BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION COMPUTER SCIENCE

DIPLOMA THESIS

Correctness of Fitness Movements Using Machine Learning

Supervisor
Dr. BORZA Diana-Laura

Author
Jeler Andrei-Iulian

2021

UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ

LUCRARE DE LICENŢĂ

Corectitudinea Miscarilor de Fitness Folosind Machine Learning

Conducător științific
Dr. BORZA Diana-Laura

Absolvent
Jeler Andrei-Iulian

2021

Abstract

Many people around the world are doing physical exercises to stay fit and relax. The fitness domain is one of the biggest in the world, which has a large number of people doing it. With a high number of practitioners, the number of injuries can be pretty high too. For someone who just started to do sport, even small or medium injuries can be deterministic in the decision to stop in the future. Mistakes during a fitness exercise can result in injuries or reduce the effectiveness of the movements.

The Artificial Intelligence domain grew enormously in the last years, evolving from solving easy problems to solving huge problems, like classifying images with high accuracy and even predicting the stock value in the future. The Human Pose estimation is one of the problems that gathered considerable attention from the Artificial Intelligence community. It is one of the main solutions for understanding people in images and videos. The main job done is to find the relative coordinates of the joints of a human body from a given frame.

This thesis has as objective the presentation of how Artificial Intelligence can be used to help the sports domain, by creating a virtual fitness coach for people who want to train at home or at the gym. The application checks constantly for the status in which the current repetition is, ranging from start and end, and checks for mistakes in execution by checking the current and previous frames of the sequence of images.

This paper proposes two approaches that are trying to achieve that goal. The first one is based only on code, making the necessary computations by hand in code, after some handcrafted geometrical rules. The second one uses a Machine Learning approach, where the repetition status and the correctness of the current movement are predicted using neural network models.

The repetition status checker uses a simple and short CNN architecture to make predictions on the current frame. For the correctness checker, multiple frames are used to make predictions, so a temporal direction is followed. For the second checker, multiple architectures and sizes were tested and compared to see how they perform. They are using the following neural network types: CNN, LSTM and CNN + LSTM.

These models are used inside a system that has a server and a client application. The description of the application and the presentation of the models tested are found at the end of the thesis.

This work is the result of my activity. I have neither given nor received unauthorized assistance on this work.

Jeler Andrei-Iulian

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Motivation

The sport domain is one of the most popular domains in existence, statistics showing that around 43% of the European Union population that was surveyed by EuroStat practised sport at least once a week in 2014 [Eur], and 19.3% of the United States population was engaged in a sport activity each day in 2019 [Dav].

A good example of a sport activity that took the eye of a large amount of curious people was Biletul de Sănătate organized by Sports Festival in the center of Cluj-Napoca, between 23 September 2020 and 31 December 2020, event that gave a free bus ticket to the people that did 20 squats in less than 2 minutes in front of a device that counted the numbers of squats performed by someone. During this period, there were given 55.000 free bus tickets (about 1.100.000 squats or 1.900 hours of sport activity) [Mir]. The event had a big success, and a new edition is in planning.

Even though so many squats were performed during this period, not all of them were correctly executed, the camera only counting the number of repetitions. So, a person does not receive a feedback for how the exercise was performed. If the execution is not correct, the effect of the activity is not at a full potential. Also, if the mistakes are huge, injuries can appear.

Depending on the severity of the injury, a person can be in huge pains, some cases resulting in a life without high intensity activities. So, a way to be able to know if the exercise is done as it should be, would be very good for those who want to start doing exercises, or start recommencing doing them. A solution would be to get help from an fitness instructor, but they can be pretty expensive.

1.2 Injuries Statistics

During physical activities a person can sustain injuries, but how probable they are, and what is the duration of recovering from an injury. According to the National Safety Council, during 2019 in USA, the following injuries were treated in hospital emergency departments: [Nat]

Sport/Activity	Injuries in 2019
Exercise, exercise equipment	$468,315$
Bicycles	$417,485$
Basketball	$403,980$
Football	$292,306$
Swimming, pools	$190,743$

Table 1.1: Injuries in USA, 2019 [Nat]

Analyzing the data from Table 1.1, the fact that exercises(exercises with weights, aerobics, etc.) result in the most injuries that happened in 2019 in USA. In the following table, the nature of injuries between 2011 and 2014 in USA is presented: [SCH16]

Nature of injury	Injuries 2011-2014
Sprains	$4,262,000$
Fractures	$2,055,000$
Superficial/contusions	$1,953,000$
Dislocation	$296,000$

Table 1.2: Nature of injury in USA, 2011-2014 [SCH16]

1.3 Covid-19 and its Impact on Sport

In the early 2020, an unpredictable event occurred and changed the many aspects of the day to day life, the Covid-19 Pandemic, which is still an issue (at the time of writing this thesis). Lockdowns and curfews started to take shape in many countries, and some domains were deeply affected by the effect of the pandemic. One of the most affected one was the sports domain, many possibilities for a person to do a physical activity were closed. The gyms were closed worldwide, so most of the people who were actively going to the gym were forced to do exercises at home.

In USA, more than 38,000 gyms were closed in May 2020, and 25% of the total number of gyms could close permanently in the near future. Also around 60% Americans plan to cancel their membership and 25% of them do not plan on returning to the gym. During this period, the digital solutions grew up, 74% of Americans using at least one fitness app during the quarantine [Nic].

In UK, the number of injuries during exercises bloomed during lockdown, over 7.2 million Brits have been injured while trying to stay fit, 30% during online classes or apps, 28% during weight training and 22% using home gym equipments [Rob].

1.4 Objectives

This Thesis has as objective the presentation of the aspects regarding the Machine Learning field and the concept of Human Pose Estimation and use these concepts in the creation of a software that can be used as a fitness coach. One of the main aim of the paper is to develop a solution that can be utilized in helping other people in correcting possible mistakes during an exercise, diminishing the number of injuries for begginers.

The application is designed as a desktop application that can give feedback about the eventual mistakes during the execution of some exercises, and count the number of repetitions done. Also, the application should be able to use a webcam or a Kinect 360 sensor (which gives better positioning). The paper will present two methods for obtaining the desired application: one solution using computation made directly by hand in code (for example the angle computation between 3 adjacency joints), and one using Neural Network models, separated in predicting the currrent status of the repetition (such as start and end) which are trained on correct executions of the exercise and incorrect executions. This software is meant to reduce the number of injuries during these exercises and to help the user to work on the desired body part, increasing the effectiveness of these exercise.

1.5 Chapters Summary

This section will provide a short summary on what the following chapters are concentrated on.

In **chapter 2** the theoretical aspects of Machine Learning and Deep Learning are presented. It begins with presenting the term of Machine Learning and the types of learning used in different kinds of algorithms. After that the concept of Neural Networks is presented, including Artificial Neural Networks and Convolution Neural Networks. The layer structure of a neural network is described, together with the structure of a node, which is contained in a layer. In the end, the Recurrent Neural Network is presented, together with one specific architecture used by RNN's, the LSTM.

In **chapter 3** the concept of Human Pose Estimation is presented. In the beginning a brief introduction is presented, such that that the concept can be understood better. After that two classification on which the job is done are presented: the dimension (2D vs 3D) and the body type. The chapter continues with presenting the joints configuration and how the algorithm obtains the desired results, by showing the approaches used in different existing models, which is divided in the number of persons that are given to extract the desired keypoints. The chapter ends with the presentation of the Kinect sensor from Microsoft, that is a hardware solution of the concept, that uses depth sensors to obtain accurate predictions.

In **Chapter 4** the proposed approaches are described. The chapter begins with the description of the problem the thesis aims to solve and is followed by a brief introduction about the two proposed approaches. The first approach is based only on code, and the chapter introduces how the approach can be used, and describes each process. The second approach, the machine learning based, is divided in 2 neural networks, one for the repetition status, that helps on counting the number of repetitions, and one that is used to check the correctness of the last frames of the recording of the user's exercise. Both models are described, the architecture is presented and also the dataset is described.

Chapter 5 provides a detailed presentation of a software application that is using the approaches described in the previous chapter. The specifications description together with the functional requirements are in the beginning, followed by the design part, which contains details about the architecture design, class and database diagrams. In the end, details about the implementation steps are presented together with a short user manual, that includes details about how to use the application and also details about the User Interface.

Chapter 6 describes the results obtained by the neural network and provides a short discussion about both approaches and comparison with some similar solutions that are found on the internet. Also, this chapter introduces a short state-of-the-art results (comparisons) with the existent solutions, together with the solution obtained in this thesis.

Chapter 2

Machine Learning

2.1 Brief Definition of Machine Learning

The domain of machine learning is one of the popular area of computer science, and one of the most promising one. One of its many definitions sounds like this: ”Machine learning is the science (and art) of programming computers so they can learn from data.” [Gér19]. Analysing the definition, the most important part is the phrase ”learn from data”, which synthesize the meaning of machine learning. The human learns in life from different sources, like personal experience, but the computer does not have this ability. The program gets the data and make some kind of inferences, making heuristics, such that it can learn how to perform the job that is required to be done.

2.2 Types of learning

According to the amount and type of supervision given by the humans, the Machine Learning concept can be divided in 4 major categories: **supervised learning**, **unsupervised learning**, **semisupervised learning** and **Reinforcement learning**.

2.2.1 Supervised learning

In Supervised Learning (Figure 2.1), the training set of the algorithm includes a label, the desired solution [Gér19]. Each input data receives a label(output), so it is a good option for a classification algorithm. Another use is in predicting a target numeric value, for example: predict the cost of a house, knowing the previous prices of the house.

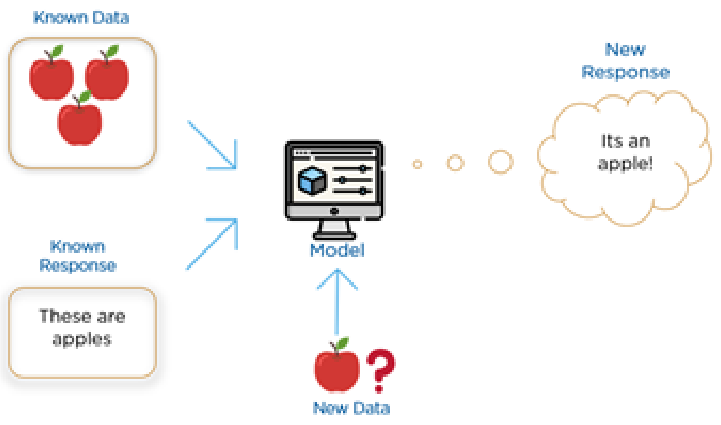


Figure 2.1: Supervised Learning [Hun]

2.2.2 Unsupervised learning

In Unsupervised Learning (Figure 2.2), the training set of the algorithm is unlabeled, and the algorithm tries to learn without a teacher (human help) [Gér19]. The algorithm tries to divide the input data in clusters, detecting similar characteristics, and establishing relations between them. It is also used to help in detecting anomalies.

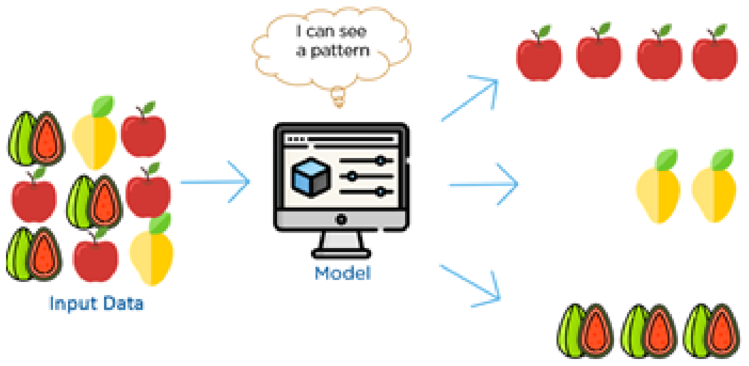


Figure 2.2: Unsupervised Learning [Hun]

2.2.3 Reinforcement learning

In Reinforcement Learning (Figure 2.3), the learning system, called an agent, can observe the environment, perform actions and get feedback (rewards for good actions, penalties for bad ones). Then, it must learn alone what the best strategy is, called a policy in this context. A policy defines what action the agent should choose when it is in a given situation. [Gér19]

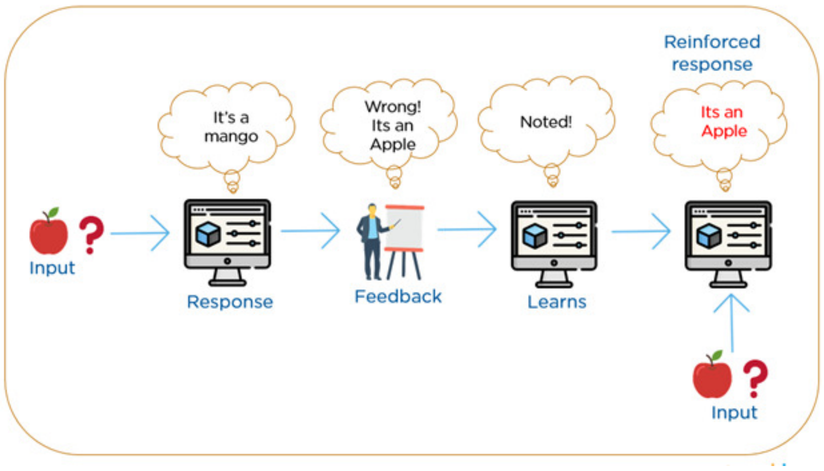


Figure 2.3: Reinforcement Learning [Sha]

2.3 Neural Networks

Neural Networks (NN) or Artificial Neural Networks (ANN) are a set of algorithms, modeled after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labeling or clustering raw input. The patterns that are recognized are numerical, inside vectors, which means that real-world data needs to be translated in vectors of numbers [Chr]. Images, sounds, text or similar categories are considered real-world data.

Neural networks are used to cluster and classify. There are a lot of applications where they are used, some of them being:

- **Classification** (depend upon labeled datasets): face detection, object identification, detect voices, classify spam emails
- **Clustering** (grouping by detecting similarities): document searching, image comparisons, anomaly detection

2.3.1 Neural Network Elements

A neural network is composed in multiple **layers** (Figure 2.4), each of them containing **nodes** (Figure 2.5).

The neural network is constructed from 3 types of **layers**: [Gav]

1. **Input layer** - the initial data feed in the neural network. Only one layer of this kind existing in a given neural network.
2. **Hidden layer** - intermediate layer between input and output where all the computation is done. Depending on the architecture of the neural network, it can contain one or multiple hidden layers.

3. **Output layer** - produce the result for the given input data. Like the input layer, a Neural network contains only one output layer. Each node from the output layer represents one label from the set of input labels. [Chr]

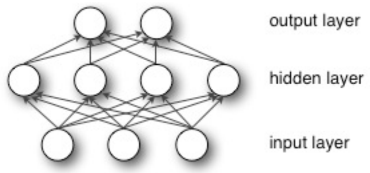


Figure 2.4: Layers structure [Chr]

A **node** (Figure 2.5) is a place where computations happen, like a **neuron** of the human brain, which fires when in encounters sufficient stimuli. A node combines the input data with a set of **coefficients (weights)**, that either amplifies or dampens the input, in other words, assigns a **significance** to the input with regard to the task the algorithm is trying to learn. These computations between the input and the weights are then summed up and then passed through a node named **activation function** to the next layer, to determine whether or not, and to what extent, that signal should progress further in the layers network. If the signal passes through, the neuron has been "activated" and the value is passed to the next layer [Chr]. The neuron output formula is [Hea15]:

$$f(x_i, w_i) = \phi(\sum_i (w_i * x_i)) \tag{2.1}$$

,where x and w represent the input and weights of the neuron, i corresponds to the the number of weights and inputs, and ϕ represents the activation function.

