discrepancy in the starting frame but accurate reconstruction of the remaining segment.

Figure 5.29: Sixth inference example.

In this final test showcased in Fig. 5.29, a significant flaw in the proposed algorithm becomes evident: the occurrence of small ending segments. This problem is primarily due to the utilization of a large window size value. To address this issue, the same test was conducted using a window size value of 5, aiming to mitigate the impact of the *window_size* parameter.

Figure 5.30: Sixth inference example with a different window size.

As observed in Fig. 5.30, the obtained outcomes exhibit a significantly higher level of accuracy. While both algorithms exhibit similar performance at the start of the segment, the PLS algorithm outperforms the proposed algorithm in approximating the end of the skill.

Through these analyses, we have discussed the strengths and weaknesses of both algorithms, as well as the labels generated by the model.

Chapter 6

Conclusion

The project comprised several distinct parts, each of which was addressed with attention and detail. Numerous design choices were adopted and justified in various sections of the project. The project commenced by constructing the dataset, which required substantial manual effort. We collected calisthenics videos from the internet and recorded some locally. To identify the starting and ending frames of each skill within the videos, we utilized Kdenlive software. At the time of writing, the dataset contains a total of 573 videos. We anticipate that this number will experience substantial growth in the future. Then, OpenPose was utilized as the pose estimator for the athlete tracking task. Through OpenPose, we extracted the keypoints from each frame within the dataset, enabling us to create a normalized numerical dataset derived from OpenPose processing. This dataset was then divided into a training set and a test set, with data augmentation techniques applied specifically to the training set. This process involved conducting extensive analysis to study the behavior of the pretrained model across various scenarios. The model was tested on each skill present in the dataset, resulting in improved tracking outcomes for poses such as iron cross, front lever, and human flag. Difficulties were encountered during the tracking of the one arm handstand. Furthermore, additional analysis was performed to examine the impact of lighting levels and background complexity on the tracking performance. The reasons behind these challenges were discussed and examined. One crucial aspect of the project undoubtedly revolves around skill recognition using the employed neural network. A specially designed multilayer perceptron with a specific structure was utilized. Several testing was conducted to configure the network, involving the variation of parameters and comparing the obtained results. This led to achieving a good level of accuracy, and a detailed analysis of the outcomes during the testing phase was carefully focused. The reasons behind certain incorrect predictions, partic-

ularly between skill-none and skill-skill classifications, were highlighted and discussed. Furthermore, an explanation for the accuracy level was provided through a segment's edge analysis, which reinforced the hypothesis about the high uncertainty in the transition between two poses. Another key phase of the project is the video segment reconstruction process. In this project, we introduced an approach for assembling video segments by utilizing an algorithm that calculates the mode within each batch. This method allows for the effective combination of different segments, enhancing the overall video reconstruction process. The algorithm's details were outlined in the proposed methodology section and compared against the method proposed in [3]. Finally, a method for making inferences on the model was presented, with specific focus on the timeline reconstruction phase. Several tests were conducted to demonstrate situations where the segment reconstruction algorithm exhibited optimal effectiveness, as well as cases where its weaknesses were identified. Despite achieving positive outcomes, there are potential areas for future improvements that can lead to better results:

- Expanding the number of recognized skills: Currently, the released version of the tool can identify eight skills, as mentioned in the first chapter. However, this number can be increased by adding new skills or incorporating different variations of existing ones.

- Upgrading the pose estimator: The pose estimator used in the project, OpenPose 1.7.0, was considered the best available solution at the time of writing. However, in the future, utilizing a newer version of OpenPose or exploring alternative pose estimation models could enhance the tracking of videos and the construction of the numerical dataset.

- Enhancing the classification task: The current classification task is accomplished using a multilayer perceptron that performs well when the pose estimator behaves correctly. However, to achieve better outcomes, other machine learning or deep learning architectures can be employed to improve the classification accuracy.

- Improving the segment reconstruction algorithm: The segment reconstruction algorithm, which aims to approximate the ground truth timeline, is based on three steps. However, its behavior, particularly at the edges of the video, can be further improved. Exploring alternative approaches that consider secondary factors in addition to label modes can enhance the accuracy of the reconstruction.

In conclusion, this project makes an initial contribution to Calisthenics through the introduction of a new dataset of skills. The integration of pose estimation,

classification, and video timeline reconstruction showcases the feasibility and potential of the task. The comprehensive analysis conducted through this work not only provides valuable insights but also sets a promising direction for future research in skill recognition within the Calisthenics field.

Acknowledgements

In this last chapter, I want to express my most heartfelt acknowledgments to all the people who have helped me in the making of this work and, more generally, during these three years.

First and foremost, I wish to express my deepest gratitude to my relator Antonino Furnari, for his availability and professionalism demonstrated throughout this work. His guidance and expertise have been instrumental in achieving the results, and I am truly pleased for the opportunity of having learned from him.

I am also eternally grateful to my family, who have been a constant source of love. To my parents, for creating a supportive environment that has made it possible for me to reach this important stage. Their efforts in simplifying things for me and their constant encouragement have been fundamental in my academic journey. To my special brother Alessandro, the better version of myself who has been a guiding light in every moment I needed and my cockatiel Cittolo, for providing me with moments of joy and companionship during long study hours. To my grandparents, the belief in my abilities has been the driving force behind my academic journey, their kind words have provided warmth and strength to me countless times. Their presence in my life has been a reminder of the importance of perseverance and determination. To my girlfriend Rossana, for her steady support and for always standing by my side. Her presence has brought joy and stability to my journey. Her understanding and upliftment has meant the world to me. To the rest of my family, including my uncles and aunts, for always being present.

I would also like to acknowledge the support of a few course colleagues, particularly Luca. His philantropy and kindness have made a significant impact on my academic and personal growth. I am also grateful to Riccardo and Chiara for their friendship during this time and to all friends of mine who are not part of the university circle.

To all those mentioned above and to everyone who has played a role, however small, in shaping my academic and personal development, including those I may have inadvertently omitted, I express my appreciation.

Bibliography

[1] R.B. Ash. *Basic Probability Theory*. Dover Books on Mathematics Series. Dover Publications, Incorporated, 2012. ISBN: 9780486135199. URL: https://books.google.it/books?id=fmNbFnfrb14C.

[2] Zhe Cao et al. *OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields*. 2019. arXiv: 1812.08008 [cs.CV].

[3] Antonino Furnari, Sebastiano Battiato, and Giovanni Maria Farinella. "Personal-Location-Based Temporal Segmentation of Egocentric Video for Lifelogging Applications". In: *Journal of Visual Communication and Image Representation* 52 (2018), pp. 1–12. ISSN: 1047-3203. DOI: https://doi.org/10.1016/j.jvcir.2018.01.019.

[4] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016. ISBN: 9780262035613. URL: https://books.google.co.in/books?id=Np9SDQAAQBAJ.