TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Department of Software Science

Allar Viinamäe, 163578 IAPM

RECOGNITION OF GYMNASTICS ELEMENTS WITH DEEP LEARNING NEURAL NETWORKS

Master's thesis

Supervisors: Sven Nõmm, PhD

Tallinn 2020

TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Allar Viinamäe, 163578 IAPM

VÕIMLEMISOSKUSTE TUVASTUS SÜGAVATE TEHISNÄRVIVÕRKUDEGA

Magistritöö

Juhendajad: Sven Nõmm, PhD

Tallinn 2020

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Allar Viinamäe

01.08.2020

Recognition of gymnastics elements with deep learning networks

Abstract

The present thesis's primary goal is to construct a dataset consisting of gymnastics activities - the backflip and the back handspring, use pose estimation, implement pre-processing strategies, and develop machine learning models capable of distinguishing between the two gymnastics activities.

Backflips and back handsprings are two of the foundational gymnastics moves performed from a standing position and including the rotation of the human body, making it challenging for machine learning algorithms designed to recognize more straightforward human actions to recognize these more complex moves.

The majority of solutions available for estimating poses in gymnastics use movement-restricting and invasive motion capture tools. Other forms of estimating poses from a distance include multi-camera setups, which are not usable in daily gymnasium environments. This thesis aims to explore and propose a more accessible alternative to non-invasive gymnastics skill recognition solutions. The solution presented in this thesis uses a consumer-grade video recorder with a single-camera pose estimation to extract time-series skeleton data. After successfully extracting skeleton data and applying data processing strategies, machine learning classifiers are developed using recurrent neural networks to train the machine to make successful distinctions between backflips and back handsprings.

Current research's primary outcome is classifier models, capable of differentiating backflip and back handspring activities, providing prediction performance up to 95%. Discussion and classifier analysis section of this thesis gives more in-depth insights into the artificial neuron activations by visualization.

The present thesis is written in English and is 65 pages long, including 7 chapters, 1 table and 25 figures.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Gymnastics, a sports discipline with an extensive history, requires athletes to have a comprehensive set of physical traits to execute exercises. These traits include balance, strength, flexibility, agility, and endurance [1]. By developing these physical attributes, the athletes can perform motions involving aerial twisting and rotations. This master's thesis focuses on what is known as *tumbling*, a sub-discipline of gymnastics. Some of the foundation movements in tumbling, ofter categorized as flips and handsprings, include backflips (aka. a back tuck) and back handsprings. These movements form the backbone of tumbling as they are needed to progress onto more advanced moves, while also used as individual components of a tumbling routine.

As backflips and back handsprings are the foundational movements for more advanced exercises, athletes train them regularly. Athlete's perfect their technique until these movements become almost intuitive for a gymnastics athlete. Seasoned athletes experience these exercises' intriguing physical properties, such as the impulse, inertia, rotation, and optimal takeoff angle, without consciously thinking about them. For example, kinematic analysis of the backflip in a tumbling series was conducted on beginner and advanced set of athletes to differentiate their techniques [2]. The differences in technique between beginner and advanced athletes contribute to both competition score and injury risk, so optimal technique can be considered a priority for every athlete.

At the time of writing, the most common methods for analyzing gymnastics movements require sophisticated motion capture tools, physically attached sensor data, and gymnastics coaches' attention to give valuable feedback to the athletes or analyze their technique [3] [4]. This master's thesis focuses on the automatic recognition of backflips and back handsprings using a combination of pose estimation tools with recurrent neural networks. The author proposes the single-camera solution presented in this thesis as a non-invasive, accessible, and state-of-the-art solution to recognize backflips and back handsprings. While not part of this thesis, the author also hopes to contribute the solution developed in this thesis to future research in automated analysis of technique, feedback for athletes, and automatic documentation of gymnastics workouts' history.

1.2 Human Activity Recognition Background

Human activity recognition taxonomy can be challenging as the diversity of available methods is extensive. We depict the broad categorization of human activity recognition methods in figure 1.1, proposed in the article [5]. We also categorize the technique used to automate gymnastics activity recognition in this paper as a *unimodal shape-based method.* While identifying gymnastics movements from multiple modalities (i.e., the addition of behavioral or emotional features) could provide useful data for the analysis, it would require more invasive tools, such as microphones and physical sensors. The author aims to develop a recognition method less invasive and free of physical attachments. For example, in competition environments, athletes are not allowed to wear any extra accessories [6], and the only viable method for recording the activity is the video modality.

Narrowing down on unimodal methods and considering the non-invasiveness of shape-based methods, brings the author to a technique called *pose estimation*, a popular research topic in the last ten years. More recently, computationally effective 2D pose estimation methods in real-time using Part Affinity Fields have been proposed [7]. This method is also used by the OpenPose software [8], which is the choice of pose estimation software in this research.

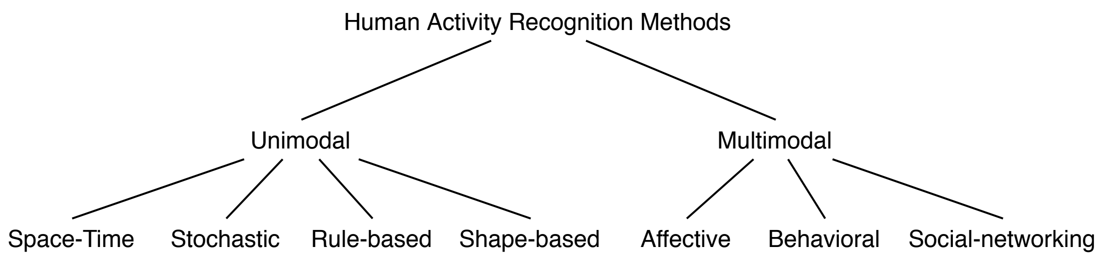

Figure 1.1: Hierarchical categorization of human activity recognition methods

1.3 Problem statement

The following list depicts the sub-problems of the problem statement:

- *Acquiring and exploring relevant data* — This thesis examines the data needed for capturing complex biomechanical movements without being invasive towards the athletes. From the author's own experience in practicing gymnastics, many already developed restricting motion capture solutions do not apply to the everyday sports environment.

  Recording human actions can be broadly classified into two categories: (a) **optical methods** — for example the *marker-based* systems used in entertainment, but also newer *marker-less* and *deep learning* based systems [9] and (b) *non-optical methods* — both single, such as accelerometers or gyroscopes and also complex unit sensing devices, such as inertial/magnetic measurements units [10]. Here the author narrows the solutions down to optical methods, excluding non-optical methods regarding the context of gymnastics. For gymnastics, the freedom of movement is crucial, since many of the foundational movements (i.e., round-offs - the movement athletes use to transform horizontal velocity to vertical velocity) require multiple meters of free space and no restricting devices on the body.

  These restrictions rule out wired, movement-restricting, and additional weight requiring non-optical sensors. Furthermore, we also rule out special equipment requiring optical marker-based systems. For example, in the article [2], research was conducted for comparing the technique of beginner and advanced athletes. While a fascinating study, the markers were attached only to one side of the athlete's body, ignoring the discrepancies of body sides and complicating the experiment's repeatability for everyday use. The author initiates another requirement in the current paper, assuming that the gymnastics athletes are not interested in wearing special equipment for the daily training sessions.

  The author uses a consumer-grade action camera to capture individual athletes' backflips and back handsprings in regular gymnastics facilities, explained in more detail in section 3.1.

- *Pose estimation as a viable option for gymnastics movements* — The author experiments and develops strategies for acquiring, processing, and augmenting data obtained by newer *pose estimation* software (*OpenPose*). We construct a generalized and reusable dataset for experimenting and developing classifiers capable of differentiating gymnastics movements. Further explanation on the pose estimation process is available in section 3.2.3.

- *Human action recognizing classifier development* — Advances in deep learning accessibility inspire the author to explore human action recognition in the context of gymnastics. We use a previously constructed dataset in conjunction with deep learning network design to develop classifiers capable of differentiating between gymnastics movements. The author also adds a cloud-based training environment description to give details of a fully prototyped end-to-end solution, section 5.
- *Visualizing classifier decisions* — The need for an interactive, easily usable, and understandable software for coaches and athletes to analyze their training sessions is expressed in the article [11], emphasizing the ease of usability and allowance of individual input. While giving athletes the feedback for improving their technique is not in the scope of this thesis, the author still makes an effort to visualize and interpret the decisions of the recognitions made by deep learning classifiers in section 5.2.

Chapter 2

Implementation

2.1 Implementation overview

This thesis aims to propose an end-to-end type solution to automate the recognition of gymnastics exercises using computers. High overview of the steps required to achieve this is specified in the following list:

1. Data acquisition
   (a) Human activity recording
   (b) Estimating and extracting keypoints
2. Data exploration
   (a) Pose estimation limitations
3. Data pre-processing
   (a) Applying pre-processing strategies
4. Classifier Training
   (a) Methodology overview
   (b) Classifier training and validation
   (c) Analysis of trained classifiers
5. Prediction explanations for obtained classifiers
   (a) Visualizing neuron activations
   (b) Discussion

2.2 Infrastructure and tools for analysis

2.2.1 Used hardware and software

There are two main modules in this thesis that require the usage of hardware optimized for the computation of multiple parallel processes. Graphics processing unit (GPU) is used for both pose estimation (section 3.2.1) and training classifiers (section 5.1.4).

A single Amazon EC2 instance with the following specifications is launched as the environment for experimentation:

- *Operating system* — Ubuntu Server 18.04 LTS (Hardware-assisted virtualization - HVM).
- *Instance type* — g4dn.xlarge (- 4 vCPUs, 2.5 GHz, Intel Xeon P-8259L).
- *Storage* — 64GB of general purpose SSD volume type. No specific requirements for storage. The amount was chosen primarily to accommodate the augmented dataset used for training.
- *GPU information* — NVIDIA T4 Tensor Core GPU [12] with 16GB of GDDR6 memory is installed on the g4dn.xlarge Amazon instance. Additionally, NVIDIA 450 Linux driver series with CUDA 10.1 was installed for accessing the GPU unit by Tensorflow.

2.2.2 Data-processing environment

All experiments were conducted in the Jupyter Notebook environment (running as a server on the backend instance). The author created separate notebook files for each classifier training, activation analysis, and data exploration. We used Python programming language for developing most of the data processing, classifier training, and validation scripts. Keras, the Python deep learning API, was used with Tensorflow backend for accessing machine learning algorithms. More notable libraries used for experimentation include commonly used *pandas*, *numpy*, *sklearn* and *matplotlib*.

Additional miscellaneous software

- *FTP server and client* — FTP server helps with the uploading and downloading of larger datasets. The *vsftpd* package was used in this thesis.

Chapter 3

Data acquisition

To combine gymnastics activity recognition and computer vision, this thesis's data acquisition phase begins by collecting the recordings of athletes performing gymnastic activities. The activities chosen are the backflip (section 3.1.3) and the back handspring (section 3.1.4). The motivation behind the chosen activities is that these activities are some of the foundation exercises that athletes use as building blocks for more difficult combinations. These activities are particularly interesting for the author of this thesis since the author uses them daily to polish the technique and progress onto more advanced exercises. The recordings are collected by a consumer-grade action camera (section 3.1.5). After recording the activities, human skeleton data will be extracted from the videos using computer vision's technique called *pose estimation*, an important step to transform the data from video format to a data format more suitable for training machine learning models to recognize the activities.

The data to be explored and used to train machine learning models in this thesis plays central part in our implementation. Although, machine learning promises to loosen the strictness of the data used in a system by trying to generalize and adapt the models themselves to the data, the quality of the initial data used to develop prototypes still directly influences the interpretation and value of the outcome. Interpretation of not only the outcome, but the entire solution is needed to have clear understanding of why certain outcomes exist, which in turn helps to spark discussion and spread knowledge gained during an experiment. The value of this research, reusable by peers for new scientific experiments, is also directly influenced by the quality of the initial data.

The data acquisition chapter is split into two parts. The first part (3.1) explains what kind and how much of data the author is collecting and the process behind it. The second part (3.2) explains a more technical approach on how the initial data for the human activity recognition algorithm is acquired from recorded human motions and the tools used to do so.

3.1 Recording gymnastics activities

3.1.1 Choosing Activities To Record

When it comes to choosing challenging biomechanical activities performed by humans, gymnastics is one on the top of the list. Floor gymnastics exercises, in particular, don't require any additional equipment by humans. Still, at the same time, physical strength, flexibility, and kinesthetic awareness are a must to perform any of the skills presented by elite athletes. The athlete's physical requirements are evaluated by versatile fitness level evaluation tools developed for gymnastics [13]. Gymnastics is also unique in its indisputable need for utilizing every part of the human body. All major muscle groups of an athlete need to be in superior condition to perform specific rotations, jumps, and holds. As age is a limiting factor in peak physical condition, many athletes start their careers at a very young age [14]. However, there is no age limit for practicing gymnastics at a recreational level.