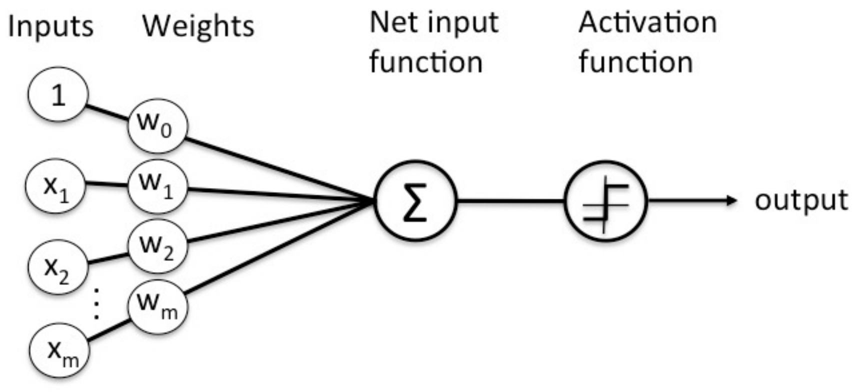


Figure 2.5: Node structure [Chr]

The **activation function** has the role to determine if a given node should be "activated", as stated above. In the creation of the neural networks, many kind of activation functions are used, starting from constant functions ($f(x) = c$) and linear functions ($f(x) = a * x + b$) and going to more popular ones like Sigmoid function. The **Sigmoid function** is one of the most widely used activation function, which allows the nodes to take any values between 0 and 1 [Gav]. It has the following equation:

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1} \tag{2.2}$$

After the predictions are made, the prediction vector is compared to the actual ground truth label, measuring the difference between those values using a so called **loss function**. Minimizing the loss function causes the neural network to make better predictions, two of those functions being the **Cross-Entropy Loss** and **Mean Squared Error Loss**. [Art].

2.3.2 Feedforward and Backpropagation

Feedforward neural networks are networks in which the output from one layer is used as an input in the next layer, the data always being feed forward and never back, which means that there are no loops [Nie15]. For example, the input layer has a connection to the first hidden layer, but no connection in the reverse order, meaning the input layer can send the output value of the node to the next layer, but the hidden layer can't feed the output to the input layer.

The **gradient descent** is the process that uses the gradient of a loss function (the derivative of the function) to improve the weight for the following epochs [Art]. In other words, it is an optimization algorithm that is used to find the coefficients of a function that minimizes a cost function [Dha].

Backpropagation is a technique used to train neural networks, by firstly computing the gradients of the cost function with regard to every parameter of the model, and performing a Gradient Descendent step after. In other words it is able to find how to tweak every connection weight and each bias to reduce the error, using only two passes through the network (one forward and one backward). [Gér19]

2.4 Deep Learning

Deep learning is considered to be one "relatively new advancement in neural network programming" [Hea15]. This is referring to the training process of deep neural network, namely any neural network that has more than two hidden layers.

Even though the main concepts of deep learning were well understood in 1989 (CNN and backpropagation), it started to be used in masses later on, around 2012.

The main technical obstacles were the **hardware** and the **data**. During the last 3 decades, CPUs became faster by a factor of approximately 5000, and GPUs can be used as well in training neural networks, one example being the CUDA library from NVIDIA. The successful rise in data had as factor the huge rise of the internet, which made the process of collecting and distributing datasets simpler. [C+18]

The **Rectified linear unit (ReLU)**, (2.3), is one of the standard activation function used for the hidden layers of such neural networks. Additionally, for the output layer, the neural network will use either a **linear** or **softmax** activation function, depending on what the neural network is used for, regression or classification [Hea15].

$$\Phi(x) = max(0, x) \tag{2.3}$$

The **Convolutional Neural Networks** are similar to traditional ANNs regarding the fact that they are comprised of neurons that optimise through learning. Each neuron will still receive an input and perform an operation and the entire network will still express a single perceptive score function (weight) The last layer will contain loss functions associated. The only notable difference between a CNN and an ANN is the fact that the CNNs are used in the field of pattern recognition within images, which allows the encoding of image-specific **features**, making the CNN more suited for this kind of tasks. [ON15]

Overfitting appears when the model is performing well on the training data, but it does not generalize well. The high number of parameters used in deep neural networks can result in overfitting, and regularization methods are used to get rid of the overfitting problem. One regularization method is the **Dropout**, which can be seen in Figure 2.6. Dropout is one of the most popular regularization methods, being a simple algorithm: at each training step, every neuron (excluding the output neurons) has a probability p of being temporarily ”dropped out”, meaning that it will be ignored during the respective training step, but it may be active in feature learning steps. [Gér19]

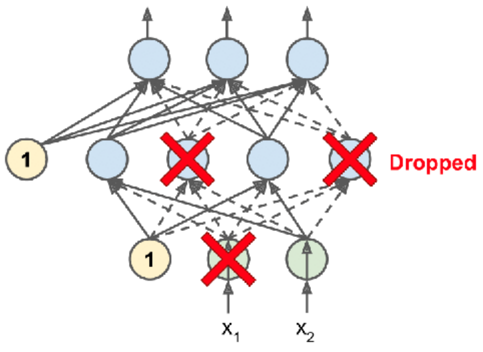


Figure 2.6: Neural network with dropout [Gér19]

2.5 Recurrent neural network (RNN)

Recurrent neural network (RNN) is a type of neural network specialized in processing sequential data of variable length, data such as the words of a sentence, the price of cryptocurrency in various moments of time, measurements of sensors over time.Recurrent neural network got this name because they apply a function over a sequence recurrently. It it similar to a feedforward neural network, excepting the fact that it also has connections pointing backward [VSS+19]. The recurrence relation of a RNN can be defined as:

$$s_t = f(s_{t-1}, x_t) \tag{2.4}$$

, where f is a differentiate function, s_t is a vector of values called internal network state at step t, and x_i is the network input at step t.

Unlike regular networks, where the state depends only on the current input and network weights, here s_t is a function of both the current input, as well the previous state s_{t-1}, which can be seen as the summary of all previous states [VSS+19]. The node structure of a RNN is presented in Figure 2.7.

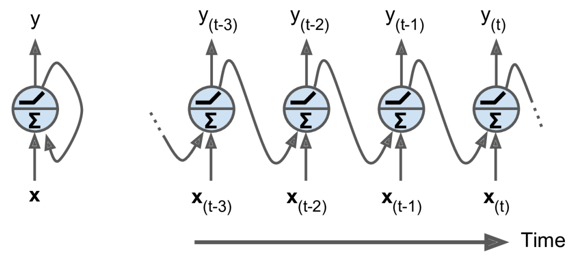


Figure 2.7: A recurrent neural network (left) unrolled through time (right) [Gér19]

2.6 Long short-term memory (LSTM)

Recurrent neural networks suffer from short-term memory, in other words, if a sequence is long, they will have a hard time in carrying the information from previous steps to later ones. **LSTM's**, presented in Figure 2.8 were created as a solution to short-term memory. Their internal mechanisms are called gates and are used to regulate the information flow. These gates are used to learn which data in a sequence is important and should be kept. [Mic]

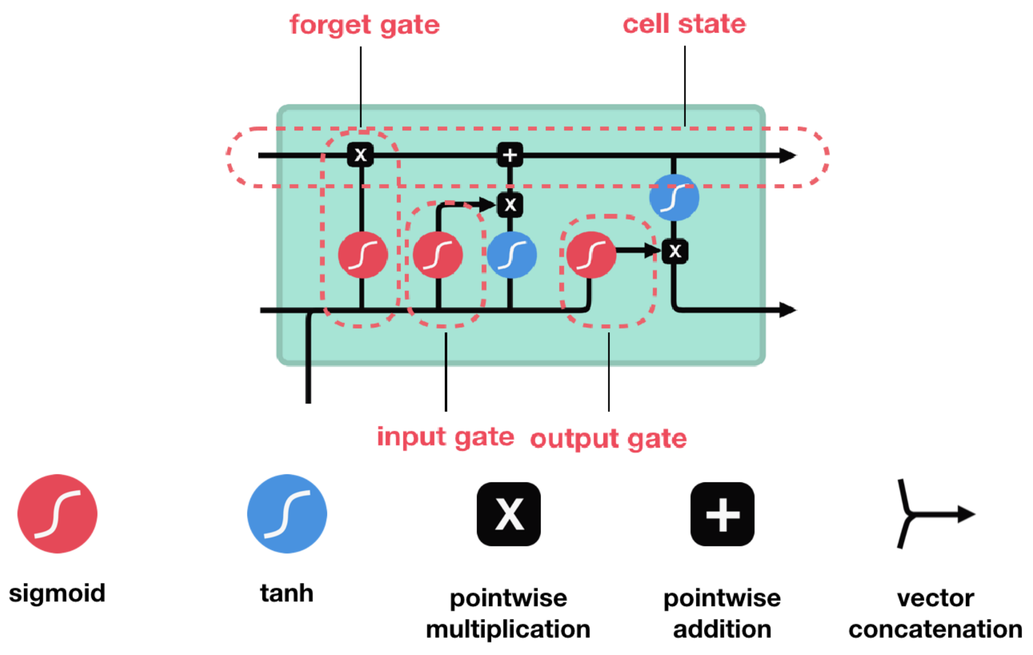


Figure 2.8: LSTM cell and and it's operations [Mic]

The **cell state** and the various gates are the main concept of LSTM's. The cell state can be seen as a transport highway that is used to transfer relative information through the sequence chain, carrying relevant information through the sequence chain. [Mic]

Every LSTM cell has 3 gates: **Forget gate, Input gate**, and **Output gate**. [Pur]

Forget gate

The forget gate decides which information should be thrown away from the cell, or kept, at a particular time stamp [Pur]. The information from previous states and from the current input are passed through the Sigmoid function. The values come out between 0 (data is meant to be forgot) and 1 (data is meant to be kept). [Mic]

Input gate

The input gate decides how much of the unit is added to the current state [Pur]. The previous hidden state and current input are passed into a sigmoid function, and they are also passed into a tanh function, and the two outputs are multiplied together. [Mic]

Output gate

The output gate decides what the next hidden state should be. The previous hidden state and current input are passed into a sigmoid function, the current modified cell state is passed through the tanh function, and the values are multiplied together. [Mic]

Chapter 3

Human Pose Estimation

3.1 Brief Introduction

The human pose estimation is defined as being the ”problem of localization of human joints”, and has enjoyed a substantial attention in the community of computer vision [TS14]. In other words, obtaining the coordinates of the joints of a person, in a specific coordinates domain (2D or 3D) from a normal RGB photo or video that can be found almost anywhere on the internet, or the mobile devices of every person. Those frames are usually taken using a mobile phone camera or other normal camera sensor. The process consists in extracting the the position of the joints in a given coordinate system from a frame.

3.2 Dimension Classification

The Pose Estimation problem can be classified, depending on the dimension of the output, into 2D Pose Estimation and 3D Pose Estimation. In 2D, the location of the joints, in terms of pixel values, are predicted in X and Y coordinates, without giving information about the depth of the frame. In 3D, a three-dimensional spatial arrangement of the joints is predicted. [Pra]. While some models are generating directly the 3D estimation, there are some models that are using as an intermediate stage the extraction of the 2D Pose and try to match it to a 3D estimation, ”based on high-level knowledge derived from anthropometric, kinematic, and dynamic constraints” [CR17].

The representation of the presented categories is presented in Figure 3.1.

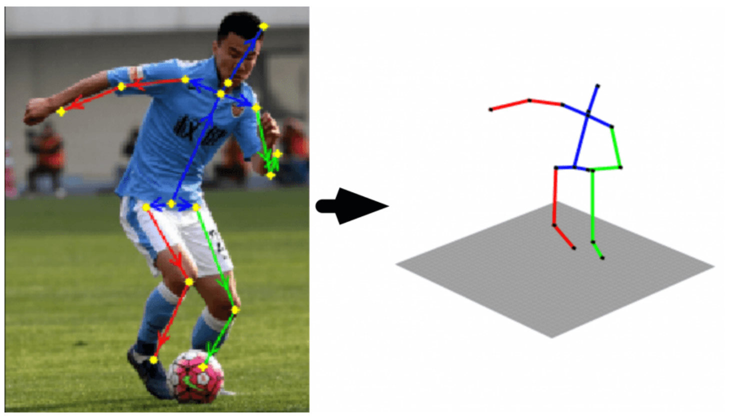


Figure 3.1: 2D Pose Estimation vs 3D Pose Estimation [Sta]

3.3 Body Models

One of the most important aspect of Human Pose Estimation is the human body modeling. This concept refers to how the keypoints that are extracted from the input data are represented. There are typically three types of models form human body modeling: [ZWY+20]

a **Skeleton based model** (Kinematic, Figure 3.2.a): is one of the most used models, and it consists in extracting the set of joints, being used in both 2D and 3D human pose estimation.

b **Contour based model** (Planar, Figure 3.2.b): consists of the contour and width of the body torso and limbs, capturing the relations between different body types. Used in 2D human pose estimation.

c **Volume based model** (Volumetric, Figure 3.2.c): consists of 3D human body shapes and poses represented by volume-based models with geometric meshes and shapes, normally captured with 3D scans.

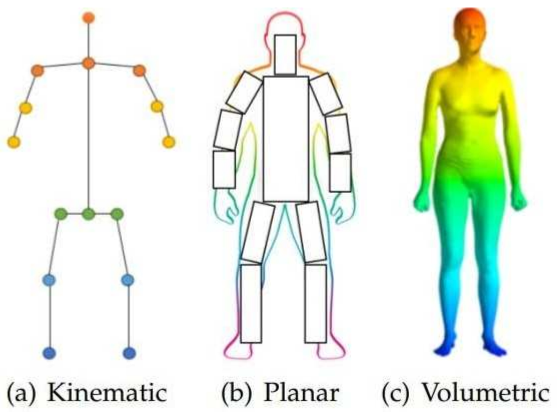
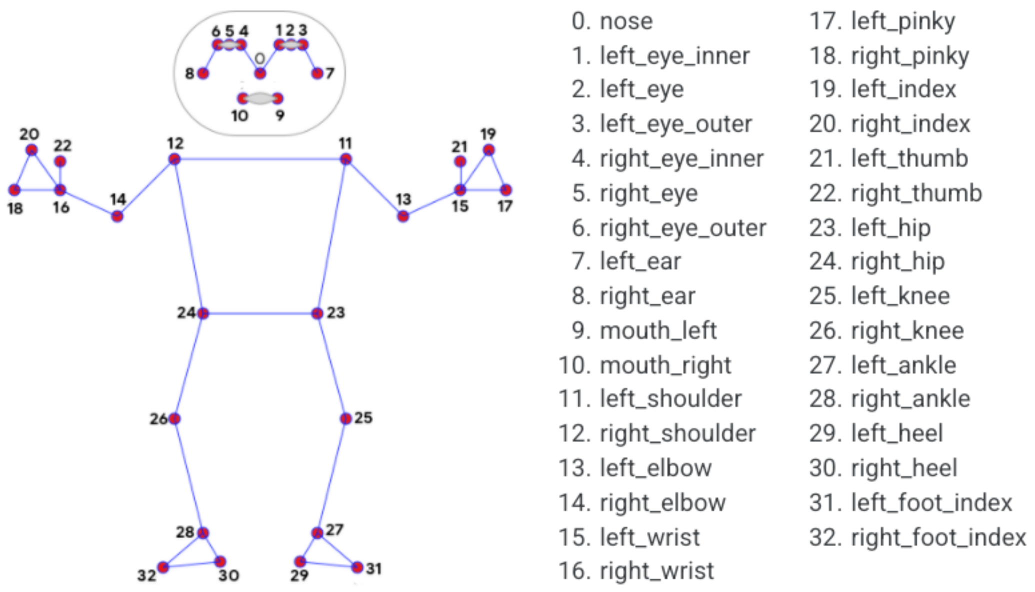


Figure 3.2: Types of human body models [ZWY+20]

In the following sections, the main focus will be on the Kinematic type, which is used by most pose estimation models.

3.4 Joints configuration

The configuration of the joints system is different for each model, all of them containing the most important parts, like: **the parts of the arm (elbow, hand, shoulder**, and **the parts of the leg (knee, foot, hip)** and others like the **neck**. Some of them contains additional coordinates for the face of the subject. In Figure 3.3 the representation used by Google for MediaPipe [Goo] can be found.



0. nose
1. left_eye_inner
2. left_eye
3. left_eye_outer
4. right_eye_inner
5. right_eye
6. right_eye_outer
7. left_ear
8. right_ear
9. mouth_left
10. mouth_right
11. left_shoulder
12. right_shoulder
13. left_elbow
14. right_elbow
15. left_wrist
16. right_wrist
17. left_pinky
18. right_pinky
19. left_index
20. right_index
21. left_thumb
22. right_thumb
23. left_hip
24. right_hip
25. left_knee
26. right_knee
27. left_ankle
28. right_ankle
29. left_heel
30. right_heel
31. left_foot_index
32. right_foot_index

Figure 3.3: Joints positioning [Goo]

Depending on the dimension of a given model, each joint will be represented as a 2 points coordinate or 3 points coordinate, some offering an extra variable representing the probability of the joint being detected (visibility).

3.5 Approaches used

Depending on the number of persons in a frame, the pose estimation can be divided in the following pipelines: **Single-Person Pipeline** and **Multi-Person Pipeline** [DYWZ19].

Single-person pipeline

Single-person estimation approaches, seen in Figure 3.4, are used to obtain the pose of a human in an image or a video. In early works, the human parts were modeled as a stickman, but recent works are modeling it as key joints because such joints are connected naturally and also give a position which is more accurate [DYWZ19]. There are two typical frameworks for the pipeline: **regression based methods** (regression methonds learn directly from a mapping, using a deep neural network, from the original image to the kinematic body model producing the coordinates of the joints) and **heatmap based methods** (body part detection using heatmaps aim to train a body part detector to predict the positions of body joints) [ZWY+20].

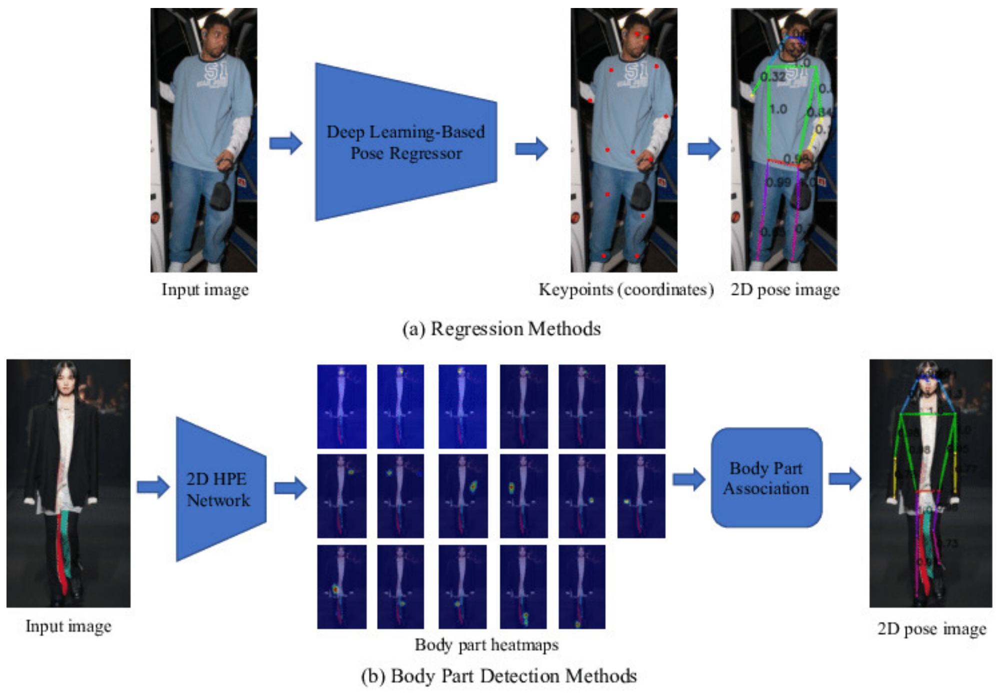


Figure 3.4: Single person pipeline. (a) Regression based. (b) Heatmap based. [ZWY+20]

Multi-person pipeline

The multi-person pipeline, seen in Figure 3.5, is harder and more challenging in comparison with the single-person one, because it needs to figure out the number of people and their position in the frame, none of the details being given. In order to solve these problems, the methods can be classified into: **top-down approach** and **bottom-up approach** [ZWY+20].

In the **top-down pipeline** there are two important parts: a human body detector and a single-person pose estimator. The first step is to detect all persons, represented as bounding boxes, that appear in a frame, and after, for each one of them, predict the locations of keypoints [ZWY+20].

In contrast, the **bottom-up approach** has a reversed procedure. The pipeline starts by extracting all the body parts, then it associate the parts to human instances. Is considered that the inference time for the bottom-up approach is likely to be slightly faster because it does not need to detect the pose for each person separately. [DYWZ19].

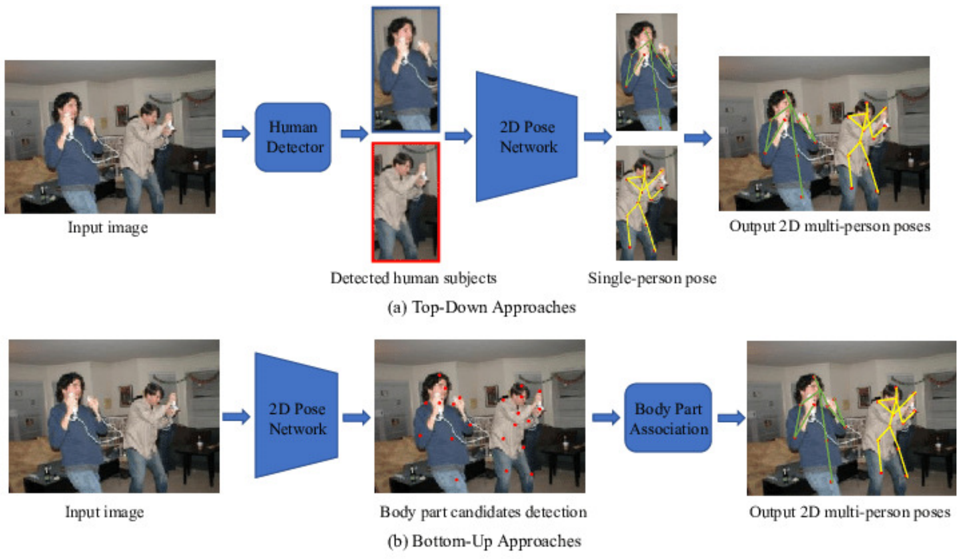


Figure 3.5: Multi-person. (a) Top-down and (b) Bottom-up approaches. [ZWY+20]

3.6 Available models

OpenPose

OpenPose is a popular bottom-up model for multi-person human pose estimation, offering detection for 135 keypoints, including body, foot, hand and facial keypoints. It is available on different platforms, such as Ubuntu and Windows, and provides support for different hardware, ranging from CUDA GPU's to OpenCL GPU's [CHS+19]. It works well on devices that have powerful GPU's, giving good frames rates and accuracy, but the performance on devices that use only CPU's is pretty bad, giving low frame rates. It is not recommended for day-to-day usages.

Mediapipe(BlazePose)

Mediapipe is a new single-person framework developed by Google, that is still in alpha phase, which includes multiple solutions (pose, face, iris, object detection, etc), and offers detection for 33 keypoints [BGR+20]. It is available on a large range of devices, such as Android, IOS and computers (desktop and web applications), and offers good performance even on phones, having low latency for devices such as Pixel 3 or MacBook Pro from 2017. [Goo].

For this thesis, MediaPipe was chosen for the human pose estimation part, because it can be runned on devices that only have CPU's and still give good frame rates, and also is easy to implement, and can be extended even on mobile devices.

3.7 Kinect

The Human Pose Estimation is a software solution for detecting the position of important parts of the body (joints), while the **Kinect** is one of the hardware solutions that are available. The Kinect 360 is produced by Microsoft, which created a SDK that can be used by developers to create apps using the available sensors of the device.

Microsoft has licensed the technology from PrimeSense (company specialized in the 3D sensing area, which was bought by Apple in 2013). The hardware of the Kinect is produced by PrimeSense, and it is responsible for the depth computation done by the device. [Mac11]

3.7.1 Kinect specification

The Kinect has a series of components that helps in in extracting the details of the human position, which can be seen in Figure 3.6. It contains a **microphone array**, a **tilt motor**, a **color sensor** (RGB camera) and a **depth sensor** (containing an **IR Emitter** and an **IR Depth Sensor**). The specifications of the Kinect for XBOX 360 are presented in Table 3.1.

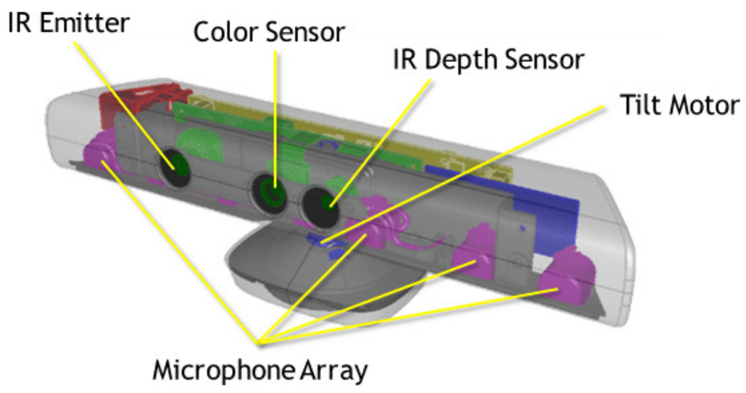


Figure 3.6: Components of the Xbox 360 Kinect [GRMHOR14]

Categories	Specifications
Sensors	Colour and depth-sensing lenses Voice microphone array Tilt motor
Field of view	Horizontal field of view: 57 degrees Vertical field of view: 43 degrees Depth sensor range: 1.2m - 3.5m
Skeletal Tracking System	Tracks up to 6 people, including 2 active players Tracks 20 joints per active player Ability to map active players to LIVE Avatars

Table 3.1: Kinect specifications [Luk]

3.7.2 How it works

The process in which the Kinect detects the body position can be seen in Figure 3.7. For inferring the body position, the Kinect is using a two steps process: compute the depth map then infer body position [Mac11].

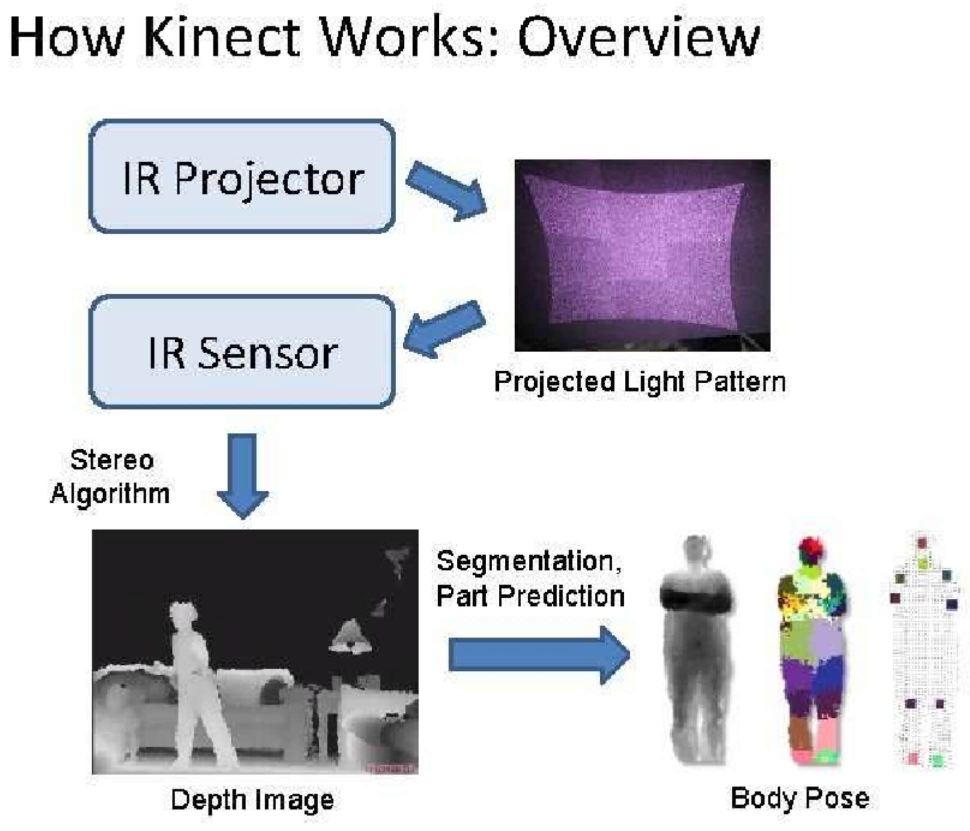


Figure 3.7: How Motion Detection Works in Xbox Kinect [Hoi13]

In stage 1, ″the depth map is constructed by analyzing a pattern of infrared laser light″ [Mac11], using the 3D depth sensor, presented in Subsection 3.7.1. The principle is called structured light, which ″projects a known pattern onto the scene and infer depth from the deformation of that pattern″ [Mac11].

The structured light principle is combined, by the Kinect, with two other computer vision techniques that are considered classic in the domain: depth from focus and depth from stereo. The **depth from focus** technique is based on the principle that the stuff that is further away is more blurry. The Kinect successfully improved the accuracy of traditional depth from focus by using a special astigmatic lens with different focal length in X- and Y- directions [Mac11].

The **depth from stereo** techniques is based on the idea that ″if you look at the scene from another angle, the stuff is close gets shifted to the side mode than stuff that is far away″ [Mac11].

In Figure 3.7, in the depth image picture, the image is seen in tones of white and black. The darker the color tone of the pixel is, the further the pixel is in comparison with whiter tones.

Stage 2 is based on the process of inferring body parts using a randomized decision forest approach, which learned from over 1 million training samples. This process is composed of 2 substages: **″first compute a depth map, then infer body position″** [Mac11].

In the first substage, the Kinect starts the process with approximately 100,000 depth images with known skeletons, using computer graphics techniques to render sequences for 15 different body types, resulting in a total number of over one million training examples.. The first substage ends by using a randomized decision forest (more sophisticated version of the classic decision tree), mapping depth images to body parts. A randomized decision forest is using multiple trees to learn, and using a random selection from the large number of possible questions.[Mac11]

During **substage 2**, the image containing the body parts is transformed into a skeleton, containing the 3D Joints coordinates, using the mean shift algorithm [Mac11].

Chapter 4

Proposed approach

4.1 Problem definition

The Human Pose Estimation (Chapter 3) is one of the most interesting application of machine learning, and it can be used to create many kinds of software applications that can be used in different domains. One such domain is the sport one, in which many uses can be found, like being used in the recovery of an athlete, extracting stats of a player and analyzing the movement of a person.

In the introduction part, statistics about injuries were presented, (Section 1.2, page 1). The number of injuries can be reduced, by analyzing the correctness of the exercise that was practiced by someone. A software can be implemented to help a person that is not sure how an exercise is executed correctly, alerting the user about the possible mistake that was executed.

The human pose estimation can be used to extract the important body parts of a person and process this data to check how correctly an user is doing a specific exercise. The main aim of this thesis is to give a possible solution for this issue.

4.2 Approaches description

This thesis aims to present two approaches that can solve this problem, **one that is based on a code-only solution**, where the computations and checks are made by hand for each frame, and **one that is based on machine learning solutions**. In order to implement and show the way in which these approaches work, three exercises will be showcased: **squats**, **bicep curls** and **shoulder presses**. Also, both approaches use MediaPipe as the model for the pose estimation part.

In the following chapters, the approaches will be presented, followed by a Result section.

4.3 Approach 1: Based on handcrafted geometrical rules

The first approach is based on making manual computations of angles and distances between different keypoints of the human body, and also make checks on the current stage of the repetition and check constantly if the exercise is executed correctly. The scheme that describes this approach is presented in Figure 4.1.