Working with gymnastic movements and trying to automate the recognition of this kind of human activity requires a reference or in machine learning terms *training data* to train our machine. The two activities chosen to prove the hypothesis of this paper are known as backflips and back handsprings. While fast and explosive, these activities are similar in their execution and achievable by all levels of gymnastics trainees. As backflips and back handsprings are the building blocks of more complex gymnastic combinations, perfecting the technique of these activities provides athletes with the confidence and skill necessary to move on to more difficult feats. From the need for perfecting the technique of these basic activities comes the motivation to analyze them and automate the feedback loop for the athletes. From the athlete's perspective, it is faster and cheaper to get the feedback from, for example, a portable device with recording capabilities, rather than a gymnastics coach. From the coach's perspective, it is also more convenient to automate the learning process of easier activities so the coach can concentrate on guiding athletes towards more interesting and challenging exercises. Figures 3.1 and 3.2 will provide the reader a visual idea of how backflips and back handsprings are performed respectively.

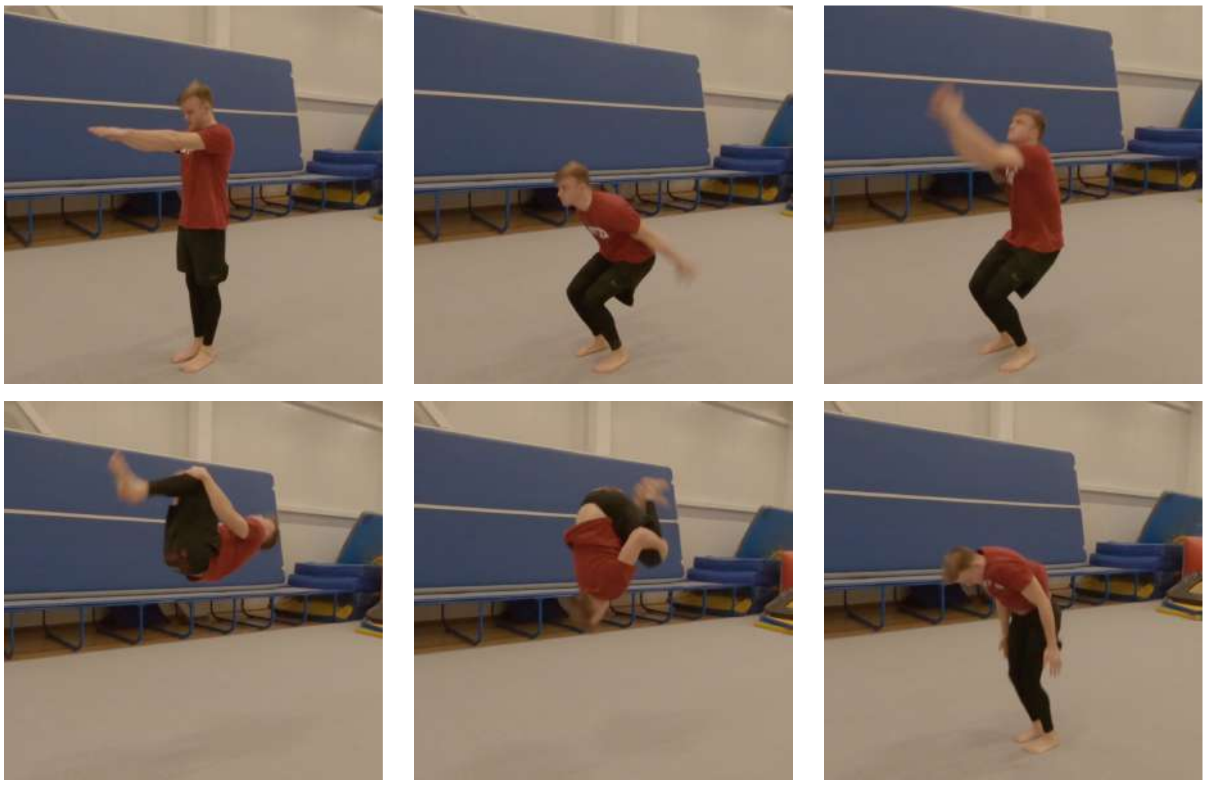

Figure 3.1: Example of a backflip

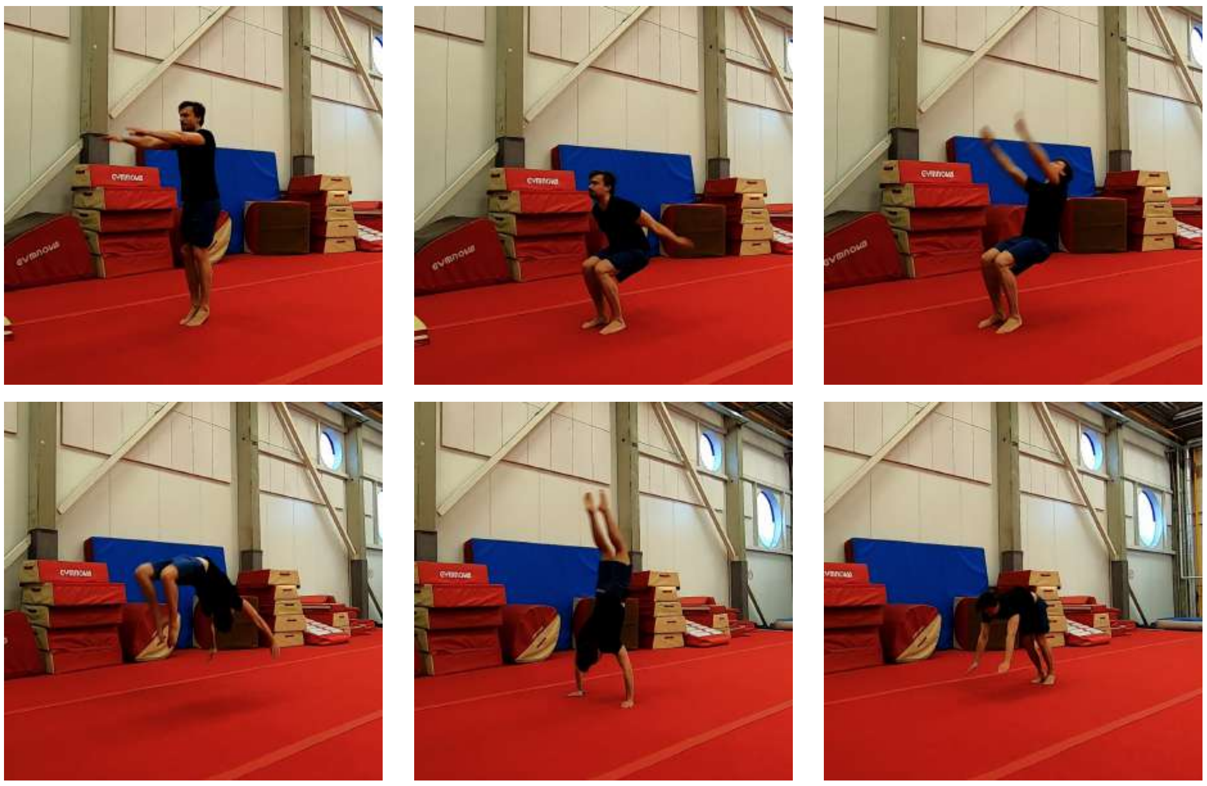

Figure 3.2: Example of a back handspring

3.1.2 Extracting gymnastics activities

The start and end of each activity under investigation needs to be defined. The author needs to trim the recorded gymnastics activities to include only each activity's full duration, removing the recording parts where the athlete is not performing the activities under investigation. This paper proposes ”start” and ”end” markers for both activities to be easily recognized and validated by a human observer. The markers should cover the full duration of a movement, all while sustaining the integrity and focusing on each activity's period of interest.

More detailed explanations of each activity and references to the visual markers are in sections 3.1.3 and 3.1.4.

3.1.3 Backflip

A backflip is a sequence of body movements in which a person leaps into the air and rotates backwards over the body's horizontal axis. For the backflip, we mark the start of a backflip as the frame when athlete's both arms pass the horizontal line at shoulder level moving downwards and generating momentum. We mark the end of a backflip as the frame when both heels of the feet touch the ground again. We choose the heels of feet so we can include the amortization part of the landing phase of the backflip in the recording. The red bars in figure 3.3 demonstrate the visual markers for trimming the sample recording to include only the activity under investigation. These markers were chosen by the author to the best of his knowledge of the domain as the clearest points for a human observer to recognize a backflip. Choosing different markers is a potential discussion topic for future improvements.

3.1.4 Back Handspring

A back handspring is similar to a backflip in that the athlete also rotates his body around the horizontal axis. However, during a back handspring, the athlete also moves backwards, while during a backflip the athlete should ideally not move in any direction. The other clear distinction between a back handspring and backflip is that during a back handspring, the athlete's arms extend and push off the floor to create the spring part of the activity and keep the athlete moving backwards.

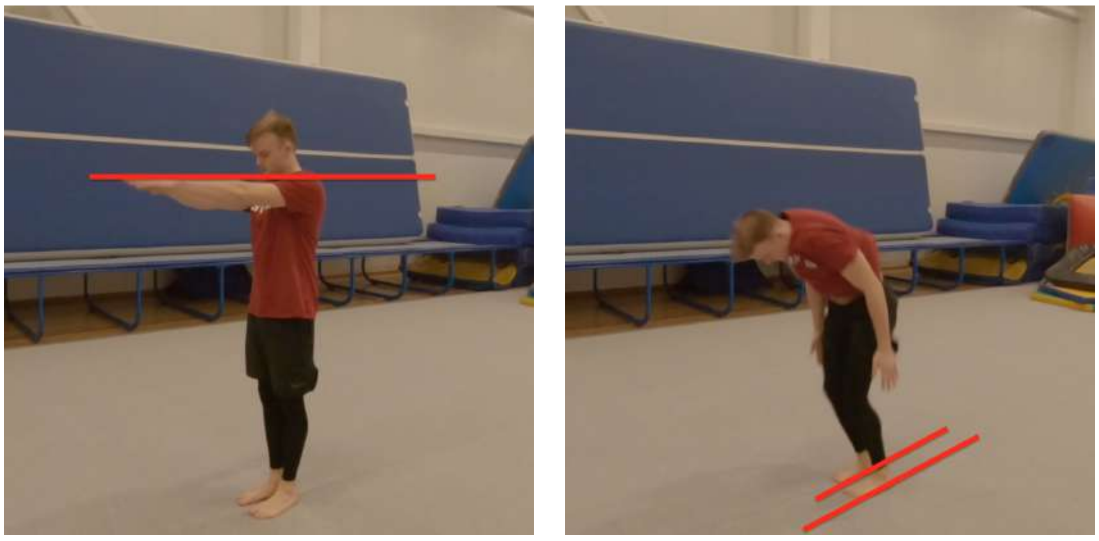

Figure 3.3: Start and end markers for covering the full duration of a backflip

Thirdly, the takeoff angle differs for both back handspring and backflip. The takeoff angle decides in which direction the athlete moves during the rotation. For the backflip investigated in this paper, the goal of the athletes was to keep the takeoff angle as perpendicular as possible to the floor to keep the athlete from moving either backwards or forwards during the rotation. However, for the back handspring, the takeoff angle should be around 45° backwards to the floor, to help the athlete move backwards and land on the hands. The green bars in figure 3.4 demonstrate the visual markers for covering the full duration of a back handspring. The color of markers are in both cases irrelevant and are chosen primarily for better visual distinction.

3.1.5 The Recording Process and Results

Static images of the backflips and back handsprings are not sufficient as we have defined both of these activities as sequences of body movements (in sections 3.1.3 and 3.1.4), therefore the author records backflips and back handsprings in the video format. The device used to capture this paper's activities is a *GoPro Hero7 Black*, with the following basic settings:

- *RES (resolution)* — 1080p
- *FPS (frame rate)* — 60
- *FOV (field of view)* — Linear

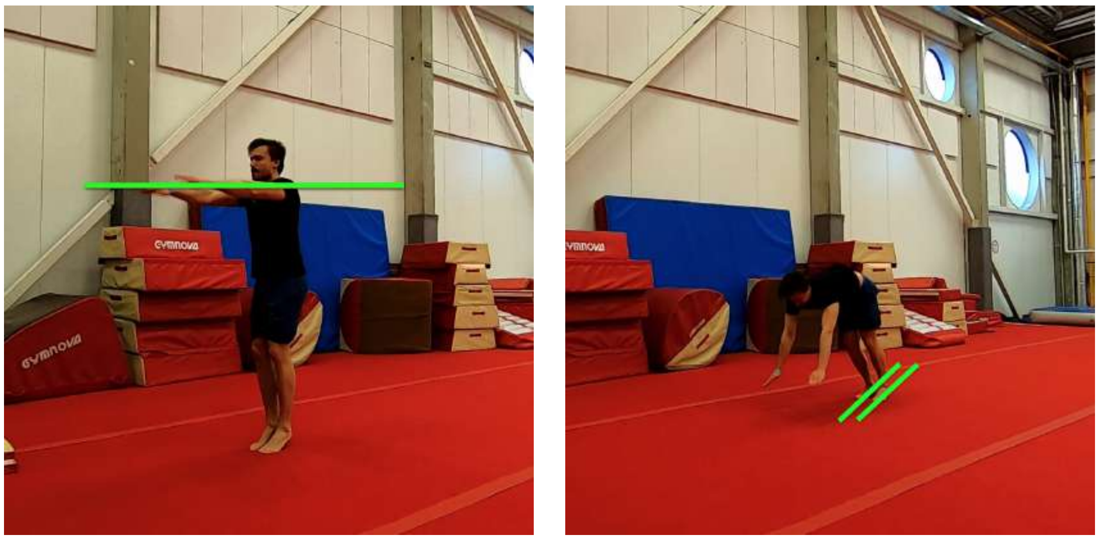

Figure 3.4: Start and end markers for covering the full duration of a back handspring

- *Low Light* — Auto
- *Stabilization* — Auto
- *Protune* — Off

Each activity is recorded as one still view from eye level fully showing the subjects body from head to toe. The *three-quarter view* (*two-thirds view*) is chosen to capture each activity. Choosing this angle avoids the limbs of the subject stacking behind each other in the recording. Similar works in this domain use either sophisticated three-dimensional motion capture systems using six cameras with depth sensors [15] or older works using bigger cameras and recording perpendicular to the plane of the movement [2]. While very thorough analyzes, taking into account the center of body mass or exact takeoff and landing angles, they are harder to reuse in a wider variety of difficult biomechanical movements. Mentioned works often require athletes to wear special gear (markers) and also require longer setup times for the recording cameras [16]. The author of this work tries to emphasize on the potential for improving *non-invasive field-based methods* to quantify and monitor technical biomechanical movements. In order to improve these methods, the hypothesis of this work is to use a single consumer camera setup accompanied with a pose estimation solution to achieve satisfactory results for recognizing technical gymnastic activities.