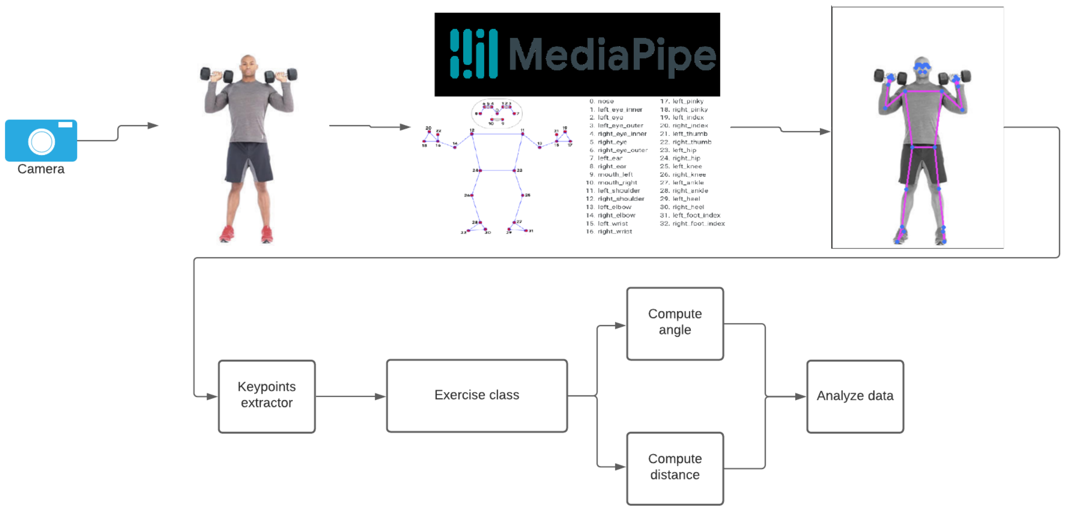


Figure 4.1: Geometrical rules based approach

The webcam captures records the user while doing an exercise available in the application, and passes the current frame to the MediaPipe pipeline that will return the position of all the visible keypoints in the frame. The keypoints are then used by the corresponding exercise class in order to compute the necessary information (angles and distances) needed in the class to check the repetition status and the correctness of the current frame.

In order to compute the angle between 3 points, the function presented in Listing 4.1 is used, that computes the angle between 3 points in 2D using the arctan2 function from numpy. The method will be used in the application constantly.

```python
def calculate_angle(a, b, c):
    radians = numpy.arctan2(c.y - b.y, c.x - b.x) - numpy.arctan2(a.y - b.y, a.x - b.x)
    angle = numpy.abs(radians * 180.0 / numpy.pi)

    if angle > 180.0:
        angle = 360 - angle

    return angle
```

Listing 4.1: Angle computation

In order to compute the distance between 2 keypoints, the euclidean distance was chosen to make this concept easier, and is presented in Listing 4.2.

```python
def euclidean_distance(a, b):
    return np.sqrt(np.square(a.x - b.x) + np.square(a.y - b.y)
        + np.square(a.z - b.z))
```

Listing 4.2: Euclidean distance

The checks for repetition status and for the correctness of the exercises are executed at each frame. For counting a repetition of the exercise, the repetition needs to have a start point and an end point, and return from the end point to the starting point. To count them correctly, the current status is stored in the specific class of the exercise, and it is a simple string that change value from **"start"** to **"end"** to know in which state the exercise is currently on (an example can be seen in Figure 3.1). Also, right at the start of the exercise, the status is set to None and wait for entering for the first time in the starting position. While being in the start status, the program waits for the user to get to the end position of the exercise, and vice versa. Shortly, the current state of the exercise needs to **start** from a position and **end** in another position.

As mentioned in the previous paragraphs, the angle is one of the main aspect that is checked. Considering the fact that the pose estimation can have some little errors (seeing a body part slightly to a side) in detecting the keypoints, the application will use an error when making the necessary checks. The error will be in the range of $10°$ and $20°$. For an example, when checking for the final position of a shoulder press repetition, seen in Listing 4.3, instead of checking if both hands are at $180°$, it will check for $165°$. Also, it checks if the elbows are above the shoulder level, by checking the y coordinate of the elbows and each side of shoulder.

```python
    def is_end(self, landmarks):
        angle_left = calculate_angle(
            landmarks[landmark_index["left_shoulder"]],
            landmarks[landmark_index["left_elbow"]],
            landmarks[landmark_index["left_wrist"]]
            )

        angle_right = calculate_angle(
            landmarks[landmark_index["right_shoulder"]],
            landmarks[landmark_index["right_elbow"]],
            landmarks[landmark_index["right_wrist"]]
            )

        return angle_left >= 165 and angle_right >= 165 and
        landmarks[landmark_index["right_elbow"]].y < landmarks[
    landmark_index["right_shoulder"]].y and
```

```
        landmarks[landmark_index["left_elbow"]].y < landmarks[
    landmark_index["left_shoulder"]].y
```

Listing 4.3: End check for shoulder presses

The part that is checking for incorrect movements is similar to the repetition status part, that is presented above. It is checked each frame of the execution, and it only responsible for looking for different kind of mistakes. For example, a mistake for the shoulder press is about the overextension of the arms while lifting weights (the arms are going to exterior instead of raising them straight. Also, a mistake for squats is when the knees are getting closer, which is checked using the euclidean distance and a maximum distance.

4.4 Approach 2: Machine learning based

The second approach is based on a machine learning architecture, and uses two neural networks to simplify the flow of execution. The first neural network is used to classify the status of repetition, that can be used in counting the number of repetitions. The second one is checking constantly for possible errors in the execution of the exercise. They are described in the following subsections.

4.4.1 Repetition counter

This model has the role to find the current state of the exercise. The exercise repetition status is divided similarly with the one presented in the first approach (Section 4.3, but it has another additional label. The labels are: **start**, **end** and **other**, presented with an example in Figure 4.2.

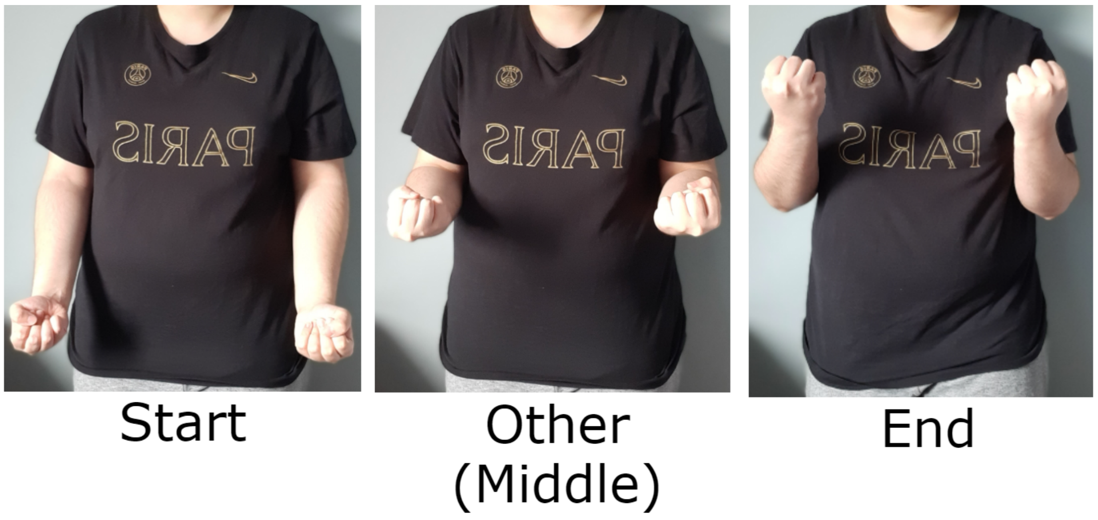


Figure 4.2: Repetition labels for biceps curls

The start and end labels have the same concept as in the first approach, and the other label is used for stage that is between the start and end statuses, in other words, the movements that are executed during the exercise, and are not considered a starting position or and ending position. First the dataset is presented with the number of frames found in the specific label of each exercise, followed by the description of the model architecture.

Another important aspect about the model is the fact that for the exercises that work on arms (bicep curl and shoulder press) only the upper body is needed, so the dataset will contain only the positions of each keypoint from the upper body are needed. For squats, the entire body is necessary.

Dataset

The process of creating the dataset started with collecting images corresponding to each label of each exercise from the internet, trying to include as many individuals as possible. In addition, recorded videos of myself performing the specific label were added (short videos that contained the corresponding status label from different angles). The frames that were collected have been passed through the MediaPipe pipeline that retrieved the necessary joints and collect all the results and store them in a csv file for each exercise, that resulted in 16.5 MB of csv, in comparison with the size of 577 MB from the media files. So a considerable amount of storage was saved in the process.

The statistics of the dataset for each exercise and each separate label are described in the following sub parts.

Bicep curl

For the bicep curl only the upper body is considered necessary, so only the keypoints of the upper body are stored in the rows of each file of the dataset (25 compared to the total number of 33). The full statistic about the file is presented in Table 4.1.

Label	Number of images	Number of videos	Number of frames	Number of individuals
Start	131	2	1530	36
Other	59	2	1132	10
End	88	2	2045	32

Table 4.1: Dataset statistics for bicep curl

In total, the dataset used for bicep curl contains 4707 frames for the 3 labels, that sums up at 319 MB of media data on disk, which translates to a 8.4 MB CSV.

Shoulder press

As in the case of the bicep curl dataset, the shoulder press CSV file only stores data about the upper body keypoints. The statistics of the dataset are shown in Table 4.2.

Label	Number of images	Number of videos	Number of frames	Number of individuals
Start	163	2	1240	83
Other	83	2	940	16
End	131	2	729	74

Table 4.2: Dataset statistics for shoulder press

In total, the dataset used for shoulder press contains 2909 frames for the 3 labels, that sums up at 151 MB of media data on disk, which translates to a 3.2 MB CSV.

Squats

The squat is mainly used for training the legs, so the keypoints specific to the entire body are necessary to be stored in the dataset (some of them can be skipped in future application that use this dataset). In Table 4.3, the statistics are presented for squats.

Label	Number of images	Number of videos	Number of frames	Number of individuals
Start	139	3	1037	61
Other	96	4	1285	35
End	173	3	1068	79

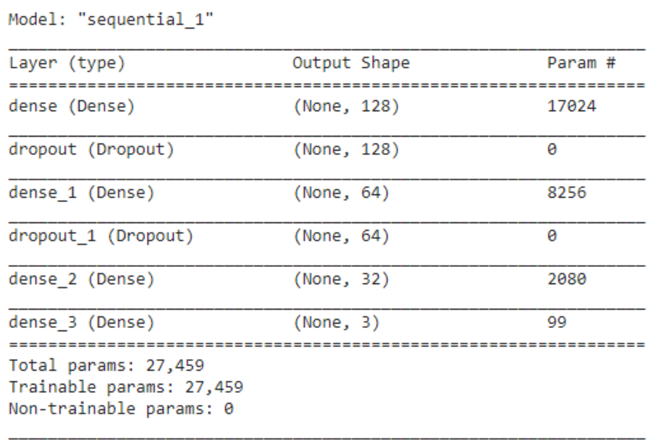
Table 4.3: Dataset statistics for squats

In total, the dataset used for shoulder press contains 3390 frames for the 3 labels, that sums up at 98 MB of images of different individuals and 10 recordings of 300 frames each (recorded directly using the MediaPipe model and saved directly in csv's, without storing them as separate videos), which are compressed in one CSV that has 4.9 MB.

Model arhitecture

The approach uses a neural network built using Tensorflow and Keras and uses the dataset described in the previous part as training and validation data for the model. The model architecture is an easier one, containing only some Dense and Dropout layers, having two Dense layers as input and output layers. The architecture can be seen in Figure 4.3 (the architecture shown is used for checking squats).

The model is checking for the position of the body parts of the person that appears in the video or stream of images (webcam stream) at some time during the sequence of frames. Depending on the type of exercise, the model will wait as input a sequence of 100 values (the upper body contains 25 keypoints, each containing the XYZ coordinates and also a visibility percentage) or 132 values (the entire body contains 33 keypoints.



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 128)               17024
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 99
=================================================================
Total params: 27,459
Trainable params: 27,459
Non-trainable params: 0
_________________________________________________________________
```

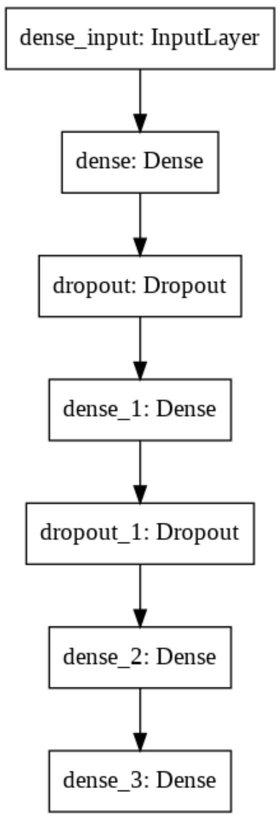


Figure 4.3: Model architecture for repetition status checker

As seen in Figure 4.3, the model contains only Dense layers, with 2 Dropout layer between them, expecting an input shape of (100,) or (132,), depending on the exercise that is checked in the respective neural network. The number of filters of the layers (parameters) was tested as well to see how it performs. Also, the output will be an array with 3 elements, corresponding to the possibility of each label to be presented at that time of the execution.

4.4.2 Correctness check

After the presentation of a possible way in which the current status of a repetition can be predicted using a neural network, the way in which the correctness of the movements during an exercise is described in this subsection. The model repetition can be considered an image classification problem, while the correctness check will be treated as a video classification problem.

This neural network need to check constantly for mistakes, looking at the current and some of the previous frames to predict the possible error in the execution. Depending on the type of exercise, only a part of the joints are needed for the model, exactly as for the model presented in the previous part. Each exercise has a variable number of labels for the model, one representing the good movement and the others representing the errors provided by the system. Each label will be described in the discussion about the dataset.

Dataset

The process of creating the dataset is similar to the one presented in the previous part (page 25), but, instead of images, videos were necessary for the dataset used for this model. Other difference is the fact that the previous dataset stores the data as a CSV file for each exercise, and for this dataset, each video is transformed into a CSV file containing the keypoints in the timeline of the execution inside the video. At some point, the recording was done directly using the pipeline offered by MediaPipe and stored in CSV files, which saved some storage space and some time.

In the following subparts, the dataset specification for each exercise is presented.

Bicep curl

The bicep curl model checks for good execution and 2 possible mistakes: **fast execution (fast)** and **usage of momentum (others)**. Usage of momentum means that other parts of the body help the bicep to perform the exercise (as example, if the elbow is not kept fixed in place, the shoulder and the triceps are also activated), resulting in a lower impact on the bicep.

If the curls are executed too fast, the effectiveness is reduced considerably, and small injuries can appear easier.

Table 4.4 presents the statistics for the dataset created and used for the bicep curl exercise. The total number of 183 records sums up to 95.271 (around 3150 seconds of media content) and has a total size of 2.1 GB of media files before converting them to csv files. The total size of csv files is reduced to 168 MB.

Label	Number of recordings	Number of frames	Number of individuals
Good	28	49284	15
Fast	84	9945	3
Others	71	36042	21

Table 4.4: Dataset statistics for bicep curl correctness

Shoulder press

For the shoulder press exercise only one mistake was chosen: **wrong extension of arms (bad)**. By wrong extension we mean that the arm(wrist) is not on the same line with the elbow (the X coordinates are not close to each other). There are not many injuries that can happen from this mistake, but the effectiveness is lower and you will get tired faster.

Label	Number of recordings	Number of frames	Number of individuals
Good	85	37760	22
Bad	51	22666	6

Table 4.5: Dataset statistics for shoulder press correctness

Table 4.4 presents the statistics for the dataset created and used for the shoulder press exercise. The total number of 136 records sums up to 60.426 (around 2010 seconds of media content) and has a total size of 1.5 GB of media files before converting them to csv files. The total size of csv files is reduced to 107 MB.

Squat

For squats, beside the good execution of the exercise, two mistakes were chosen for showcase: **bad leg movements (legs)** and **arching the back (back)**. By bad leg movements we mean mistakes like: getting up on the toes or inward knees (getting the knees closer to each other). These mistakes can result in bad knee injuries that will take some time to recover.

When arching the back during a squat, the back will activate, but this is not a good exercise for back, which can result in back pains. Try to keep the back straight to avoid ugly back injuries.

Table 4.6 presents the statistics for the dataset created and used for the squat exercise. The total number of 151 records sums up to 63985 frames (around 2130 seconds of media content) and has a total size of 2.3 GB of media files before converting them to csv files. The total size of csv files is reduced to 131 MB.

Label	Number of recordings	Number of frames	Number of individuals
Good	62	28516	26
Legs	55	16288	24
Back	34	19181	10

Table 4.6: Dataset statistics for squat correctness

Model Architecture

For the creation of this model, more model types were selected and tested to see which is better: **CNN**, **LSTM** and **CNN + LSTM**. Also, the model does not need all the data available in the dataset files, some keypoints are not considered to get faster predictions. For the upper body exercises, the face's keypoints are omitted, and for squats, the face and the hands keypoints are not needed.

CNN model

This is a classic approach in the deep learning community, where convolution layers are used together with pooling ones. The architecture is simple, combine Conv1D layers with MaxPooling layers to obtain a neural network configuration [Lin]. For the architecture, two configurations were chosen, one that contains only one pair that and one that contains two pairs that are combined using a Dense layer. Both models have as output layer a Dense layer that has as output shape an array which has as size the number of labels corresponding to a given exercise (between 2 and 3).

LSTM model

Considering that the model works on data processed from videos, so it has a temporal attribute present. LSTM is used on data sequences of different sizes which has a temporal/spatial attribute, so it is a good option for the architecture of the model proposed in this section. Different combinations were tested, one in which only a LSTM layer is found in the network and the result of this node is passed directly to the output layer and the other one in which two LSTM layers are implemented, being joined together by a Dense layer.

CNN + LSTM model

For a third architecture, the two models presented above were combined in one network and created [LWW+17]. The result obtained from the CNN part is passed to a LSTM layer which will send the result to a dense layer that is the output layer.

Chapter 5

Software Application

5.1 Specifications and requirements

5.1.1 Specifications

The main problem this thesis aims to tackle is described more precisely in chapter 4 (page 21). Shortly the human pose estimation can be used in a large and diverse number of areas, one of these areas being the sports domain. The movements of a person can be translated into a list of numbers corresponding to the position of each important part of a person's body. This chapter explains how an application that uses this information in order to check the correctness of the execution is described in this chapter.

The application is oriented on analyzing the movement of a person during an exercise. It should let the user record the exercise, via a Kinect 360 or a web cam, extract the important joints of the body, and give a feedback to the user according to the way the exercise is performed. The user should be able to select an exercise, a workout or perform freely.

5.1.2 Functional requirements

Keeping in mind that the application has the objective presented in the beginning of the chapter (Subsection 5.1.1), the functionalities that are proposed by the application are presented in Figure 5.1. Those functionalities are: *see the list of exercises, select an exercise, perform the exercise, give feedback to the user while performing the exercise.*

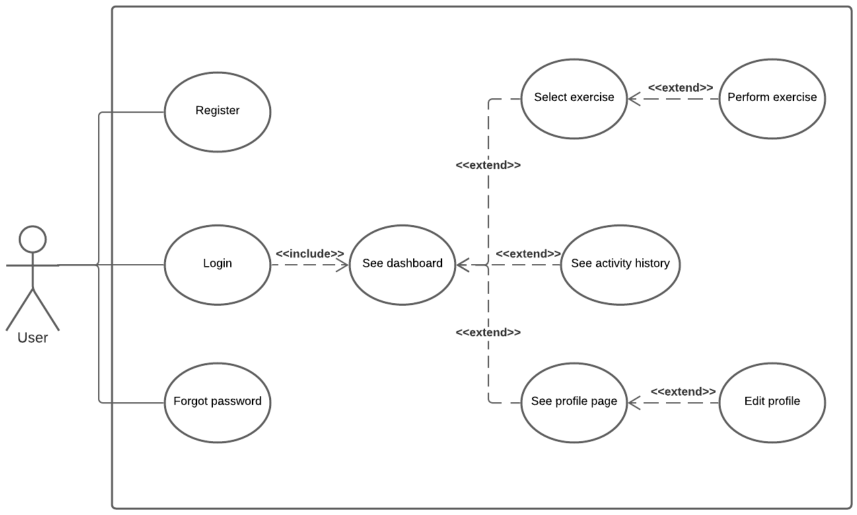


Figure 5.1: Use case diagram

The principal use cases are presented in the following list:

- *Login*: the user has the ability to authenticate and use the application
  - actor: user
  - precondition: the app is running and the login window is rendered
  - postcondition: the dashboard (main page) is opened if the login succeeds, or an error message is shown if the credentials are wrong
  - flow: the user opens the app, the login page is opened and rendered, and the user introduces the credentials for the application and presses the login page
- *Register*: the user has the ability to create an account
  - actor: user
  - precondition: the register button from the login window is pressed and the information is completed
  - postcondition: in case of success, the account is created, a welcome email is sent and the user is redirected to the login page
  - flow: the register button is pressed, the user introduces the necessary information and presses the registration button. In case of success a notification will appear and an email is received
- *See list of exercises*: the user should be able to see a list of exercises, from which an exercise can be chosen

  - actor: user
  - precondition: the user is authenticated and the exercises page is selected
  - postcondition: the list of exercises is shown
  - flow: the exercise page is selected and the list is rendered

- *Select an exercise*: the user should be able to select an exercise from the list of exercises
  - actor: user
  - precondition: the list of exercises is shown
  - postcondition: the exercise is choosen and a new page opens where the user can select the details of the exercise (number of reps, if the NN model should be used and also if the application should work on the webcam or the Kinect sensor)
  - flow: the list of exercises is rendered, the user selects one of them the details selection page is shown where the user can start the exercise

- *Perform exercise*: the user performs the selected exercise and get a feedback about the number of repetitions done and the mistakes mad
  - actor: user
  - precondition: the exercise is selected and the exercise window is opened
  - postcondition: the application captures the execution of the exercise and give feedback on the execution
  - flow: the exercise window is rendered and the application records the user on how it executes the exercise and give the specific feedback, if it is the case

5.2 Design

5.2.1 Architecture design

The system architecture is divided in multiple layers: the front-end app that is used by the user and the server which is the gate to the Keras model. The arhitecture diagram is presented in Figure 5.2.

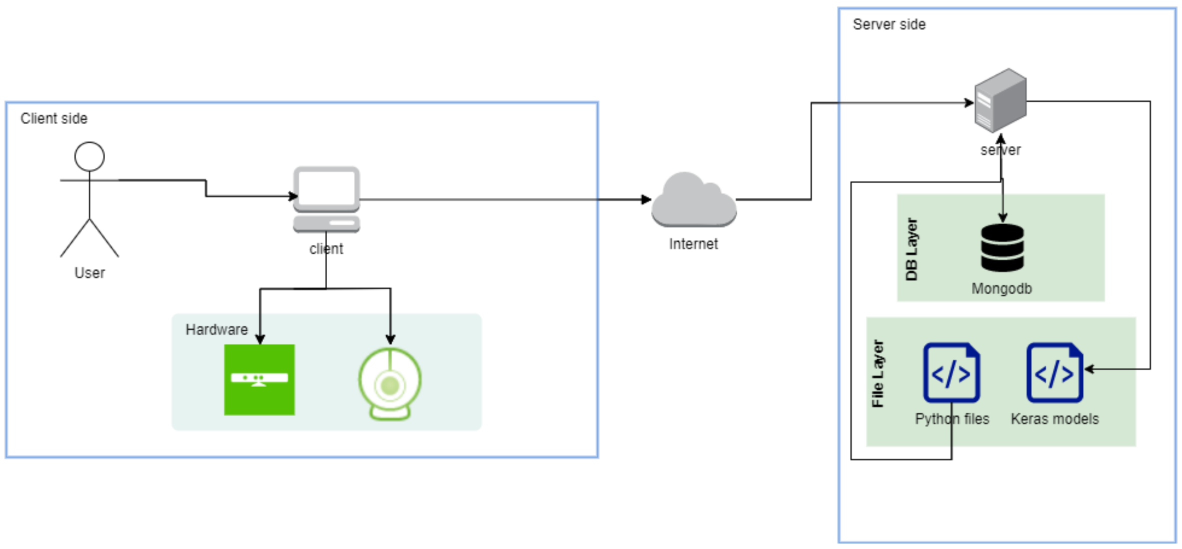


Figure 5.2: Architecture diagram

The front-end app is meant to be used by the user, in order to record the exercises and receive feedback. The application can use a Kinect camera or an ordinary webcam in order to offer the functionality of recording the user and let the application use the models to obtain the desired feedback. Also, it offers an authentication feature, such that each user can see their previous activities, and see how they evolved. The main application is built using Python and QT and is able to open a smaller .NET application that is able to use the Kinect to obtain (theoretically) better results.

The server, written in Python using Flask, is the gate between the front end app and the Keras models of the proposed approaches. It is able to store information about the user, and know where the model is stored in the storage space. Via an API call, the server will send the specific model, such that the front end is able to know what would be the correct way of doing the exercises. The model is sent to the server, such that the client application does not sent a large number of requests per second, reducing the time of prediction and possible server failures. Later, an offline usage can be provided, because the models are sent to the client, which stores them locally for further usage.

5.2.2 Class diagram

The application does not require a complex and numerous number of classes, the diagram being presented in Figure 5.3.

For each exercise, the name, the body region, the difficulty are stored in an object, such that the program can separate them on categories, giving the user the possibility to train a specific part of the body, or do an exercise of a given difficulty. Also, the exercise stores the location of the models used in the application, such that it can be retrieved later by API calls and used in the code by the client application, together with the json stored as a string, that stores the label with the corresponding values for each model (reps_labels and correctness_labels) and a separate variable that stores explanations for the correctness model (mistakes and why they are mistakes). Depending on the body region, the application will know what data to send to the model for the predictions of the models.

For the user, the email, password and name are the fields used for identifying the user, together with a field for knowing if the account was activated, and one that stores the location of the profile picture in the server.

When an user uses the application, the exercise performed is stored inside an activity object, that stores the date and time, the exercise object, the user that performed it and a list of sets objects. The set object stores the number of repetitions and the number of seconds in which they were performed, also including the number of repetitions that were performed correctly. Also, additional details from the user, which can be related to how difficult the set was or if any additional weights were used.

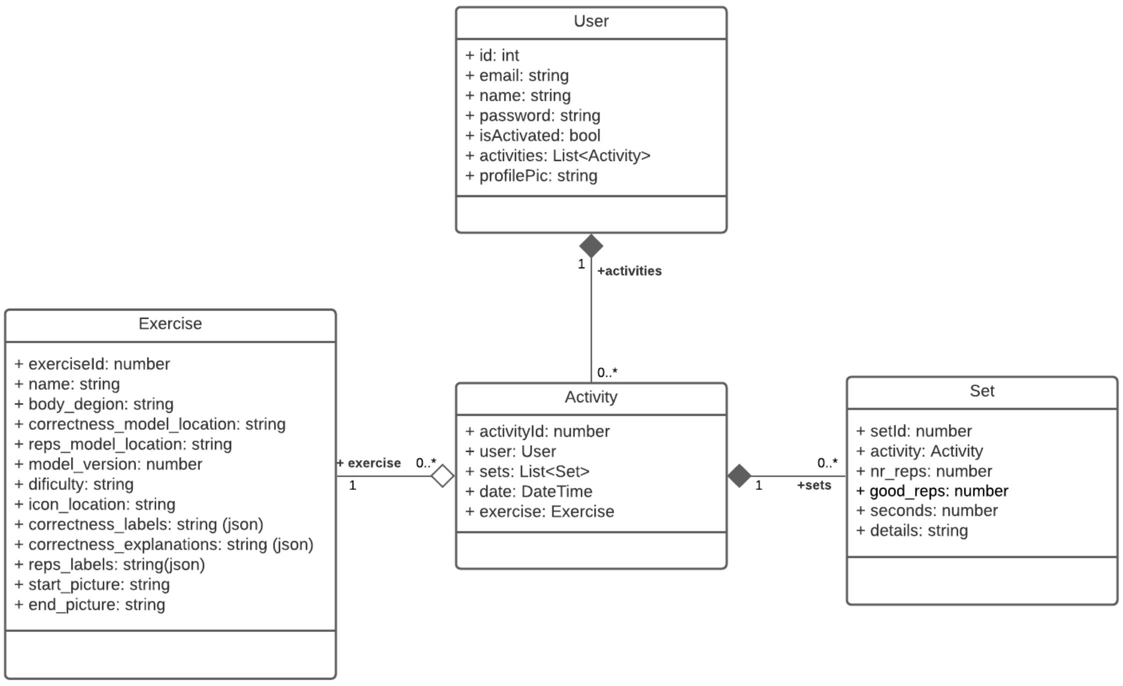


Figure 5.3: Class diagram

5.2.3 Database diagram

The database, presented in Figure 5.4, contains tables corresponding to each class from the Class Diagram presented in Figure 5.3.

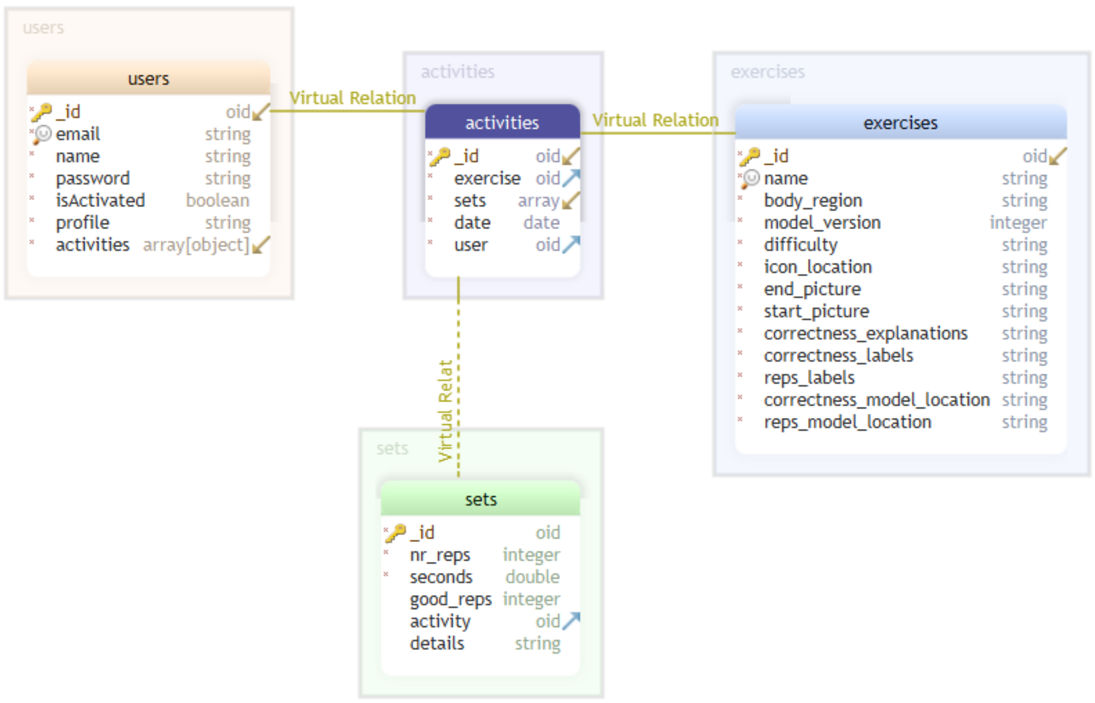


Figure 5.4: Database diagram

The database is mostly use to store the location of the neural network models used by each exercise, but also to keep an history of the sessions executed in the past, playing the role of a small journey, letting the user to see the progress made up to that moment. Using the information that are stored in the application, the application can provide the user with a detailed small journey, which is an important ”tool” used by a lot of fitness practitioners.

5.3 Technologies used

All the technologies used are presented in this subsection, in categories, in order to be easier to separate them.

Back end

Python, because is one of the best programming languages for machine learning, being dynamically-typed. It offers good compatibility with frameworks like

Tensorflow and Keras, being used by a large portion of programmers in a AI field of study.

Flask ([Gri18]) is a framework for Python that is used to create web applications or, in this case, servers that work on the REST principle, exposing some endpoints for an API.

C# (.NET) is a programming language invented by Microsoft which offers a multitude of options for developing applications, one of them being the a Windows Form Application that ensures an easier way to develop applications with GUI's. This language programming was mainly chose because it offers the best Kinect integration.

MongoDB ([GGPO15]) is a database system that uses NoSQL (an approach for database that does not use the classic relational model, which can become ineffective with large size of data). MongoDb can be fast in executing operations on large number of information, storing the data as documents.

Tensorflow because is a machine learning system which has a large community, good support and many information can be found about Tensorflow. Also, it provides great support for Keras.

Keras is an API used by many people for creating neural networks, in the deep learning department. It was chosen because it is pretty fast, offers good results, and also state-of-the-art-results.

OpenCV ([BK00]) is a comprehensive Computer vision library that provides solutions for a large range of problems, such as image and video processing. It was chosen because it provides a simple interface for Python and is not difficult to implement.

Front end

QT is a front-end solution that provides users with GUI components and opportunity to create large scale GUI applications. Also, the files generated using QT can be used in many programming languages, for example C++ and Python, so, a GUI file generated using QT can be used in a range of different types of systems.

Human pose

Kinect SDK for using the Kinect camera in the application. The SDK offers the the best skeleton position of the body pose estimation for the Kinect sensor, getting better results than a normal camera using an API (like Media pipe), because it uses depth sensors, making it more sure to obtain the best results for the 3D Pose Estimation.

Media pipe ([Goo]) is one of the newest API for pose estimation, developed by Google. It offers good results providing good frame rates for many configurations, having implementation for different programming languages, ranging from Python to JavaScript, even Kotlin and Swift, giving the possibility of enlarging the application to other devices in the future.

5.4 Implementation

The implementation of the application is divided in steps, which will be presented in this section, respecting the timeline in which they were approached.

Creation of the Dataset

One of the first step in creating the overall application was to create the datasets for the neural network models described in the previous chapter.

The process started with searching for images specific to each possible state for a repetition (start, end and others) and searching for videos specific to each good execution and mistake of an exercise. Additionally, videos recordings of myself were made to add to the existent pile of media to enlarge the size of the dataset. It was a pretty costly process considering the storage of each image and video. After the media was collected, each image and video was taken, frame by frame, by a preprocessing program that extracted the keypoint for each frame and stored them in csv files that will be used as datasets for training and testing the models.

The obtained dataset can be used in future enhacements of the application and by other developers to implement solution for fitness related applications.

A piece of code from the preprocessing function can be seen in Listing 5.1