The results of the recording process amounts to a total of 96 backflips and 84 back handsprings captured.

3.2 Estimating and extracting gymnastics activities

The second section of this chapter focuses on the process of extracting human skeleton data from raw video files captured in the first section of this chapter. The skeleton data will be later preprocessed and used to train machine learning algorithms to recognize gymnastic activities.

3.2.1 Pose estimation

The author's aim is to approach the problem of extracting skeleton data from raw videos using the most state-of-the-art methods. In recent years, some scientific papers have been released using machine learning based methods for estimating the human pose in images [17]. For example, in 2015 an article was published, where convolutional neural networks were used to estimate human poses in videos [18]. These new methods exceed previous solutions in terms of prediction accuracy.

In recent years, even more complete solutions have been proposed, enabling to estimate human poses in real time thanks to their computational efficiency. For example, a method called *Part Affinity Fields*, essentially a set of vectors encoding the direction of a body part, has been proposed. Every limb is described using an *affinity field* between body parts [7]. Such method is used by, for example, an open source solution called *OpenPose* [8]. OpenPose allows to estimate the skeletons of multiple humans from a video recorded with a monocamera. This enables the athlete's skeleton to be identified in an environment with bystanders in the background (a common sight in gymnastic centers).

The author of this work hopes to build a practically usable prototype based on previous work in the pose estimation field. The author uses the free software OpenPose to estimate poses during gymnastic movements. Alternatives to OpenPose performing pose estimation include *PoseNet* [19] and *wrnchAI* [20].

3.2.2 OpenPose's infrastructure and requirements

The mandatory requirement for using OpenPose's pose estimation in a reasonable time is a dedicated GPU unit and access to its general-purpose computing API [21]. The two main API's supported are CUDA for Nvidia GPU and OpenCL for an AMD GPU. Other tools for using OpenPose include CMake to compile the OpenPose software and Python programming environment to access OpenPose's API.

A CPU-only setup of OpenPose is also supported but highly advised by the author to be used only for testing purposes. For initial software inspection purposes, the author compiled OpenPose on a 2015 MacBook Pro with a *2,7 GHz Dual-Core Intel Core i5* CPU. The time required for pose estimation in a sample recording averaged to around 30 seconds for a single frame. Since using this setup for pose estimation for 180 samples (recorded at 60 FPS) would result in an unreasonable time spent on this process, the author decided to use an Amazon GPU instance instead.

After compiling OpenPose on the Amazon instance, an average pose estimation time of 0.08 seconds was achieved for a single frame in sample recordings.

3.2.3 The Pose Estimation Process and Results

The author developed a separate Python module for extracting body key points [22]. The module uses the Python wrapper of *OpenCV* (open-source computer vision library) to access each frame of each sample video. It then uses the Python wrapper of OpenPose on each frame to estimate poses. Finally, the module dumps extracted data to separate comma-separated files.

For each sample recording a separate directory is also created. Each directory contains csv files with the filename pattern *activityPerformed-sampleIndex-subjectName.mov-frameIndex-personIndex.* A total of 19119 files were generated from 180 sample recordings. The figure 3.5 shows an example output dumped into a csv file by the author's written Python OpenPose extraction module. The X and Y columns mark the corresponding coordinates and the *Confidence score* column represents the confidence factor by OpenPose when estimating body parts. Finally, the I column represents the body part index estimated by OpenPose. These indexes match the body part model *BODY_25* shown in figure 3.6.

The extraction module automatically ran for every sample and generated a total of 19119 csv files. To give it some context, 19119 csv files amount to 5 minutes and 19 seconds of backflips and back handsprings recorded. Using the Amazon GPU instance, the total duration for processing all samples took less than 30 minutes. An estimate could be made for running the same process on a 2015 MacBook Pro with CPU-only setup. The process would take around 6 days and 15 hours if estimated on the basis that the average processing time of each frame is 30 seconds. In conclusion, running the process on an Amazon GPU instance is vaguely more than 300 times faster. This concludes this chapter on data acquisition. The next chapter explores how this data is explored and preprocessed for machine learning training.

I	X	Y	Confidence score
1	1.092518798828125000e+03	4.508595275878906250e+02	8.697314858436584473e-01
2	1.095280273437500000e+03	5.039057617187500000e+02	9.449880123138427734e-01
3	1.063033203125000000e+03	5.036833190917968750e+02	8.230457901954650879e-01
4	1.015858398437500000e+03	4.213815002441406250e+02	8.257341980934143066e-01
5	9.983654174804687500e+02	3.419716186523437500e+02	8.841910362243652344e-01
6	1.124838012695312500e+03	5.068528442382812500e+02	8.021396994590759277e-01
7	1.121924560546875000e+03	4.126282043457031250e+02	8.438682556152343750e-01
8	1.092484497070312500e+03	3.214095458984375000e+02	8.586141467094421387e-01
9	1.074806030273437500e+03	6.920960693359375000e+02	8.156118392944335938e-01
10	1.051205078125000000e+03	6.891304931640625000e+02	7.759792208671569824e-01
11	9.834980468750000000e+02	7.981050415039062500e+02	8.500082492828369141e-01
12	1.045180419921875000e+03	9.069448242187500000e+02	9.030723571777343750e-01
13	1.098396118164062500e+03	6.950711059570312500e+02	7.734714150428771973e-01
14	1.024742309570312500e+03	8.127473754882812500e+02	9.561880826950073242e-01
15	1.092337402343750000e+03	9.305974731445312500e+02	9.035368561744689941e-01
16	1.092454467773437500e+03	4.478778686523437500e+02	2.429898679256439209e-01
17	1.107154418945312500e+03	4.420529479980468750e+02	7.689275145530700684e-01
18	0.000000000000000000e+00	0.000000000000000000e+00	0.000000000000000000e+00
19	1.118930419921875000e+03	4.627049255371093750e+02	5.308289527893066406e-01
20	1.048442871093750000e+03	9.688068847656250000e+02	8.219341039657592773e-01
21	1.068972167968750000e+03	9.716963500976562500e+02	8.162919282913208008e-01
22	1.101324096679687500e+03	9.423051757812500000e+02	7.726828455924987793e-01
23	1.001352600097656250e+03	9.394185791015625000e+02	7.975240349769592285e-01
24	1.004130065917968750e+03	9.305712280273437500e+02	7.877947092056274414e-01
25	1.048436645507812500e+03	9.129010009765625000e+02	7.727123498916625977e-01

Figure 3.5: Sample body parts estimated by OpenPose for one frame

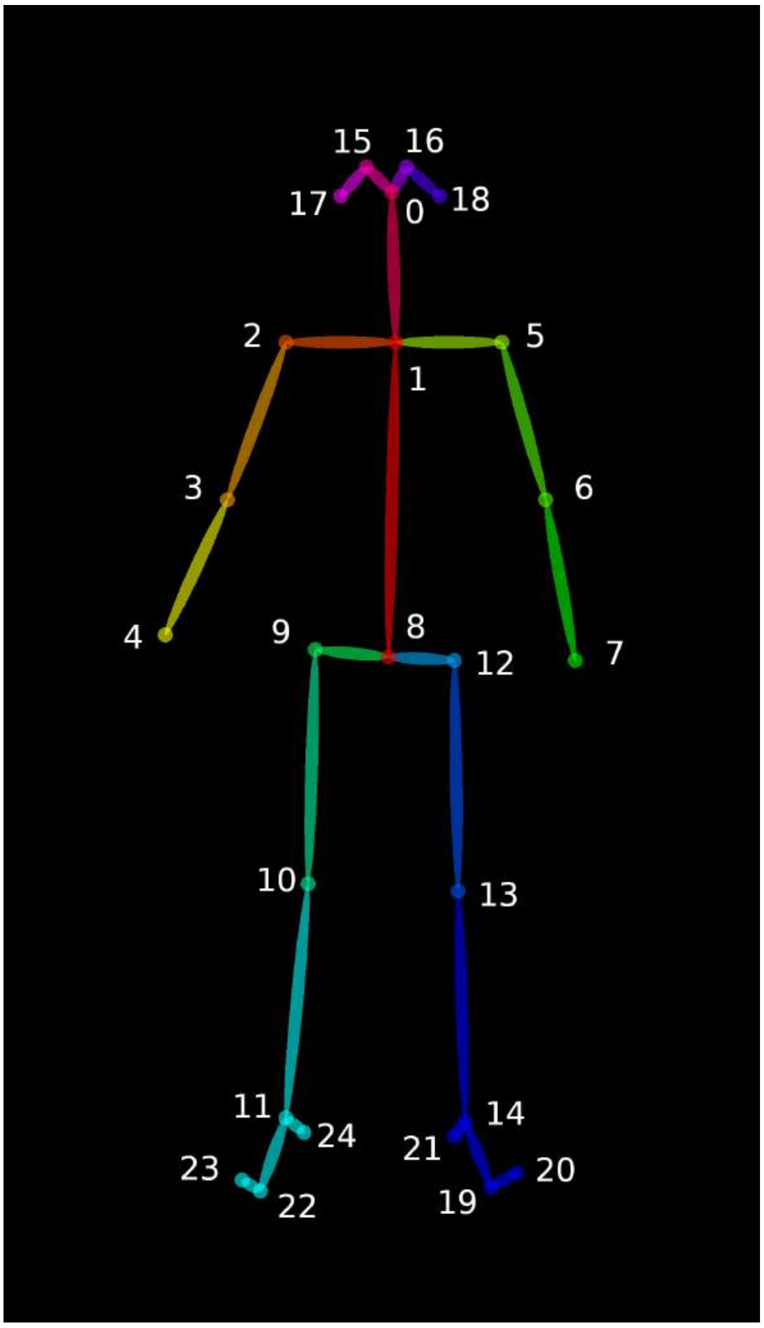

Figure 3.6: Pose Output Format — "BODY_25"

Chapter 4

Data pre-processing

An essential step between acquiring the data and passing it to machine learning algorithms is to ensure that the dataset used for training represents our learning objectives. Our learning objective is to train the system to recognize two of the well-known gymnastics movements - the backflip and the back handspring. A well-balanced and interpretable dataset is necessary for avoiding overfitting or biasing the machine learning model to a specific dataset.

In section 4.1, we start by sampling the data and visualizing it to understand better the pose estimation results obtained in the previous chapter. Then, based on the observations, pre-processing strategies for overcoming the shortcomings of the dataset are proposed by the author in section 4.2. Lastly, the final dataset, ready to be used for machine learning, is described in the 4.2.6 section of this chapter.

4.1 Data exploration

During the data acquisition phase, individual csv files (figure 3.5) were generated for each frame of each sample recording. Since the data is generated for each frame, one suitable visualization method is the time-series line plot. Each frame's data contains 25 estimated keypoints, both X and Y coordinates for each keypoint and also the Confidence Score issued by OpenPose. Given the amount of dimensions for such dataset, a sample entity with the following parameters is chosen for demonstration purposes:

- *Sample activity* — Back handspring
- *Sample no.* — 17
- *Keypoint index* — 21 (left heel), figure 3.6
- *Axis* — Y

There is no particular reason for choosing the left heel of the subject, other than given the rotation of the subject's body during the sample activity, we can expect the data range of the left heel's position to differ vastly on the Y axis. This is because the subject's body will be upside down at some point during the movement.

The figure 4.1 represents the subject's left heel's trajectory along the Y axis during the back handspring. For the first second of the activity, the pose estimation seems not to have problems recognizing the left heel during the momentum generation phase. For the takeoff, rotation, and landing phase, the pose estimation's confidence score falls under the threshold and results in filling the low confidence frames with the zero values, making the data not usable by machine learning algorithms. Such strong deviations can be labeled as anomalies. They will most likely strongly affect the gradient computed during the backpropagation, ultimately making the algorithms learn something else besides the activities investigated.