```python
#new_file - the file where the result is stored
#pose - the object corresponding to the mediapipe model
#image - the variable in which the current frame details are stored

# Make detection
results = pose.process(image)

# Extract landmarks
try:
    landmarks = results.pose_landmarks.landmark

    row = list(np.array(
        [[landmark.x, landmark.y, landmark.z, landmark.visibility] for
    landmark in landmarks]).flatten())

    with open(new_file, mode='a', newline="") as out:
        csv_writer = csv.writer(out, delimiter=",", quotechar='"',
    quoting=csv.QUOTE_MINIMAL)
        csv_writer.writerow(row)
```

```
except:
    pass
```

Listing 5.1: Sample from the preprocessing function

Creation of the Neural Network models

With the datasets created, the neural networks were created. More details about the creation and the architecture of these models are already presented in Chapter 4 (page 4.4), and some results of these models are presented in Chapter 6.

Development phase: Server

The implementation continued with the development of the applications proposed. The first one that was created was the server, that has the role to store the obtained neural networks and send them to the client app when needed. Also, it provides a management system for users.

For the creation of the server, Python with Flask were the chosen combo, because they are pretty simple to implement and maintain. The server provides a REST approach that exposes endpoints as an API for other application to communicate with it. The authentication endpoint is shown in Listing 5.2