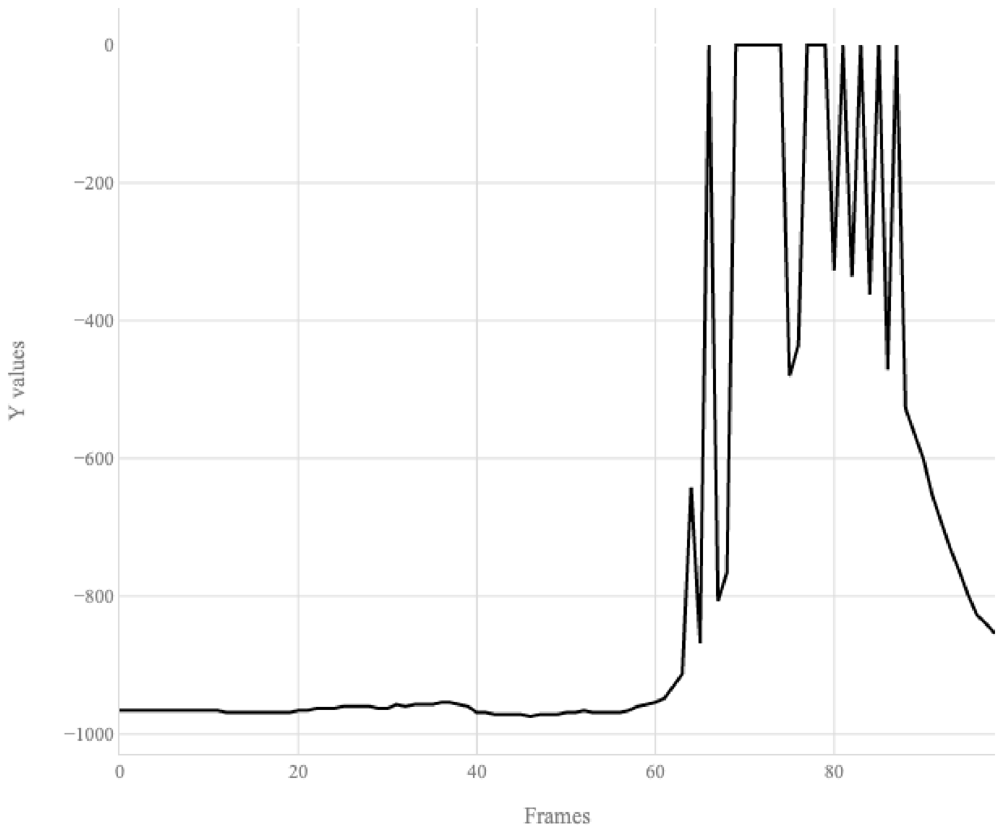

Figure 4.1: Subject's left heel trajectory during the back handspring - raw data

4.2 Data pre-processing strategies

Four mitigation steps are chosen to overcome the shortcomings of initial pose estimation results. It is worth mentioning that for clear interpretation purposes, the strategies described in the next sections are demonstrated on a single sample. Python scripts were developed by the author to automatically apply all pre-processing strategies on every sample's every body part for all frames.

4.2.1 Dealing with low confidence keypoints

During the pose estimation process described in Chapter 3, a confidence score in the range of 0-1 is issued for each keypoint by OpenPose. The confidence score represents OpenPose's certainty when determining keypoints of the subject. A confidence score of 0 by OpenPose means the system fails to estimate a particular body part in a frame. For a keypoint with confidence score approaching 1, translates to high confidence in detecting the specific keypoint.

A simple algorithm was implemented by the author to fill the missing keypoints - moving through frames and filling missing keypoints with the averages of existing keypoint with a positive confidence score, figure 4.2. The improved sample's subject left heel trajectory can be seen in figure 4.3.

4.2.2 Moving average smoothing

During the pose estimation process described in section 3.2.3, the OpenCV library is used to access each frame of the recordings. The frame's data is then fed into OpenPose's estimation function, and estimated keypoints are obtained for each frame. The process is stateless by design, so nor previous or next estimations are taken into account when estimating the current frame, resulting in fine-grained variations between frames. *Moving Average Smoothing* is a technique applied to time series data to remove noise and better expose the signal of the underlying process.

We define the moving average in the equation 4.1. More specifically, we categorize the moving average equation defined as the *centered moving average*. We use the

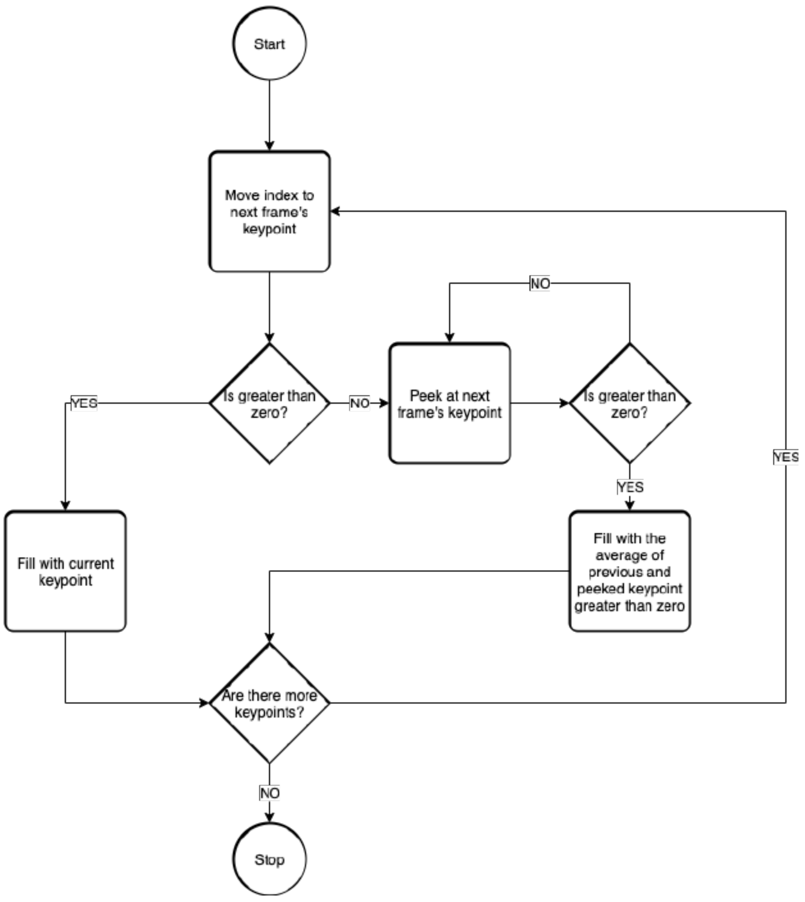
Start
Move index to next frame's keypoint
Is greater than zero?
YES
NO
Peek at next frame's keypoint
Is greater than zero?
NO
YES
Fill with current keypoint
Fill with the average of previous and peeked keypoint greater than zero
Are there more keypoints?
YES
NO
Stop

Figure 4.2: Moving average filling algorithm

centered version of the moving average since all values in the set are known before the smoothing phase. In the formula, we use n of 3 and define it as the width of the moving window. The x defined in the formula is the set of values for each body part coordinate for every frame of the activity. Figure 4.4 displays the transformed left heel's trajectory after applying the smoothing technique.

$$x(t) = \frac{1}{n} \sum_{i=-\left\lfloor \frac{n}{2} \right\rfloor}^{\left\lfloor \frac{n}{2} \right\rfloor} x_{t+i} \tag{4.1}$$

4.2.3 Unrecognizable body parts

One limitation worth mentioning when using the pose estimation technique is the contrast between the background and the athlete in the recordings. In low light and low contrast environments, the body parts of an athlete are not easily recognizable

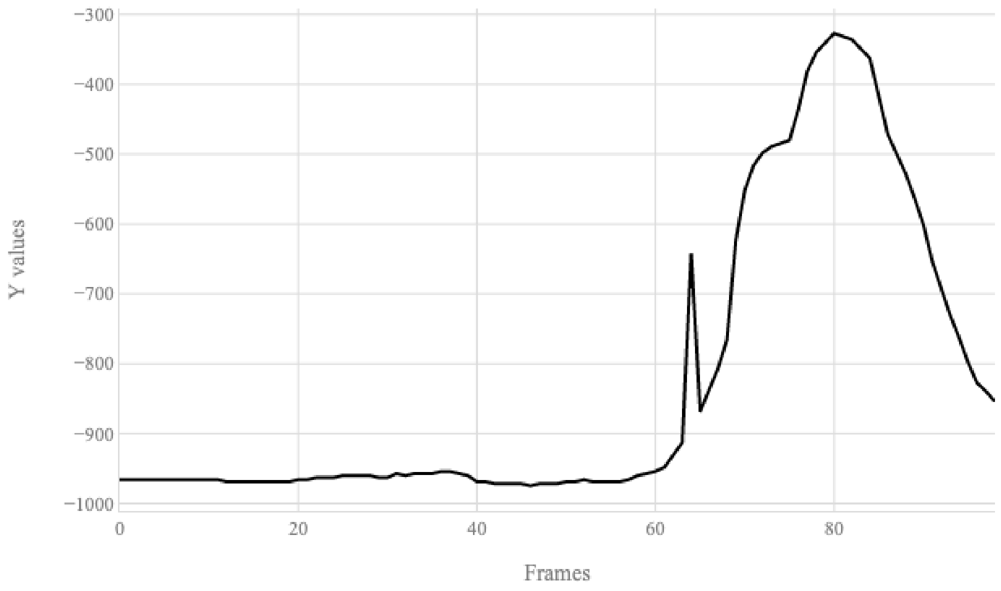

Figure 4.3: Subject's left heel trajectory during the back handspring - filled

(as demonstrated in figure 4.5), which leads to the low confidence score for some body parts and makes it impossible to construct a full skeleton. Possible solutions (not explored in this thesis), to improve the recognition of body parts in low contrast environments, include controlling the contrast of the recordings with some post-production software or rerecording the activities with athletes wearing clothes with a higher contrast against the background. For this thesis, however, the samples with unrecognizable body parts were left out of the dataset during the preprocessing phase.

4.2.4 Centering to unified coordinate origin

Inspired by pre-processing data methods in [23], the authors Yong Du, Wei Wang, and Liang Wang remarkably point out that human actions are independent of its absolute spatial position. Since the starting coordinates of gymnastics movements were not defined before recording the activities, normalizing the samples to a unified coordinate origin significantly decreases fluctuations between the same features. This paper's samples are normalized to the coordinate system origin using the middle hip keypoint indexed as 8 in the *BODY_25* model. Figure 4.6 demonstrates how

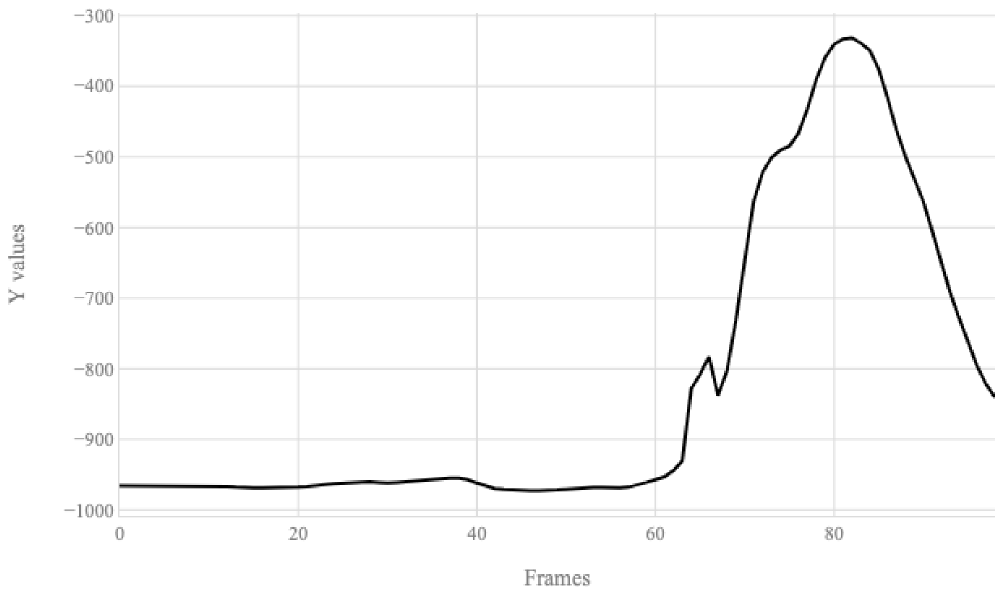

Figure 4.4: Subject's left heel trajectory during the back handspring - smoothed

the skeleton moves relative to the coordinate system origin after normalization.

4.2.5 Data augmentation

One is required to supply well-balanced and a greater amount of data to train classifiers and validate for generalization. Data augmentation techniques exploit domain knowledge to increase the number of training examples and improve generalization without reducing the effective capacity [24]. The result of 113 samples obtained through previous data processing strategies is enough for building network architectures. However, the amount is not enough to validate the model for overfitting. There are many augmentation strategies applicable to skeleton manipulation [25].

The strategy chosen in this case applies random displacement degrees between the local joints of the skeleton. Intuitively, pairs of directly connected keypoints are found, after which a pseudo-random degree between 0 and 15 is generated for each pair. We then apply rotation for the pairs of joints for every frame in the sample. Finally, this augmentation is repeated ten times, amplifying the previous 113 samples to a total of 1130 new augmented samples. Figure 4.7 demonstrates one original backflip sample with two augmented variations.

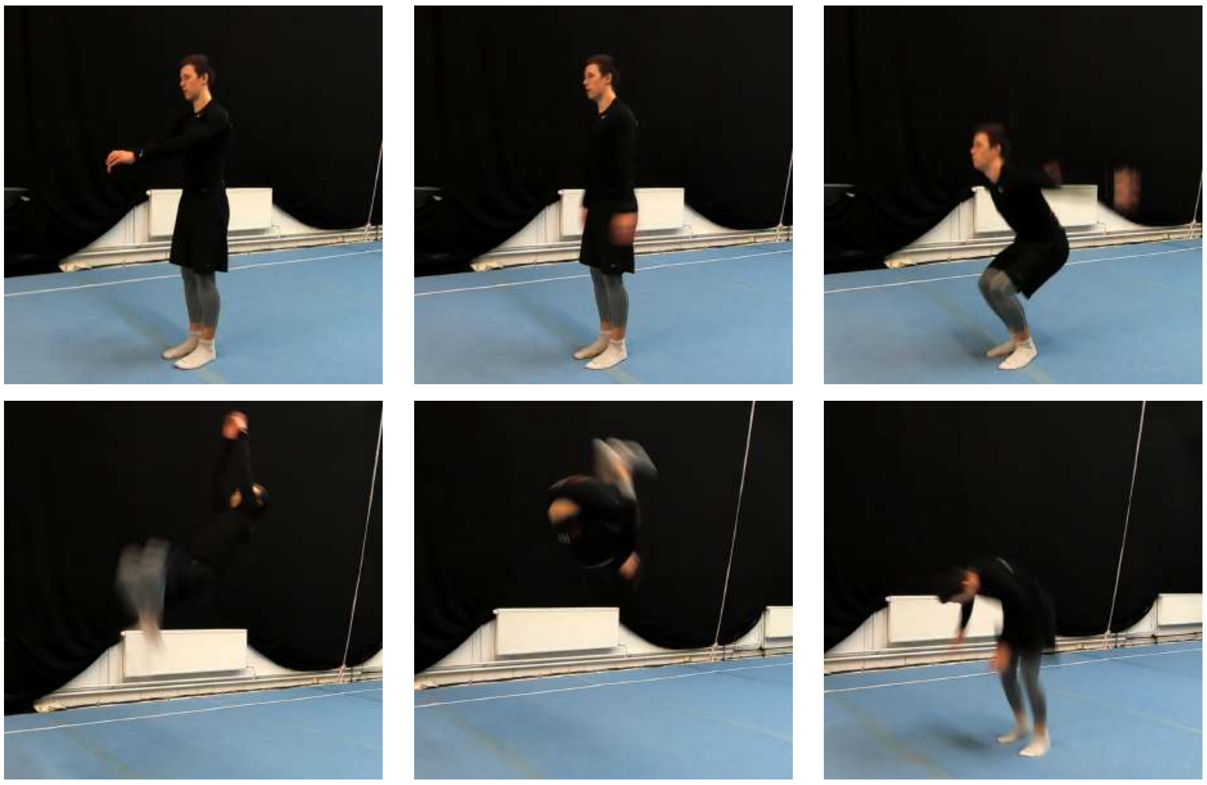

Figure 4.5: The movement of some body parts during this backflip sample are unrecognizable

4.2.6 Results

The final dataset yielded after applying all pre-processing strategies consists of 510 backflip and 620 back handspring samples, totaling to 1130 samples. The samples are persisted as *csv* files for each frame of each sample.

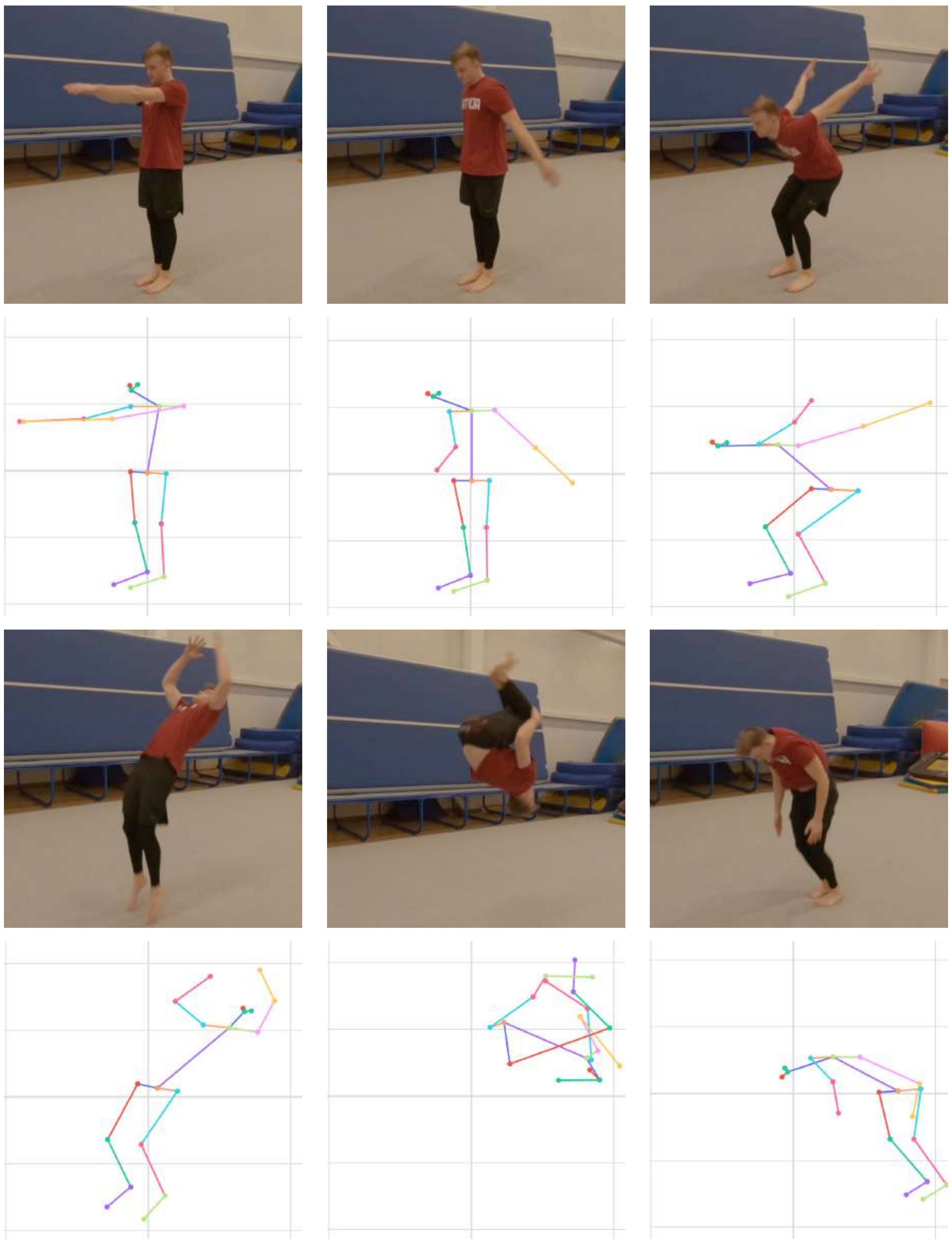

Figure 4.6: Backflip sample centered to coordinate system origin

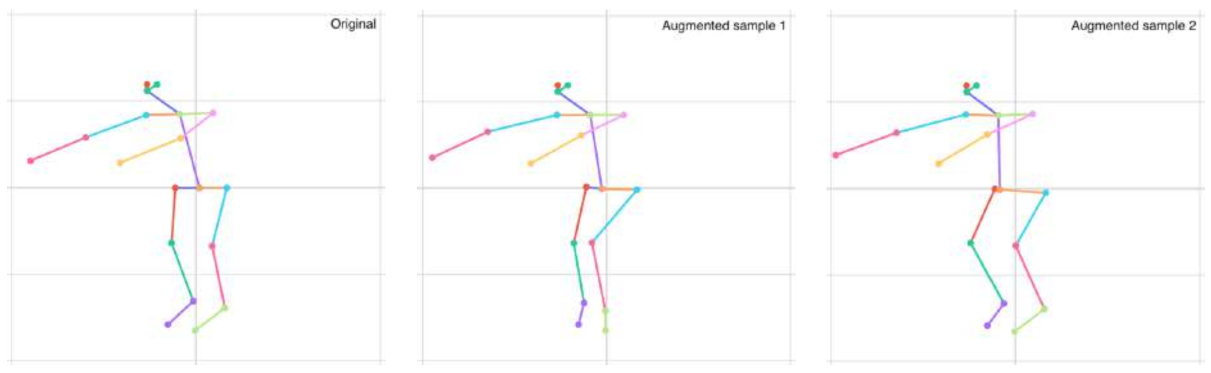

Figure 4.7: Original backflip sample (left) with two augmented samples

Chapter 5

Classifier Development

As demonstrated in recent research by Farzan Majeed Noori, Benedikte Wallace, Md. Zia Uddin and Jim Torresen in [26], a combined architecture of using OpenPose for pose estimation and Recurrent Neural Networks (RNN) for human activity recognition could prove as the new state-of-the-art solution for robust and non-intrusive human activity recognition. In the cited research, OpenPose was used to extract keypoints from a subset of the Berkeley Multimodal Human Action Database (BMHAD) [27]. What makes it remarkable is the classification accuracy obtained using a Recurrent Neural Network with Long Short-term Memory (LSTM) cells over more conventional machine learning classifiers, such as Support Vector Machines, Decision Trees and Random Forests. Since the dataset contained human activities recorded from different angles, the solution is also view-invariant, increasing the robustness of this solution. Another paper, a technical report by Chinmay Sawant [28], also supports the combination of OpenPose with LSTMs for time-series classification, reporting similar accuracy results in a real-time application on the same BMHAD dataset, making use of the efficiency of this solution.

The difference between the cited work and current work is the custom dataset constructed for gymnastics based movements. In contrast to the BMHAD dataset, which includes 11 general activities, such as jumping jacks, waving, and clapping, the dataset used in the current thesis uses more complex biomechanical activities, including human body rotations and airtime, such as backflips and back-handsprings. The author of this paper expects the RNNs used for general action recognition also to be applicable to gymnastics movements. This research aims to validate this hypothesis and investigate neural network architectures most suitable for gymnastics action recognition.

5.1 Methodology

A combination of empirical knowledge with the mathematical theory of artificial neural networks is used as the starting point to develop neural networks capable of recognizing gymnastics movements. The development process starts by training a simple recurrent neural network with gymnastics data and analyzing the results. Simple RNN is chosen primarily for its known property of representing information from context window [29]. RNNs have *long-term memory* in the form of weights, enabling the network to *remember* a gymnastics element represented as a sequence. An iterative process is used to train, validate, analyze, and compare each classifier's results. The steps followed in the current thesis are represented in figure 5.1.

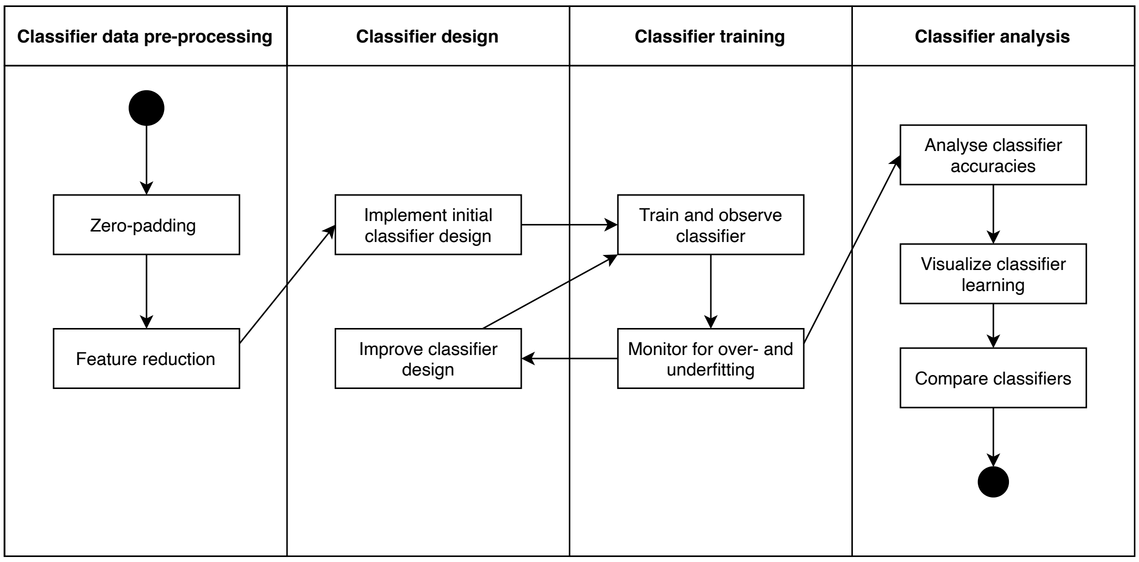
Classifier data pre-processing
Classifier design
Classifier training
Classifier analysis
Zero-padding
Feature reduction
Implement initial classifier design
Improve classifier design
Train and observe classifier
Monitor for over- and underfitting
Analyse classifier accuracies
Visualize classifier learning
Compare classifiers

Figure 5.1: Tasks completed regarding classifier development in the thesis

5.1.1 Classifier Prerequisites

Two prerequisites before feeding training data to the classifiers include:

- *Zero-padding* — Gymnastics activities performed vary in their length, and the samples are zero-padded to keep the classifier dimensions static. Zero-padding is done by finding the longest sample performed and increasing the timesteps of other samples with zero values until they are equal in length to the longest sample.
- *Feature reduction* — Recurrent neural networks are prone to overfitting [30].