```
@app.route('/api/v1/user/login', methods=['GET'])
        def login():
            email = request.json.get('email')
            password = request.json.get('password')
            try:
                token = self.__user_service.login(email, password)
                return {"token": token}
            except Exception as ex:
                return {"error": str(ex)}, 501
```

Listing 5.2: Authentication endpoint

The authentication endpoint returns, in case of success, a JWT Token that is required as an Authentication header in a large number of exposed endpoints. This approach ensures that only users that are authenticated can make calls to the server.

For storing the information of exercises and users, the server uses a database powered by MongoDB, that was described earlier in Subsection 5.2.3.

Client application

With the server up and running, only the client application needs to be developed. In the beginning the GUI QT files were created for each window and the functionalities for these windows were created in the order of the GUI creation. The GUI interface can be seen late in section 5.5, where the User Interface is described together with the functionalities found on each window.

The client application has two programs written in Python and C. The main program is written in Python, where all the logic is used, and the C program is used to run the main feature using the Kinect Sensor.

5.5 Features presentation

In this section the user interface and the flow of the application is presented like in a user manual.

Login

The login page contains fields for the necessary credentials: email and password, a button for authentication, a register button that opens the register window where an user can create a new account, and a forgot password label which, if clicked, opens a modal where the user can type the email address, and the password is reseted and sent to the email. The interface of the login page is presented in Figure 5.5.a.

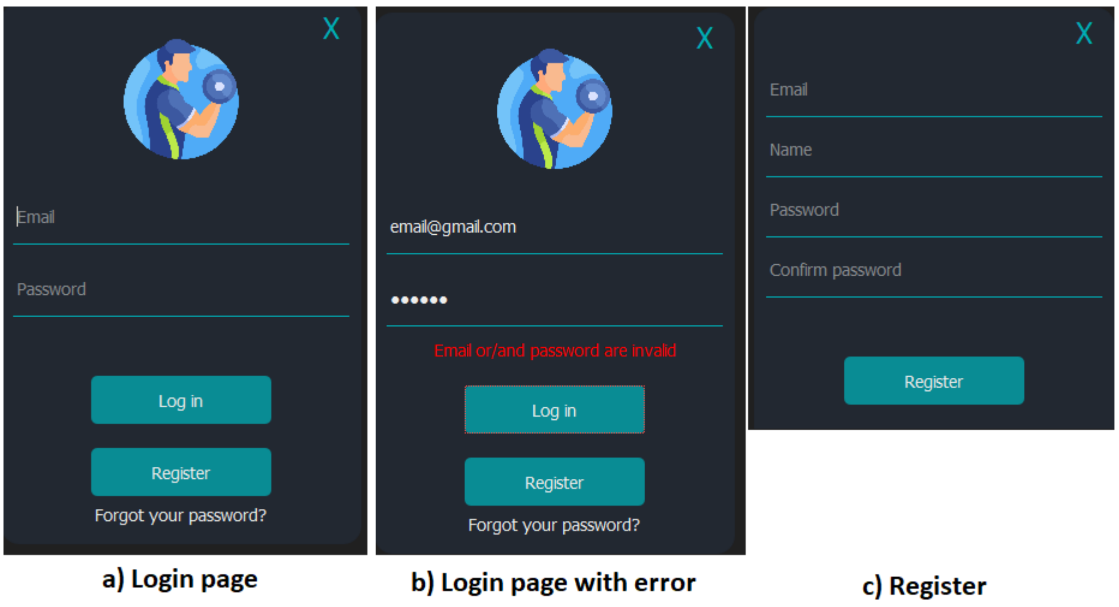


Figure 5.5: Login and Register windows

In the case in which there is no user with the given email, the password is incorrect, an error message will appear between the inputs and the Log In button, such that the user know about the mistake made. The error is presented in Figure 5.5.b.

Register

The Register window contains inputs for the necessary information needed by the application: email, name and password, also containing a field for confirm password and a button that will create the account when pressed. In case of insuccess a MessageBox will appear and notify the user about the error, mostly there is an existing account with that email or the two passwords do not match. If the registration is executed with success, the login page will be opened and a welcome email is sent to the corresponding email address. The interface is presented in Figure 5.5.c.

Exercise selection page

When the user successfully authenticated with an account, the main menu page is opened. This page contains, in the left side, different options, like opening the window containing the list of exercises available for training, opening the page which contains an historic with the previous activities done in the application and opening the profile page, where the user's information are available. In the main part of the main page layout, the current option selected from the left sidebar is selected. Also, at the bottom of the sidebar, a logout button is found that let the user sign out.

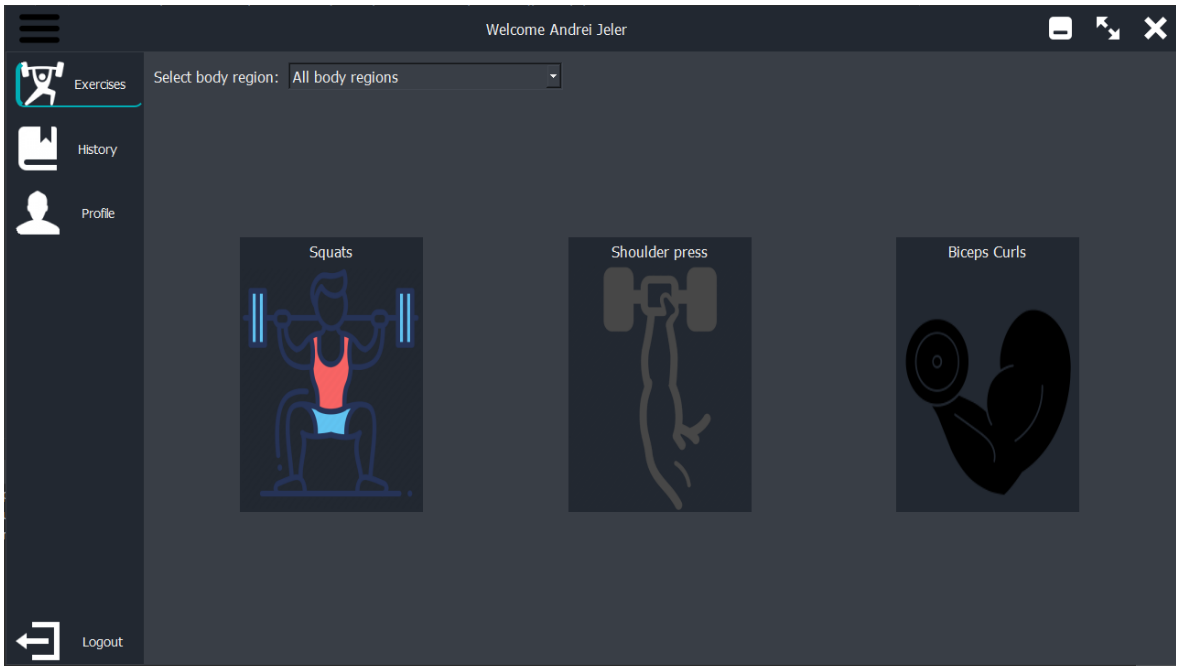
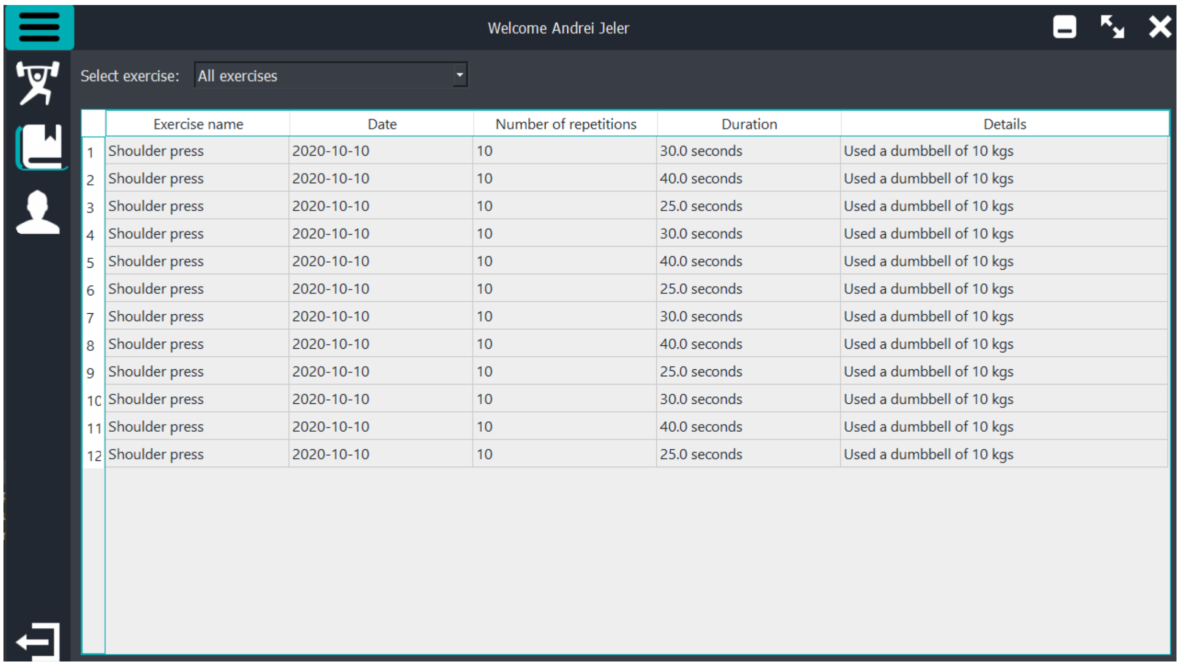


Figure 5.6: Exercise selection window

For showing the list of exercises available, a list in a grid layout is shown, together with a combo box that let the user select a specific body region to filter the exercises. The interface for selecting an exercise and the layout of the main page are presented in Figure 5.6.

Journey and user profile pages

The other windows available directly in the main page of the application are the history window (small journey), Figure 5.7 and the user profile Window, Figure 5.8.

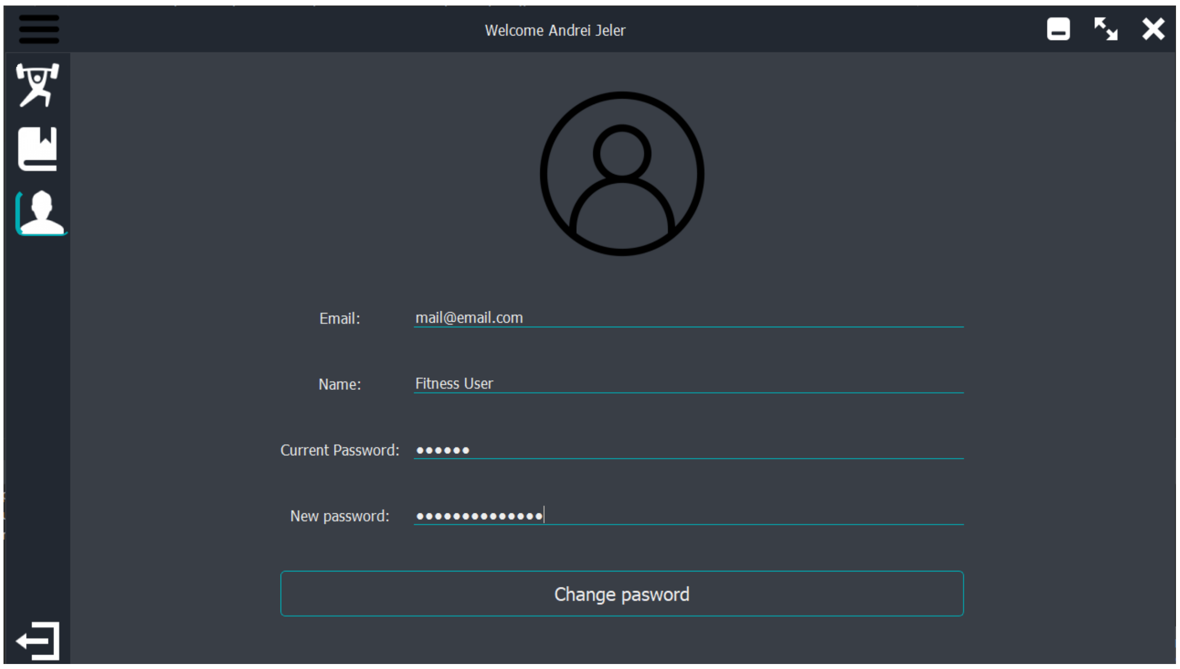


Welcome Andrei Jeler

Select exercise: All exercises

	Exercise name	Date	Number of repetitions	Duration	Details
1	Shoulder press	2020-10-10	10	30.0 seconds	Used a dumbbell of 10 kgs
2	Shoulder press	2020-10-10	10	40.0 seconds	Used a dumbbell of 10 kgs
3	Shoulder press	2020-10-10	10	25.0 seconds	Used a dumbbell of 10 kgs
4	Shoulder press	2020-10-10	10	30.0 seconds	Used a dumbbell of 10 kgs
5	Shoulder press	2020-10-10	10	40.0 seconds	Used a dumbbell of 10 kgs
6	Shoulder press	2020-10-10	10	25.0 seconds	Used a dumbbell of 10 kgs
7	Shoulder press	2020-10-10	10	30.0 seconds	Used a dumbbell of 10 kgs
8	Shoulder press	2020-10-10	10	40.0 seconds	Used a dumbbell of 10 kgs
9	Shoulder press	2020-10-10	10	25.0 seconds	Used a dumbbell of 10 kgs
10	Shoulder press	2020-10-10	10	30.0 seconds	Used a dumbbell of 10 kgs
11	Shoulder press	2020-10-10	10	40.0 seconds	Used a dumbbell of 10 kgs
12	Shoulder press	2020-10-10	10	25.0 seconds	Used a dumbbell of 10 kgs

Figure 5.7: Historic window



Welcome Andrei Jeler

Email: mail@email.com

Name: Fitness User

Current Password: ••••••

New password: ••••••••••••••

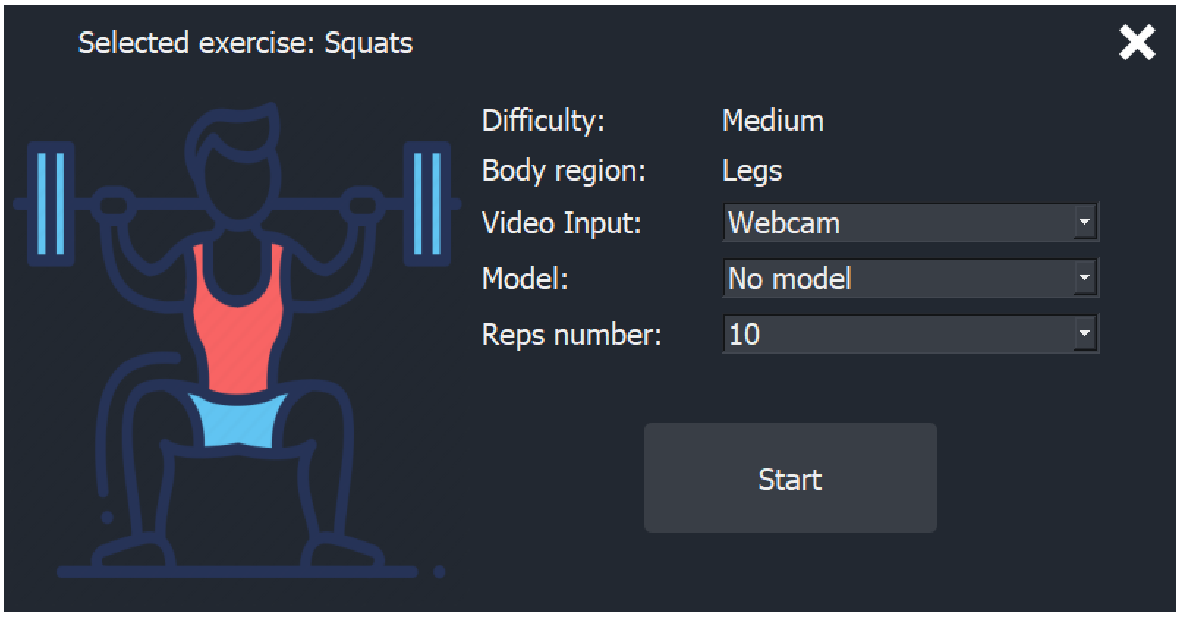
Figure 5.8: User profile window

For the window that plays the role of a fitness activity tracker a simple design was chosen, containing only a table that has as columns the important information of the previous sets (the exercise which was performed, the date, the number of repetitions, the number of seconds elapsed and additional details about it - additional weights or similar information), together with a combo box for selecting the exercise for which the user wants to see the historic. The recordings of the table are shown in decreasing order of the date.

The user profile page contains only the main information about the user, that are asked when a registration is made, the name, email, profile picture (can be changed in time) and the possibility to change the current password.

Exercise details selection

This page is mainly used for debugging reasons and it is the gateway between the window that show the list of exercises and the one where the recording happens. This window let the user select some parameters used for the window where the user can record the fitness activity or give a video to the program to be analyzed. The settings that can be changed in this window are related on how the exercise is going to be formed, namely the video input (select between the webcam, the Kinect or browse for a video locally), model usage (if the neural network models should be used or not) and the number of repetitions in a set (10, 12, 20 or unlimited).

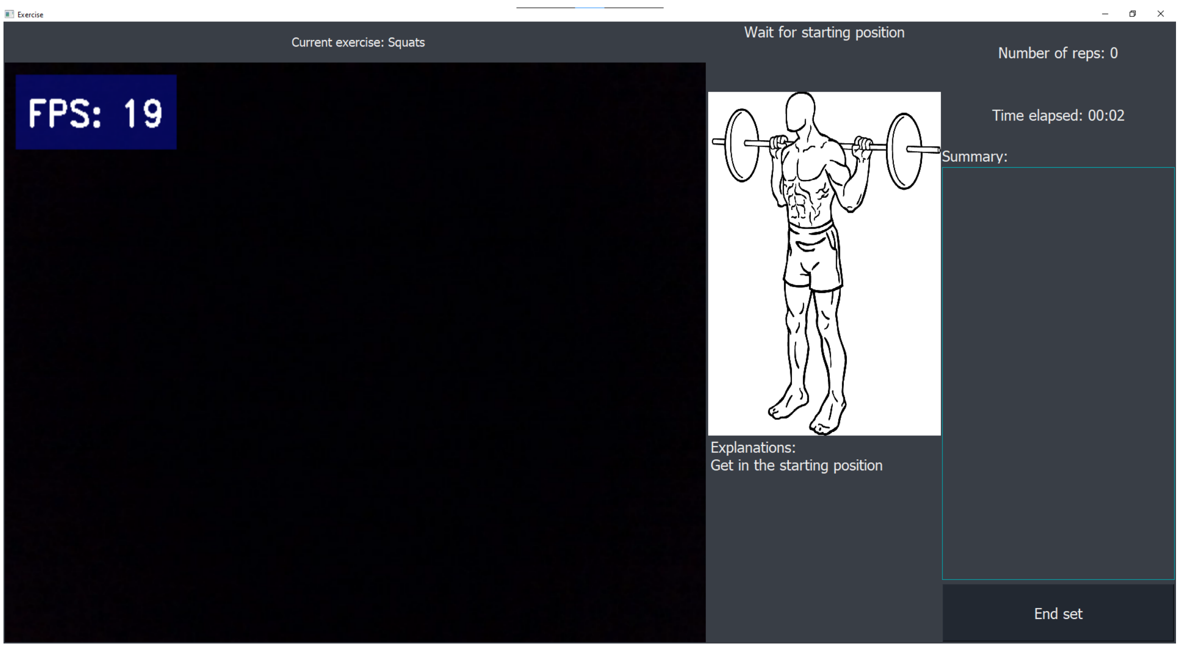

Selected exercise: Squats

Difficulty: Medium
Body region: Legs
Video Input: Webcam
Model: No model
Reps number: 10

Figure 5.9: Exercise details options

Exercise Page

In this window, the main functionality of the application is found. Figure 5.10 presents the way it looks like. In the left side, the person can see the live execution of the exercise, with an FPS counter in the upper left corner of the frame. In the right side information about the execution are found: the current state of the repetition, explanation about what the user need to do next (image + text explanation), the current repetition number, a summary frame which contains details about the possible mistakes and a button which can stop the current or start a new set. After a set is done, a modal is shown that lets the user to introduce additional information about the set that was done.



Explanations:
Get in the starting position

Figure 5.10: Exercise window

Analyzing the interface presented in Figure 5.10 the following information can be extracted, the user chose the squat as the exercise to be performed, the program waits for the user to start the execution, explanations about what position should be reached are available (get to the starting position), the user is currently at repetition 0, because none was made so far, 2 seconds elapsed from the beginning of the exercise and the summary is empty because nothing happened yet.

Chapter 6

Results and discussions

This chapter has the role to describe the results of the neural network models and discuss about the proposed approaches.

For the first approach some short outlines are presented, together with a short presentation of other available solutions. For the second approach, a more detailed discussion together with a presentation of the obtained results for each model and a comparison with other available similar solutions.

For testing the model, to see if it performs well, many tests could be performed. A first kind of classic testing would be to test the accuracy and the loss of the model after the fitting process, by dividing the data from the dataset into a training set and testing set. The result can be compared with similar architectures to see which one performs better. After that, the obtained result can be compared with similar solutions available on the internet, in order to see how well it performs in comparison with others and create a short state-of-art comparison.

Given the fact that the model runs on videos (sequences of a large number of images), the speed of executing the process can be measured in frames per seconds. Shortly, in how many frames per seconds the application can work. The app needs to record the actions, and perform live operations on the video.

The results were obtained on a machine powered by an Intel I7 7700HQ, 16 GB RAM and a Nvidia Geforce GTX 1060 Mobile.

6.1 Approach 1

For this approach there are not many ways to test the solution, mostly offering good results and good frame rates such that the application is usable.

The code based solution does not need large knowledge on difficult areas of software developing or machine learning, using easy concepts as computation of angles and distances. Only a small amount of code is necessary for implementing this approach.

Even though it is pretty simple to implement, this approach comes with some **disadvantages**. Every new exercise needs to implement the base exercise class and write new methods for each mistake of that exercise, so, in time it can be very costly to upgrade and the file system can become pretty big fast. Also, the estimation are not always 100%, as they should be, so, small errors should be implemented in checking the necessary conditions (small difference in angles or distances in the case of this approach).

A solution that can be found on the internet that is similar to the presented system is the project made by Jason Chin, namely the **phormatics** project ([Jas]). This project is also taking advantage of a human pose estimation model (namely OpenPose) in order to detect the possible 'critiques' in the execution of an exercise. It does not provide a repetition counter functionality and it is mostly using the angles of different joints to make the necessary checks.

6.2 Approach 2: Repetition status model

This checker (or classification) can be categorized as a image classification, in each one image is taken, the data from the frame is processed and sent to the neural network, that will make the prediction. In this section, details about different results are described, ending with a comparison with other similar solutions and their results.

For this model, the architecture was described in Chapter 4 and it is used for the results that are going to be presented. The architecture contains multiple dense layers, and the number of units from the parameters list of the layers, excepting the output one, will differ from a model type to another presented in the results. By shallow model we mean the model where the dense layers have as units the following values: 32, 16 and 8, the model resulting in having 4947 trainable parameters (for the squat model). For the deeper model, the number of units for the dense layers are: 128, 64 and 32, resulting in 27,459 trainable parameters.

Training phase

For training each model, the dataset was divided into **75% training data and 25% validation data**, and the **Adam optimizer (with a learning rate of 0.001)** was chosen for every model together with the sparse_categorical_crossentropy loss function from Keras. All the models were trained for **500 epochs**, because they needed a very short time for each epoch, $< 1 second$.

Accuracy and loss

For testing the quality of the models, the accuracy and the loss of each model are compared. For testing the higher the value is, the better the model is, and the loss needs to be as small as possible. Table 6.1 presents the final values of training for the models, presenting the accuracy and the loss of each model.

Exercise	Model type	Training Accuracy	Validation accuracy	Training loss	Validation loss
Bicep curl	Shallow	97.59	98.30	0.07	0.04
Bicep curl	Deeper	98.90	99.06	0.02	0.02
Shoulder press	Shallow	96.56	97.93	0.08	0.11
Shoulder press	Deeper	98.44	97.25	0.04	0.12
Squat	Shallow	98.22	97.99	0.04	0.06
Squat	Deeper	99	97.07	0.02	0.11

Table 6.1: Accuracy and loss for the repetition status model

As the table presents, the accuracy values of the models are all in the high 90's (over 95%) which provides pretty good predictions. The loss is also in the smaller range of values. The larger models have higher training accuracy in all cases, while the validation accuracy is higher only in the case of bicep curl. The average difference in training accuracy between small and large models is around 1.32%, while the validation accuracy difference average is around 0.78%.

The evolution of the accuracy value and loss value can be presented in a plot, as a function graph. The plots are very similar for the exercises, but they differ a little depending on the number of parameters of the model. In Figure 6.1 the plot curves for the small and large model for squats is presented. They are somewhat similar, but some differences can be seen.

For the smaller model, the accuracy for the training and the validation sets are similar, are increasing, having some occasional drops, having almost equal values at some steps. The loss for the smaller model is also getting better, and also the validation loss remains higher than the training most of the time.

For the larger model, both accuracies are growing up until a point, and then they variate, not having big differences from epoch to epoch, the validation one remaining lower than the training one. The loss for the training set is similar to the smaller model's accuracy, but for the validation set is getting bigger.

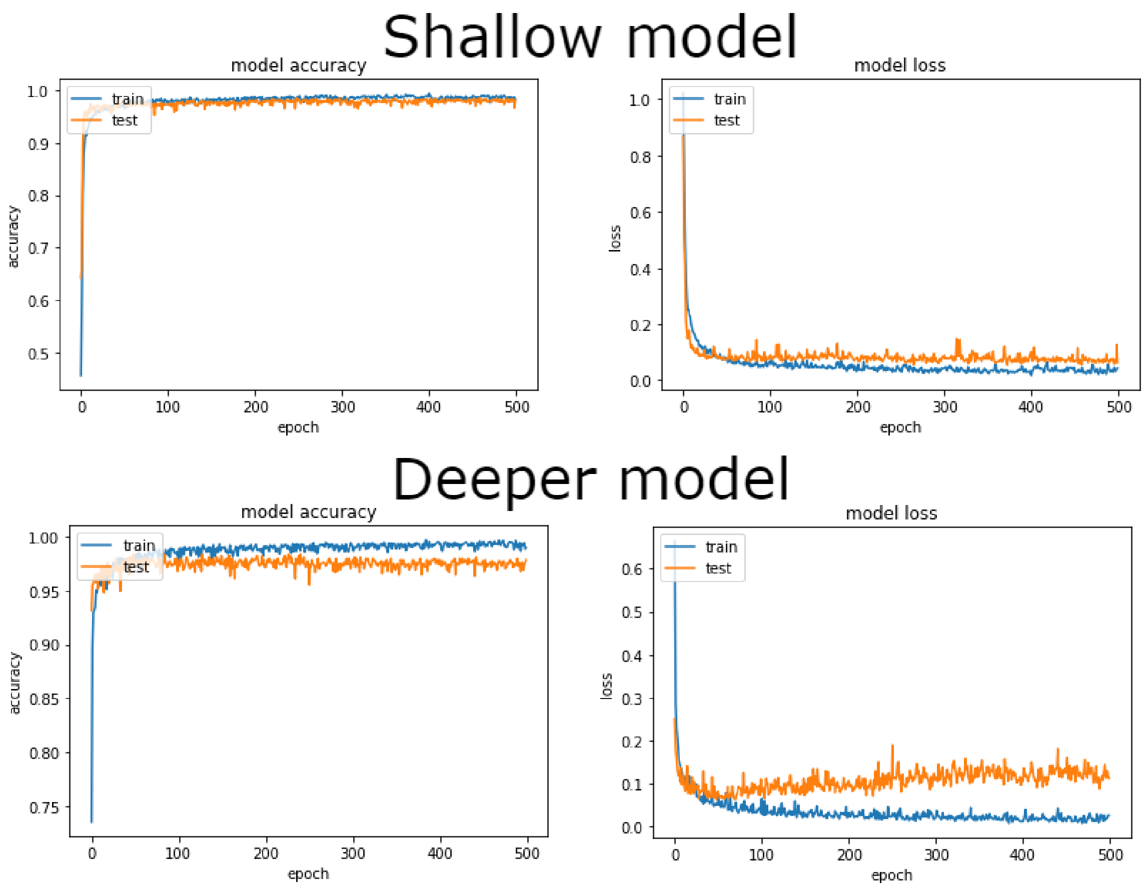


Figure 6.1: Statistics for the repetition status network for squats

Retrain

With the models trained, a retrain session was made for the short model for squats. During this session, the model was retrained for another 100 epochs and the final values for the accuracy and for the loss are presented in Table 6.2. An improvement can be seen, which is not the biggest one possible, but it results in a system that provide almost 99% accuracy.

Type	Training Accuracy	Validation accuracy	Training loss	Validation loss
Normal	98.22	97.99	0.04	0.06
Retrained	98.89	98.46	0.03	0.06

Table 6.2: Accuracy and loss for the retrained models

Performance testing

The high accuracy for each model is a good thing, but the performance aspect of a model is very important. In the context of the proposed application, there is a need for good frame rates such that the application can run as smoothly as possible. This means that the application needs to have a small frame drop (the number of frames per seconds is getting lower). For the performance comparison of the models, two tests were made: testing the average latency of the model (how fast it can predict a data samples) and testing the average FPS (record a number of frames, count the seconds and see what the average FPS is). The comparison of those values is presented in Table 6.3. The tests were made on non optimized models, and a prediction is made every 3 frames and no synchronization.

Exercise	Model type	Average latency	Average frame rate
Bicep curl	Shallow	35.76 ms	22 FPS
Bicep curl	Deeper	37.82 ms	21 FPS
Shoulder press	Shallow	35.07 ms	22 FPS
Shoulder press	Deeper	36.44 ms	21 FPS
Squat	Shallow	33.87 ms	21 FPS
Squat	Deeper	36.40 ms	21 FPS

Table 6.3: Performance comparison for the proposed models - repetition status