With the combination of the author's domain knowledge about gymnastics, the 25 total keypoints obtained during pose estimation are reduced to 15, filtering out less prominent keypoints necessary to recognize gymnastics movements. The 15 keypoints used for classification include the trunk, head, and limbs of the skeleton.

5.1.2 Classifier Algorithms

Classifier algorithms experimented with in the thesis are:

- *Simple RNN* — The first classifier (shown in figure 5.2) experimented with is a network model with one simple recurrent layer. The recurrent layer consists of 2 units, chosen according to the number of outputs the network produces - the movements are either a backflip or a back handspring. A dropout layer with a 0.5 rate to prevent overfitting is added next. An activation layer with rectified linear units follows with a flattening layer to reduce dimensions, and finally, an output layer with softmax activation is used for representing the different activities. Categorical cross-entropy is used as the loss function during stochastic gradient descent with an Adam optimizer with a learning rate of 0.001.
- *Hierarchical RNN* — The Simple RNN works well for the short duration activities analyzed in this paper, but the downside of using one recurrent layer is observability. By feeding all available dimensions into one recurrent layer, we lose the visibility of what exactly the different units in a neural network learn. A more advanced and deeper neural network (inspired by article [23]) consists of several recurrent layers, each representing some logical unit. The model used for experimentation in this paper is shown in figure 5.3. Hierarchical RNN uses multiple input units, each representing a human skeleton subsection. In this example, the skeleton is divided into the left arm, right arm, trunk, left leg, and right leg subsections. In subsequent layers, the recurrent units are fused, and finally, neural activations are applied on a fully connected skeleton layer. One of the advantages of this method is that we can now intercept some layer of interest and observe the neuron activations only regarding a skeleton's subsection.

- *LSTM* — An RNN with LSTM cells was also implemented. The simple recurrent layer is substituted with a hidden LSTM layer with two hidden units, accordingly representing the network's outputs. Other layers remain identical with the *Simple RNN* classifier.

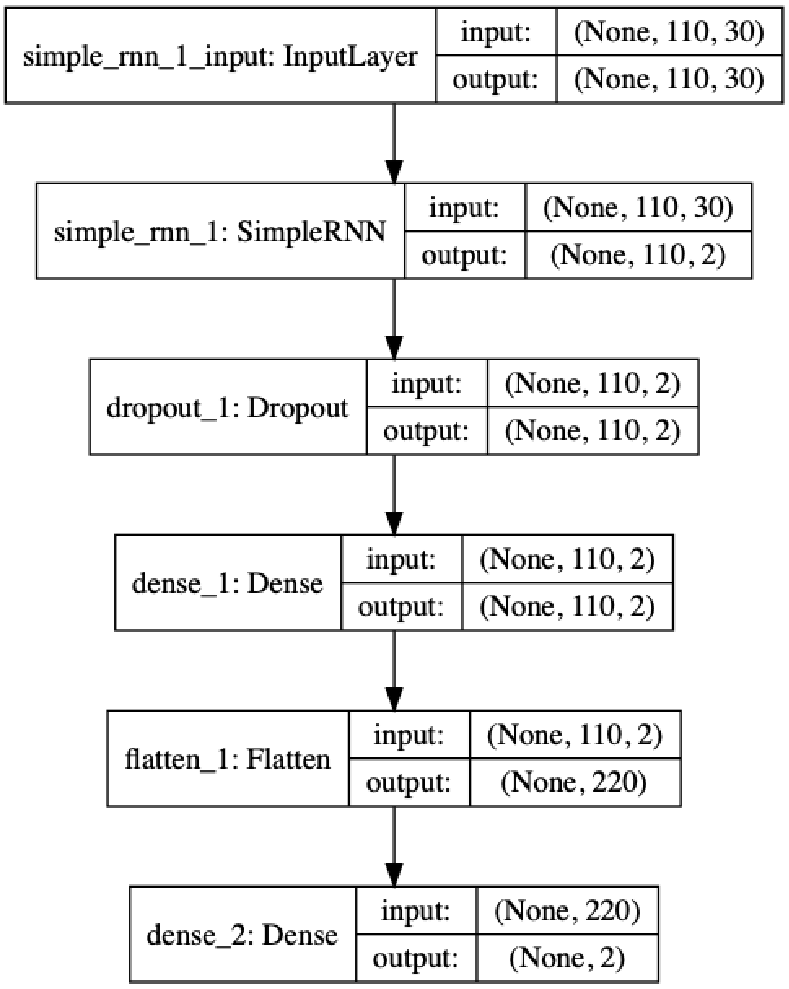

Figure 5.2: The neural network design with a simple recurrent layer

5.1.3 Classifier Validation

Validating the learning process of our neural networks requires the usage of some standard classifier validation techniques. First, we list these validation methods for reference and then explain each one more thoroughly:

- *Test split* — 0.2 rates split the sample data between training and testing data to test for unbiased results when the model training has stopped.
- *Validation split* — A validation split of 0.33 rate is also used while training

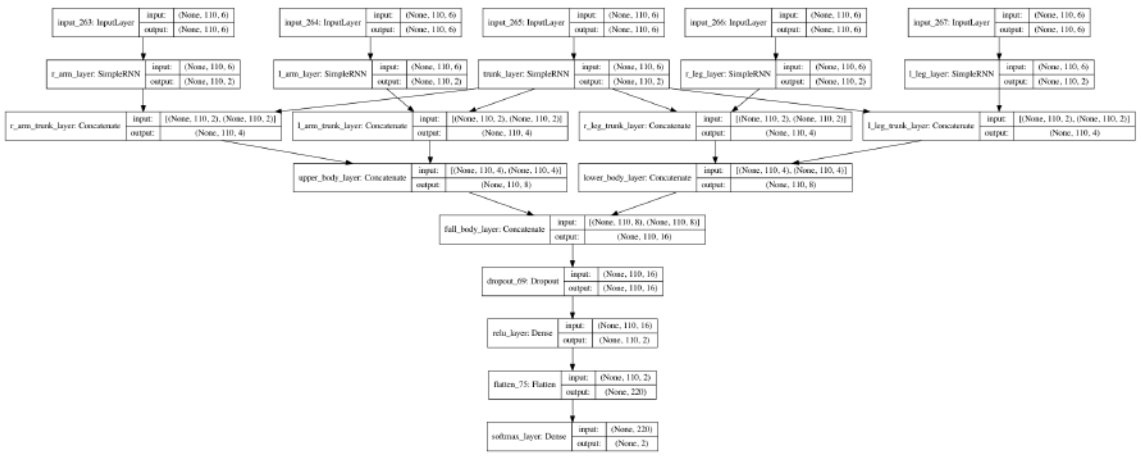
input_263: InputLayer | input: (None, 110, 6) | output: (None, 110, 6)
input_264: InputLayer | input: (None, 110, 6) | output: (None, 110, 6)
input_265: InputLayer | input: (None, 110, 6) | output: (None, 110, 6)
input_266: InputLayer | input: (None, 110, 6) | output: (None, 110, 6)
input_267: InputLayer | input: (None, 110, 6) | output: (None, 110, 6)
r_arm_layer: SimpleRNN | input: (None, 110, 6) | output: (None, 110, 2)
l_arm_layer: SimpleRNN | input: (None, 110, 6) | output: (None, 110, 2)
trunk_layer: SimpleRNN | input: (None, 110, 6) | output: (None, 110, 2)
r_leg_layer: SimpleRNN | input: (None, 110, 6) | output: (None, 110, 2)
l_leg_layer: SimpleRNN | input: (None, 110, 6) | output: (None, 110, 2)
r_arm_trunk_layer: Concatenate | input: [(None, 110, 2), (None, 110, 2)] | output: (None, 110, 4)
l_arm_trunk_layer: Concatenate | input: [(None, 110, 2), (None, 110, 2)] | output: (None, 110, 4)
r_leg_trunk_layer: Concatenate | input: [(None, 110, 2), (None, 110, 2)] | output: (None, 110, 4)
l_leg_trunk_layer: Concatenate | input: [(None, 110, 2), (None, 110, 2)] | output: (None, 110, 4)
upper_body_layer: Concatenate | input: [(None, 110, 4), (None, 110, 4)] | output: (None, 110, 8)
lower_body_layer: Concatenate | input: [(None, 110, 4), (None, 110, 4)] | output: (None, 110, 8)
full_body_layer: Concatenate | input: [(None, 110, 8), (None, 110, 8)] | output: (None, 110, 16)
dropout_69: Dropout | input: (None, 110, 16) | output: (None, 110, 16)
relu_layer: Dense | input: (None, 110, 16) | output: (None, 110, 2)
flatten_75: Flatten | input: (None, 110, 2) | output: (None, 220)
softmax_layer: Dense | input: (None, 220) | output: (None, 2)

Figure 5.3: The neural network design with hierarchical recurrent layers

the RNN to observe the model's loss and accuracy, essential to help with hyperparameter tuning. The validation set is kept separately from the test set, so when we optimize our hyperparameters, we can validate our results against the test set.

- *Monitoring accuracy and loss* — Figure 5.4 demonstrates how the LSTM model accuracy and loss are monitored during epochs to detect overfitting or underfitting. When the training and test set accuracies and losses do not diverge too much, we can be more confident that our classifier is generalizing and not just optimizing for the input data [31].
- *Repeated training on unconnected models* — We repeat each classifier training on five unconnected models. As stated in [32], the optimization of a cost function in recurrent neural networks can potentially lead some minima that are not the most optimal (global). By running our experiment on five unconnected models, we avoid basing our observations and analysis under the influence of one sub-optimal (local) minima.
- *Recurrent neural network regularization* — As we train a deep neural network on a small training set, we experience what is known as the "overfitting" phenomenon, observed when training the classifiers, and achieving a test accuracy of 100%. In our case, this happens due to repeated training on a small dataset with little variance. We add dropout layers to randomly omit half of the feature detectors on each training case [33].

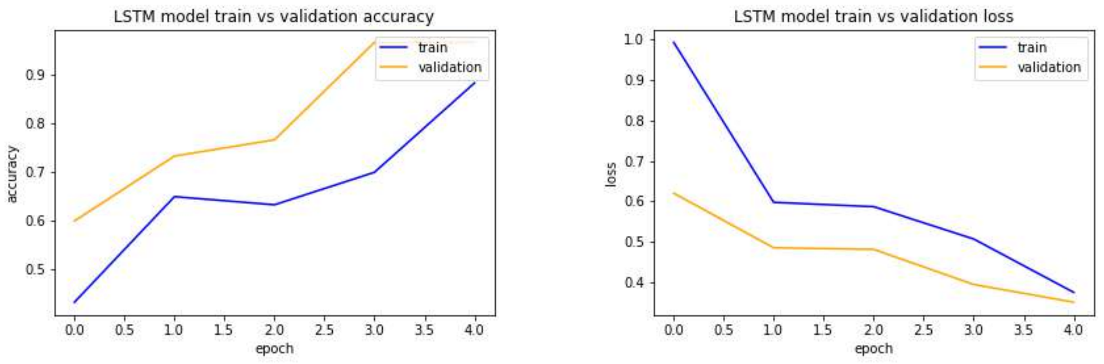

Figure 5.4: Monitoring for model accuracy and loss to detect overfitting/underfitting

5.1.4 Classifier Training Process

SimpleRNN training process

Due to the small sample data size, the RNN classifier is trained for 5 epochs with stochastic gradient descent. The hyperparameters are empirically selected for the classifier, with the goal to maximize the accuracy while preventing overfitting.

An early stopping hook (callback) is also implemented to monitor for validation loss improvements. No improvements in validation loss commonly indicate that the network has stopped learning and is starting to overfit to the training data [31]. After three epochs with no improvements, the hook stops the learning process of the neural network. We adjust recurrent layer units, dropout, the training set size, and epochs until all five models finish training without being interrupted by the callback hook. All five models are depicted in figure 5.5.

LSTM training process

We can observe the test accuracies for the LSTM classifier in figure 5.6. The training process was repeated for five epochs and five separate models. The accuracy and loss monitoring for training and validation sets is depicted in figure 5.7. It is interesting to note that without changing the hyperparameters, the LSTM classifier achieves even higher accuracies than the SimpleRNN classifier. As the LSTM units extend the recurrent self-connecting cells with input and output gates connected to other memory cells [34], it is also more complicated to intercept and visualize the inner activations of this classifier and not covered in the scope of this thesis.

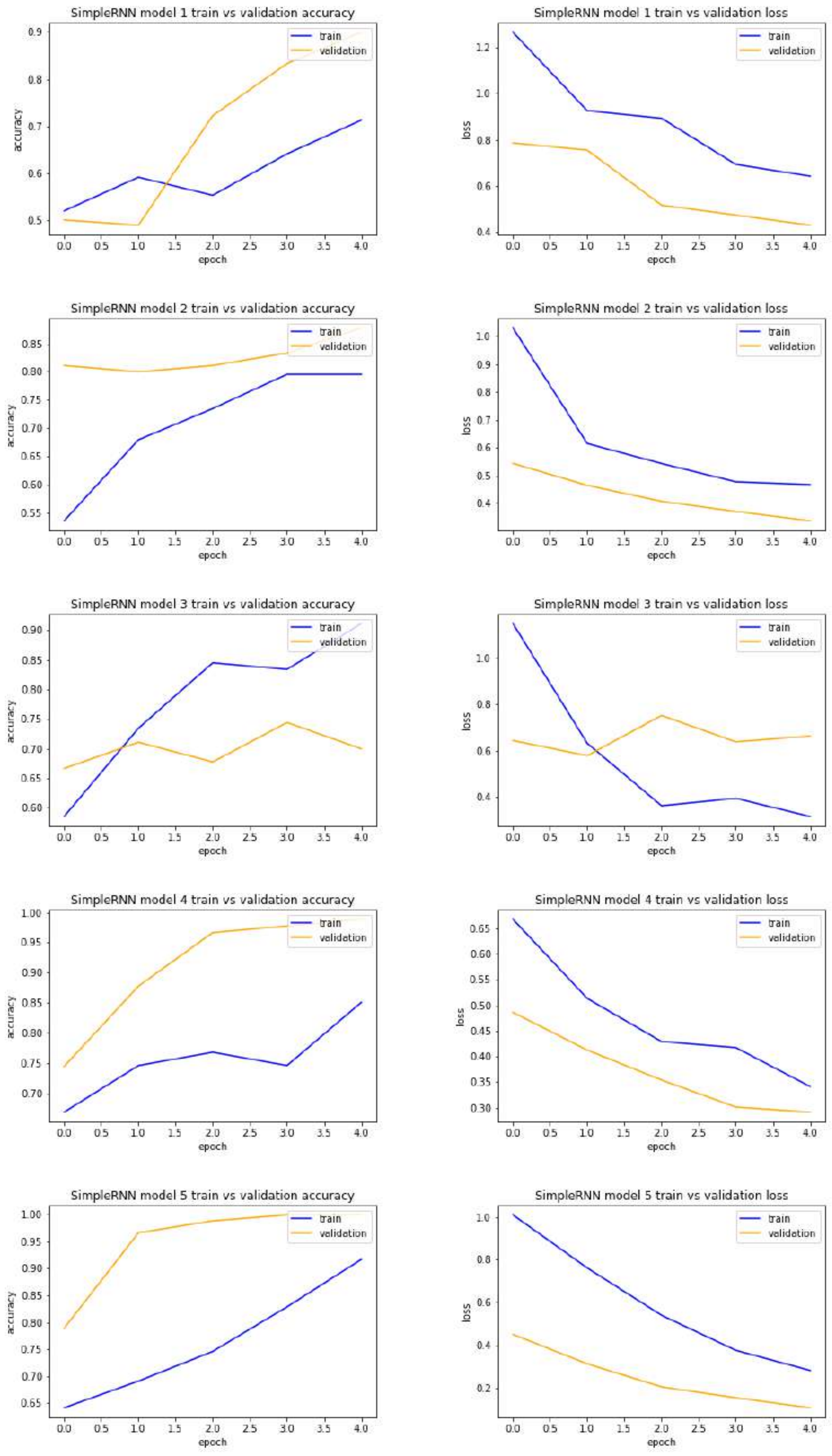

Figure 5.5: Comparison of training and validation accuracies/losses for SimpleRNN

Classifier	**Model 1**	**Model 2**	**Model 3**	**Model 4**	**Model 5**
SimpleRnn	83.824%	92.647%	69.118%	98.529%	100.0%
LSTM	80.882%	100.0%	95.588%	100.0%	97.059%
Hierarchical RNN	80.882%	100.0%	97.059%	92.647%	86.764%

Figure 5.6: Test accuracies obtained during each classifier run

Hierarchical RNN training process

We can observe the test accuracies for the Hierarchical RNN classifier in figure 5.6. This classifier's training process was the same as with the previous two, repeated for five epochs and five separate models. The accuracy and loss monitoring for training and validation sets is depicted in figure 5.8. Hierarchical RNN achieves higher mean test accuracy than SimpleRNN. Since the hyperparameters were not changed for this classifier, we can conclude that the design of Hierarchical RNN is more suitable for the distinction between the backflip and back handspring. Comparing the figures 5.8 and 5.5, we also observe how in the case of Hierarchical RNN the training and validation accuracies/losses tend to approach the same results more closely than in the case of SimpleRNN.

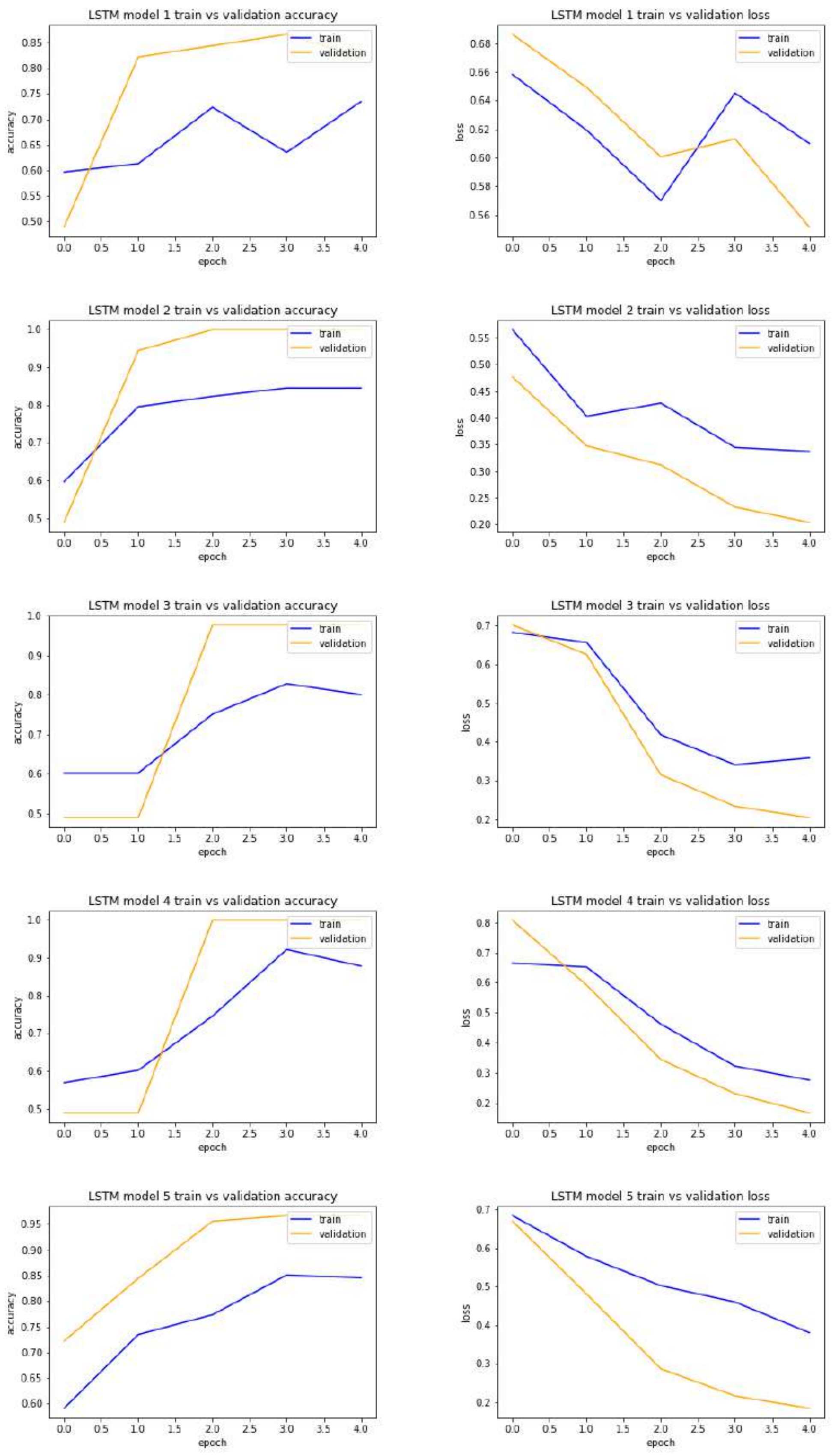

Figure 5.7: Comparison of training and validation accuracies/losses for the LSTM classifier

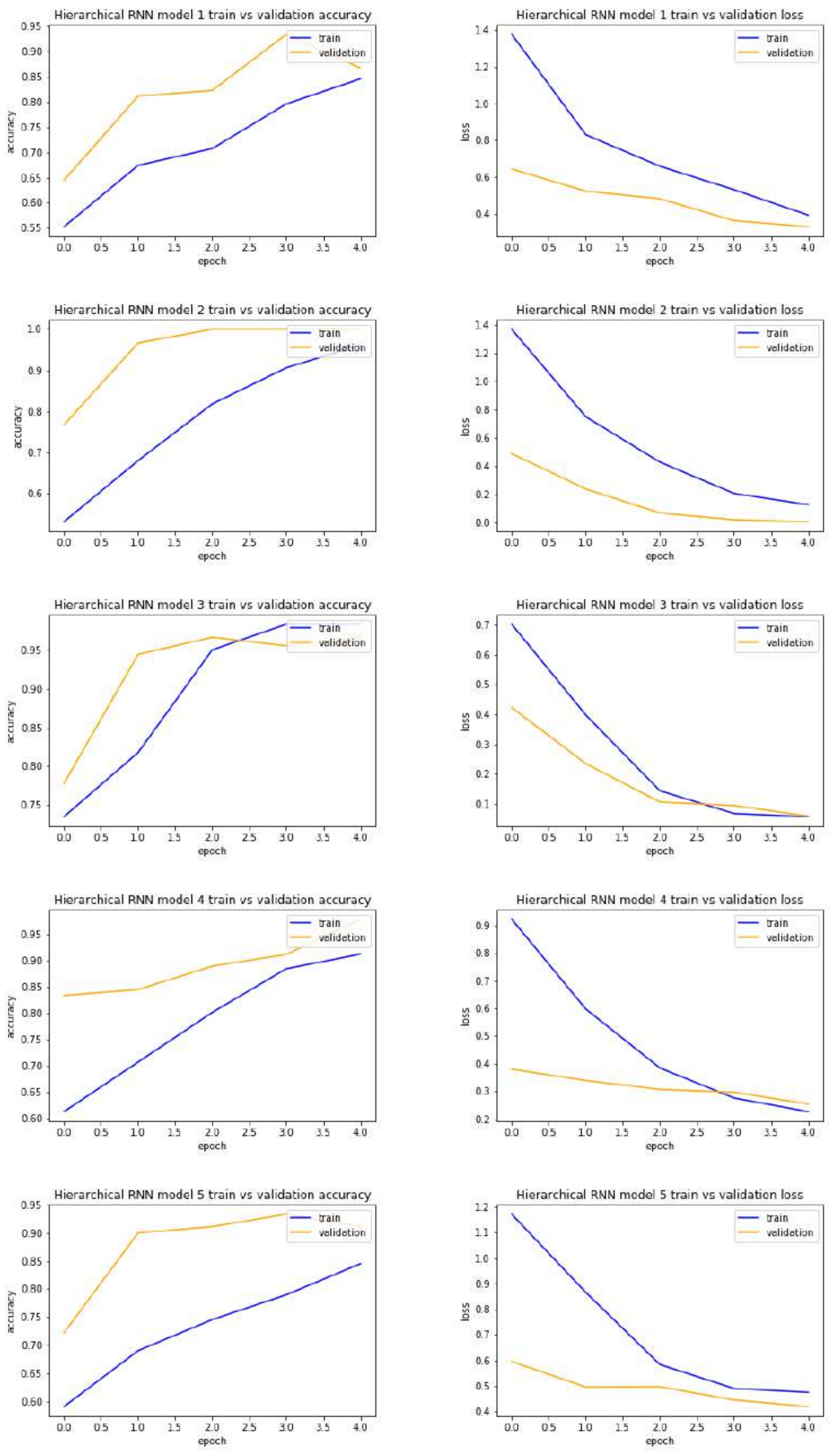

Figure 5.8: Comparison of training and validation accuracies/losses for the Hierarchical RNN classifier

5.2 Classifier Visualization and Analysis

Simple RNN

We choose the fourth experiment run (from table 5.6) to dive deeper into the classifier's inner workings.

By configuring the model's RNN layer to return hidden states, it is possible to intercept this layer's output values and investigate the neuron activations. It is also possible to visualize these activations to understand better what the neurons are learning. The neuron visualization code is based on the code found in *Deep Learning Cookbook* by Douwe Osinga [31], but with a significant difference in the data being used as the input for the recurrent network. In section *5.5 Visualizing Recurrent Network Activations*, Douwe Osinga uses a bitmap to visualize a **text sequence** being provided to the network and intercepting the activations of a specific layer. However, the current work uses **time-series data** representing the skeleton keypoints location for each frame of a gymnastics activity. This requires some modifications to the visualization code provided in the book. The modified code snippet is added as an appendix (A.1) to this thesis, as to the best of the author's knowledge, there are not many examples of recurrent neural network's neuron activation visualization code for time-series skeleton data available.

The author of this work compares neuron activations of backflips (figure 5.9) and back handsprings (figure 5.10). In both figures, we visualize the activations using four horizontal lines. Each horizontal line also consists of three inner lines, where the first represents timestep indices, and the other two represent activations of two hidden layer neurons. The outer horizontal lines are split by 30 timesteps, which correspond to 0.5 seconds of activity time since the activities are recorded at 60 frames per second.