There is not a big difference in the average latency between the shallow and the deeper models, but even 1 ms can give a smoother experience.

Similar solutions

Keeping in mind the results shown for the proposed models, is time to take a look at similar solutions and make comparison between the solution created in this paper and the ones from the internet.

One similar approach is the solution proposed by Levan Sanadiradze from `https://medium.com/@levansanadiradze/counting-squats-with-a-simple-convolutional-neural-network-5613a9668fa1`. This work provides a way in which a program that can count the number of squats done by a person using a CNN model, which takes advantage of Conv2D layers. It is also based on separating the exercise in more stages (in this case upper, middle and lower) to determine if a repetition should be counted. It is not taking advantage of a pose estimation solution, instead is transforming the image in gray scale and make computation on the obtained transformation. As results, the **training accuracy is 69.48% (ours is**

98.22%) and the validation accuracy is 79.98% (ours is 97.99%). Also the validation loss is 0.43 (ours is 0.11).

Another system that is similar to ours is the one proposed in [SRFS19]. This approach uses a sensor (3D accelerometer provided by Movesense) attached on the chest of an individual and tracks four types of exercises: pushups, situps, **squats** and jumping jacks. The data obtained from the sensor are processed and passed to a CNN, which contains three hidden layers. **The average accuracy for the four types of exercises is 97.9%, while the accuracy for squats is 97.8%**.

6.3 Approach 2: Exercise correctness model

The first type of checker proposed by this paper can be considered an image classification. For checking the correctness of the current movements of an user, multiple frames are needed to make the prediction, so the temporal matter arises. This problem transforms into a video classification.

The description of the proposed architectures can be found in Chapter 4 and will also be described in this subsection.

Training

For training the model the dataset was divided in proportion of 66% training set and 33% validation set, the Adam optimizer, with a learning rate of 0.001, was chosen together with the categorical_entropy loss function from Keras. In Table 6.4 the average fitting time per epoch is presented, together with a description of each model architecture.

Model type	Model architecture	Number of epochs	Average time for an epoch
CNN	One pair of Conv1D (32 filters) and MaxPooling1D layers	100	3 seconds
CNN	Two pairs of Conv1D and MaxPooling1D layers united by a Dense layer	100	6 seconds
LSTM	One LSTM layer with 32 units	50	9 seconds
LSTM	Two LSTM layers with 64 and 32 units	50	20 seconds
CNN + LSTM	A combination between a CNN and a LSTM	100	4 seconds

Table 6.4: Training statistics for the correctness checker

Accuracy and loss

For simplifying the table the models will be called CNN_1, CNN_2, LSTM_1, LSTM_2 and CNN+LSTM (in the same order they were presented in Table 6.4). Table 6.5 contains the accuracy value and the loss for each model. Also, the models were trained using samples of size 10 (10 inputs inside a sequence).

Exercise	Model name	Training Accuracy	Validation accuracy	Training loss	Validation loss
Bicep curl	CNN_1	95.54	95.95	0.12	0.11
Bicep curl	CNN_2	97.69	97.65	0.0459	0.0451
Bicep curl	LSTM_1	98.09	97.90	0.034	0.038
Bicep curl	LSTM_2	97.83	97.73	0.042	0.043
Bicep curl	CNN+LSTM	97.61	97.39	0.04	0.05
Shoulder press	CNN_1	97.07	95.94	0.07	0.10
Shoulder press	CNN_2	99.08	98.61	0.02	0.04
Shoulder press	LSTM_1	99.07	99	0.023	0.26
Shoulder press	LSTM_2	99.41	99.16	0.01	0.02
Shoulder press	CNN+LSTM	98.73	98.83	0.032	0.03
Squats	CNN_1	96.14	96.53	0.10	0.09
Squats	CNN_2	99.29	98.84	0.02	0.03
Squats	LSTM_1	99.02	99.28	0.028	0.022
Squats	LSTM_2	99.55	99.67	0.14	0.10
Squats	CNN+LSTM	98.84	98.03	0.03	0.05

Table 6.5: Accuracy and loss for the correctness checker model

Performance testing

Depending on the approach used, the model can make a prediction each frame or once every few frames. Whatever when the prediction is made, a low latency is good, which results in low frame drops. Table 6.6 presents the average latency and the average frame rates for each model.

The tests were made on models that were not optimized, the ones used for squats, and the average frame rate without the model is around 23 FPS. The average frame rate is computed on predictions made every 3 frames.

Model name	Average latency	Average frame rate
CNN_1	43.23 ms	21 FPS
CNN_2	41.13 ms	22 FPS
LSTM_1	47 ms	22 FPS
LSTM_2	49.27 ms	20 FPS
CNN+LSTM	46.75 ms	23 FPS

Table 6.6: Performance comparison for the proposed models - correctness checker

Similar solutions

Knowing the results provided for the proposed models, we can check other similar solutions and make comparisons.

The second approach presented in the similar solutions for the repetition status checker (solution proposed in [SRFS19], found at page 50) also provides a model for detecting the exercise performed. **The overall accuracy is 90.6% for the validation set and 89.6% for the training set, with a validation loss of 0.206**.

A solution based on inertial sensors is described in [LJLC20]. The inertial sensors are placed on different parts of the user's body (the thighs, the calves and lumbar region) and provides classification for the squat posture. It proposes two classification models: one using random forests and one using CNN+LSTM. They provide result for different number of sensors used on a subject. Using 5 sensors, **the average accuracy for the Random Forest approach is 75.4% and for the CNN+LSTM combo is 91.7%**.

One approach that used the Human Pose estimation to process the frame containing the image is described in [OSSII19]. This solution creates a distance matrix for each frame (computes the distance between every pair of keypoints obtained by the pose estimator) using the Euclidean distance, resulting in additional computations for each frame. After obtaining the matrix, it is flatten and sent to a 1D Convolution. **The accuracy obtained is 75%, which raises to 81.05 using frame smoothing**.

Another similar solution is one used for classifying yoga positions from videos, which is presented in [Kot20]. It uses OpenPose as the Human Pose estimation model and is extracting the 2D coordinates of every keypoint available and are stored in sequences of 45 frames. Different kind of Machine Learning models were used, such as SVM's, CNN's and CNN+LSTM. **The CNN model provides an accuracy of 98.78% and an accuracy of 99.87% for the CNN+LSTM approach**. The model has as dataset videos showcasing 8 types of yoga movements, which are pretty different from each other.

Chapter 7

Conclusions and future works

This thesis aims to solve the problem of preventing the injuries during the execution of a fitness exercise, which can lead to huge pain. One of the solution that is popular in the Deep Learning community is the Human Pose estimation, which is an important milestone in the processing images or videos containing humans.

Using the concept of pose estimation, a method than can help detect mistakes in execution of some fitness exercises can be created, helping a large portion of people which are doing them incorrectly. The solutions proposes in this paper are constructed from scratch, starting from creating even a dataset for the neural network models. This step raised some difficulties, because there are not many examples of wrongly executed exercises.

Providing multiple architecture for the proposed models, the thesis obtains good results regarding the accuracy and the loss of a given model. Different kinds of layers were used during the presentation of the approaches, starting from simple CNN's, using LSTM's and in the end combining those two layers to obtain a single neural network. All the details about the models assembled in this work can be found in Chapter 4, and the results for each individual one of them are presented later, in Chapter 6.

When thinking about future improvements, there are many possible enhancements for both the study presented in this thesis, and also the application. For the study, more possible directions can be considered. One of the main additions the study can bring is the discussion about a neural network model that works on all the proposed exercises, detecting which exercise is performed. Entering in further possible refinements, the study can analyze a model that combines the two models proposed in the paper.

For the application, there are also a lot of possible refinements, some of them being the addition of new exercises for the users of the application and the addition of more functionalities, such as grouping the exercises in workouts. Also, a gym mode can be implemented such that the a camera is integrated in a gym equipment and the practitioners can see the execution on a screen.

All in all, the Artificial Intelligence can be used to create solutions for a big range of domains that do not seem connected to each other, taking advantages of solutions already implemented, such as the Human Pose estimation.

Bibliography

[Art] Artem Oppermann. What is Deep Learning and How does it work? `https://towardsdatascience.com/what-is-deep-learning-and-how-does-it-work-2ce44bb692ac`. Online; accessed 25 May 2021.

[BGR+20] Valentin Bazarevsky, Ivan Grishchenko, Karthik Raveendran, Tyler Zhu, Fan Zhang, and Matthias Grundmann. Blazepose: On-device real-time body pose tracking. *arXiv preprint arXiv:2006.10204*, 2020.

[BK00] Gary Bradski and Adrian Kaehler. Opencv. *Dr. Dobb's journal of software tools*, 3, 2000.

[C+18] Francois Chollet et al. *Deep learning with Python*, volume 361. Manning New York, 2018.

[Chr] Chris Nicholson. A Beginner's Guide to Neural Networks and Deep Learning. `https://wiki.pathmind.com/neural-network#element`. Online; accessed 21 March 2021.

[CHS+19] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Openpose: realtime multi-person 2d pose estimation using part affinity fields. *IEEE transactions on pattern analysis and machine intelligence*, 43(1):172–186, 2019.

[CR17] Ching-Hang Chen and Deva Ramanan. 3d human pose estimation = 2d pose estimation + matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[Dav] David Lange. Physical Activity - Statistics Facts. `https://www.statista.com/topics/1749/physical-activity`. Online; accessed 9 March 2021.

[Dha] Dhanoop Karunakaran. Deep learning series 1: Intro to deep learning. `https://medium.com/intro-to-artificial-`

intelligence/deep-learning-series-1-intro-to-deep-learning-abb1780ee20. Online; accessed 25 May 2021.

[DYWZ19] Qi Dang, Jianqin Yin, Bin Wang, and Wenqing Zheng. Deep learning based 2d human pose estimation: A survey. *Tsinghua Science and Technology*, 24(6):663–676, 2019.

[Eur] EuroStat. Statistics on sport participation. https://ec.europa.eu/eurostat/statistics-explained/index.php/Statistics_on_sport_participation#Active_participation_in_sport. Online; accessed 8 March 2021.

[Gav] Gavril Ognjanovski. Everything you need to know about Neural Networks and Backpropagation — Machine Learning Easy and Fun. https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a. Online; accessed 02 April 2021.

[Gér19] A. Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019.

[GGPO15] Cornelia Győrödi, Robert Győrödi, George Pecherle, and Andrada Olah. A comparative study: Mongodb vs. mysql. In *2015 13th International Conference on Engineering of Modern Electric Systems (EMES)*, pages 1–6. IEEE, 2015.

[Goo] Google. Mediapipe. https://google.github.io/mediapipe. Online, accessed 30 April 2021.

[Gri18] Miguel Grinberg. *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.", 2018.

[GRMHOR14] Marco Garduño-Ramón, Luis Morales-Hernández, and Roque Osornio-Rios. Morphological filters applied to kinect depth images for noise removal as pre-processing stage. 05 2014.

[Hea15] Jeff Heaton. *AIFH, volume 3: deep learning and neural networks*. Heaton Research, 2015.

[Hoi13] Derek Hoiem. How the kinect works. *Lecture notes for Computational Photography*, 2013.

[Hun] Hunter Heidenreich. What are the types of machine learning? `https://towardsdatascience.com/what-are-the-types-of-machine-learning-e2b9e5d1756f/`. Online, accessed 04 April 2021.

[Jas] Jason Chin. Phormatics: Using ai to maximize your workout. `https://github.com/jrobchin/phormatics`. Online, accessed 10 June 2021.

[Kot20] Shruti Kothari. Yoga pose classification using deep learning. 2020.

[Lin] Lindsry Kitchell. Intro to deep learning: Convolutional neural networks. `https://kitchell.github.io/DeepLearningTutorial/4cnnsinkeras.html`. Online, accessed 10 June 2021.

[LJLC20] Jaehyun Lee, Hyosung Joo, Junglyeon Lee, and Youngjoon Chee. Automatic classification of squat posture using inertial sensors: Deep learning approach. *Sensors*, 20(2):361, 2020.

[Luk] Luke Plunkett. Report: Here Are Kinect's Technical Specs. `https://kotaku.com/report-here-are-kinects-technical-specs-5576002`. Online; accessed 02 April 2021.

[LWW$^+$17] Chuankun Li, Pichao Wang, Shuang Wang, Yonghong Hou, and Wanqing Li. Skeleton-based action recognition using lstm and cnn. In *2017 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*, pages 585–590. IEEE, 2017.

[Mac11] John MacCormick. How does the kinect work. *Presentert ved Dickinson College*, 6, 2011.

[Mic] Michael Phi. Illustrated guide to lstm's and gru's: A step by step explanation. `https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21/`. Online, accessed 04 June 2021.

[Mir] Mirel Alexa. Biletul de Sănătate se extinde. `https://www.sportsfestival.com/news/biletul-de-sanatate-se-extinde/`, note = Online; accessed 9 March 2021.

[Nat] National Safety Council (NSC). Facts + Statistics: Sports injuries. `https://www.iii.org/fact-statistic/facts-`

statistics-sports-injuries. Online; accessed 20 March 2021.

[Nic] Nicholas Rizzo. Covid's impact on the fitness industry. https://runrepeat.com/pandemics-impact-fitness-industry. Online, accessed 30 April 2021.

[Nie15] Michael A Nielsen. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, 2015.

[ON15] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[OSSII19] Ryoji Ogata, Edgar Simo-Serra, Satoshi Iizuka, and Hiroshi Ishikawa. Temporal distance matrices for squat classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.

[Pra] Prakhar Ganesh. Human Pose Estimation : Simplified. https://towardsdatascience.com/human-pose-estimation-simplified-6cfd88542ab3. Online; accessed 9 March 2021.

[Pur] Purnasai Gudikandula. Recurrent neural networks and lstm explained. https://purnasaigudikandula.medium.com/recurrent-neural-networks-and-lstm-explained-7f51c7f6bbb9/. Online, accessed 04 June 2021.

[Rob] Rob Kemp. Exercise injuries are booming in lockdown. https://www.telegraph.co.uk/health-fitness/body/exercise-injuries-booming-lockdown/. Online, accessed 30 April 2021.

[SCH16] Yahtyng Sheu, Li-Hui Chen, and Holly Hedegaard. Sports-and recreation-related injury episodes in the united states, 2011-2014. *National health statistics reports*, (99):1–12, 2016.

[Sha] Shanika Perera. An introduction to reinforcement learning. https://shanikaperera11.medium.com/an-introduction-to-reinforcement-learning-1e7825c60bbe. Online, accessed 04 April 2021.

[SRFS19] Kacper Skawinski, Ferran Montraveta Roca, Rainhard Dieter Findling, and Stephan Sigg. Workout type recognition and repetition counting with cnns from 3d acceleration sensed on the chest. In *International Work-Conference on Artificial Neural Networks*, pages 347–359. Springer, 2019.

[Sta] Stanislav Isakov. Inferring a 3d human pose out of a 2d image with fbi. `https://test.neurohive.io/en/state-of-the-art/inferring-a-3d-human-pose-out-of-a-2d-image-with-fbi/`. Online, accessed 10 June 2021.

[TS14] Alexander Toshev and Christian Szegedy. Deeppose: Human pose estimation via deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1653–1660, 2014.

[VSS$^+$19] Ivan Vasilev, Daniel Slater, Gianmario Spacagna, Peter Roelants, and Valentino Zocca. *Python Deep Learning: Exploring deep learning techniques and neural network architectures with PyTorch, Keras, and TensorFlow*. Packt Publishing Ltd, 2019.

[ZWY$^+$20] Ce Zheng, Wenhan Wu, Taojiannan Yang, Sijie Zhu, Chen Chen, Ruixu Liu, Ju Shen, Nasser Kehtarnavaz, and Mubarak Shah. Deep learning-based human pose estimation: A survey. *arXiv preprint arXiv:2012.13392*, 2020.