The model correctly predicts the chosen samples. It is interesting to observe how the first neuron's activation starts to flicker when it encounters an activity's ending. We can also observe how back handsprings tend to last longer than backflips, easily noticeable from neuron activations. The second observation is that the first neuron

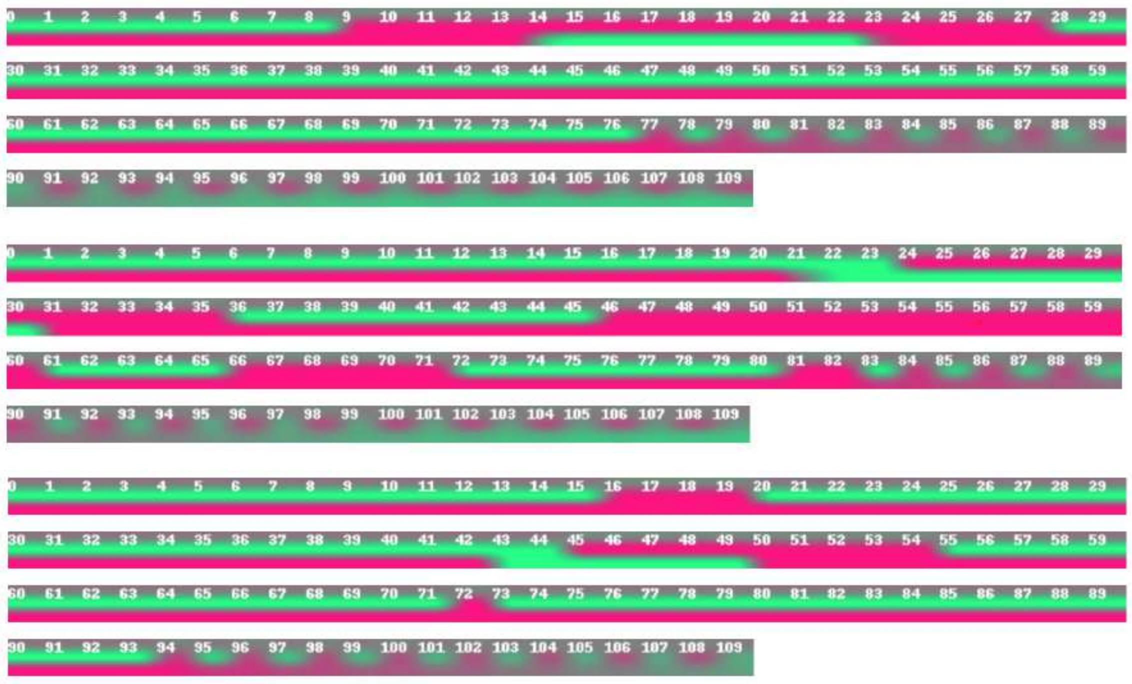

Figure 5.9: Simple RNN neuron activations for three backflip samples

seems to activate throughout almost the whole duration of a backflip.

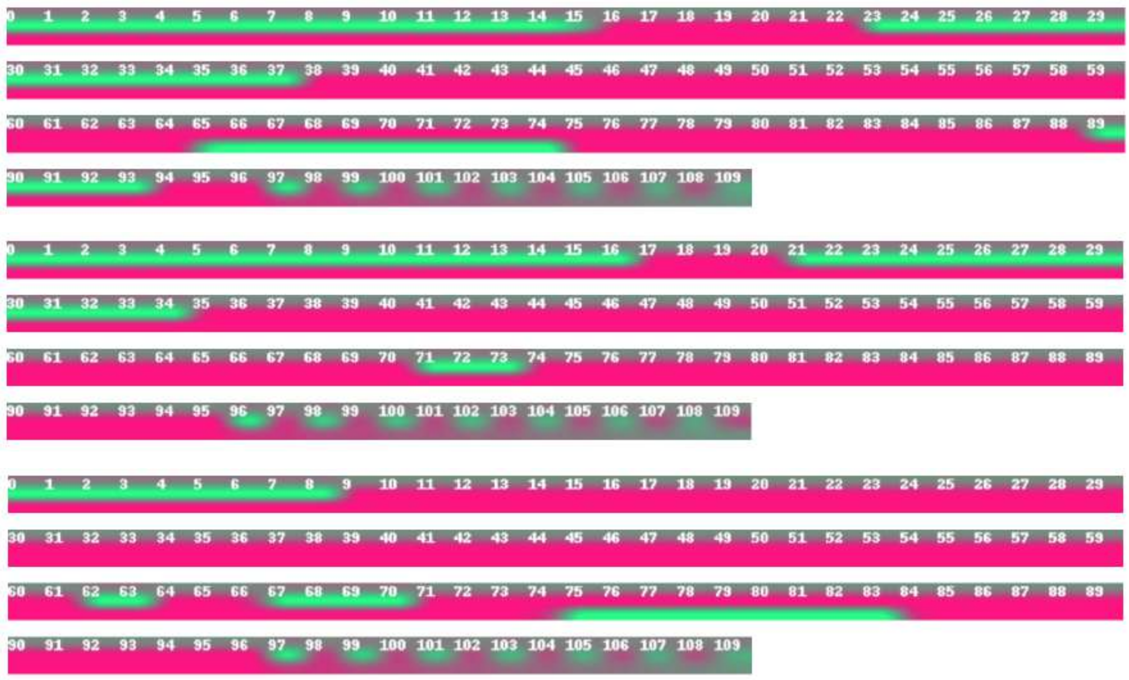

Figure 5.10: Simple RNN neuron activations for three back handspring samples

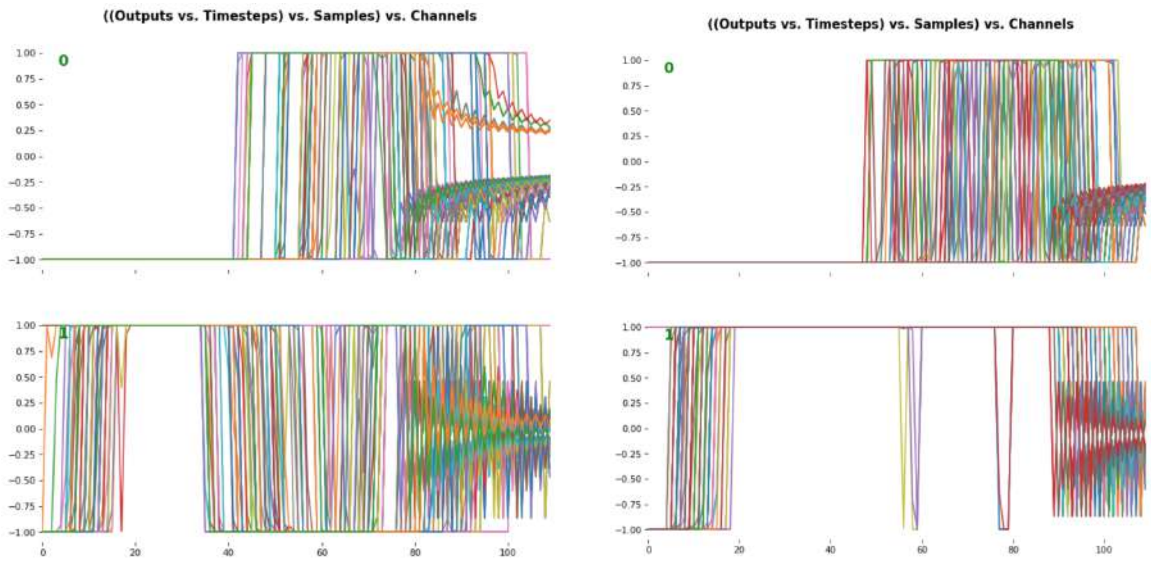

Figure 5.11: All backflip samples (left) and all back handspring samples (right) output activations combined show how neurons "learned" to activate differently for these activities.

Chapter 6

Discussion

6.1 Classifiers comparison

All classifiers presented in this thesis have outstanding results with distinguishing the two gymnastics elements - backflip and back handspring. The results show impressive potential distinguishing short duration complex body moments. It is worth mentioning that the time-series input data also contains full rotation of the human body, separating this work from other example works using recurrent neural networks to recognize simple human actions [28]. Backflips and back handsprings are similar actions in terms of human body movement, differentiating mostly in the way that during the backflip's rotation athlete's arms are not extended, and also noticeable vertical movement occurs during the back handspring push-off phase (section 3.1.4). The table 6.1 depicts all classifiers with corresponding average test accuracy for five unrelated model training runs. The table also contains total wall time for all classifiers with a noticeable difference in the duration, clearly demonstrating how more complex neural network architectures require more computational power.

With the dataset used for experimentation, the classifiers tend to be sensitive to overfitting, requiring heuristic hyperparameter tuning. The author's training process involved training the classifiers to overfit at first runs and using hyperparameter tuning with callbacks to finish training before reaching 100% accuracy. With a training set of 339 samples, almost all recurrent classifiers finish at sub-hundred percent accuracy, enabling the comparison of classifier accuracy ratios.

With equal sample sizes and training epochs, the LSTM finishes with the highest average accuracy of 94.706%; however, due to the complex LSTM units being the most challenging network to interpret. The hierarchical RNN provides more interpretability with body part specific neuron activations and even exceeds the SimpleRNN in terms of accuracy, but requires four times as much training time as the SimpleRNN classifier. In summary, experimental results show positive potential using recurrent network architectures for differentiating gymnastics movements, such as backflips and back handsprings.

Classifier	SimpleRnn	LSTM	Hierarchical RNN
Average test accuracy	88.824%	94.706%	91.470%
Wall time	13min 4s	30min 34s	1h 1min 38s

Table 6.1: More complex classifier architecture strongly impacts wall time when training recurrent neural networks to distinguish gymnastics elements

6.2 Future work

Even though this thesis successfully demonstrates an end-to-end solution for gymnastics action classification, it is not an out-of-the-box readily usable solution. It has a lot of potential for improvements. Following is a brief discussion about some ideas for future work:

1. *Real-time recognition with action classification* — The potential of current work includes packaging the code and deploying it as a backend service. For this, some restructuring of the code is necessary. Also, two main services should be developed for better contextual boundaries. The first service should deal with the pose estimation and pre-processing of the data. For pose estimation, the advice would be to deploy OpenPose on a server with at least one graphics processing unit, and the data-preprocessing strategies have been thoroughly described in section 4.2. The Python-based technology stack used for data pre-processing in this thesis was primarily used for demonstration and validation purposes. The second proposed backend service would be responsible for the classification task of gymnastics action recognition. After initial model training, the model should be deployed as a backend service with prediction API usable by a controller tier application. It should also be feasible to implement the continuous learning of the model. A controller tier application should be developed for a complete real-time solution, aggregating the two main backend services described before. The controller tier application should be deployed onto a device with video recording capabilities, i.e., a smartphone.

2. *Multi-person action recognition* — For increasing the robustness and field usability of the solution developed in the current thesis, concurrent multi-person recognition support should be implemented. Many gymnastics athletes tend to train in groups, and capturing multiple persons in the same frame is unavoidable without interfering too much with the formation of training groups in gymnastics facilities.

   OpenPose already offers the indexing of each frame's pose estimation. By grouping and filtering, it is possible to implement a solution to filter out the bystanders in the subject's frame and target only athletes of interest for action recognition.

3. *3D poses with burst photography* — At the time of writing this paper, researchers Dan Oved, Amelia Pisapia, and Anna Gudnason from the *New York Times* published an article with exciting advancements in the pose estimation field, while also focusing on the sports of gymnastics. Using a similar pose estimation tool *Wrnch* [20], an alternative to *OpenPose*, they have improved real-time pose estimation in gymnastics by constructing a 3D pose by triangulating using 2D poses and using burst photography instead of video, yielding sharper and higher-resolution images than video [35]. Similar improvements could be made to the solution developed in this thesis. Additionally, with improved poses, researchers could extract gymnastics-related metrics, such as jump height, body acceleration, and rotation speed. These metrics could prove as useful features for non-invasive human action recognition. Further research with mentioned metrics should be conducted when improving the recognition of gymnastics elements.

Chapter 7

Conclusion

In short summarization, the contributions made by the author in this thesis include:

- A balanced dataset with backflip and back handspring samples, acquired using a consumer-grade action camera and extracting poses with OpenPose.
- Development of data pre-processing strategies relevant for skeleton-based pose estimation and applicable for standing backflips and back handsprings - activities including human body rotations.
- Recurrent neural network classifiers capable of making successful distinctions between backflips and back handsprings.
- Time-series skeleton-based recurrent neural network output activation interception and visualization for interpreting the outcomes of classifiers.
- Combining two different research fields, pose estimation and human action recognition, to achieve a full prototype capable of recognizing two gymnastics activities.

This thesis's primary purpose was to give an overview of a full end-to-end solution of combining recent advancements in pose estimation with human action recognition in the sports of gymnastics. To the best of the author's knowledge, there is not much literature demonstrating such solutions for gymnastics action recognition. Most previous solutions capturing gymnastics actions use restricting devices or garments. Restricting solutions have not found use in daily training sessions with complex biomechanical movements, such as in the sports of gymnastics. The main requirements set by the author were completed. These included using a consumer-grade action camera (*GoPro Hero 7*) to capture back handsprings and backflips in regular gymnastics training sessions, with no additional equipment. The author demonstrates how open sourced pose estimation tool *OpenPose* could be used as a viable option for capturing the movements. Finally, classifiers with recurrent network architecture were developed and integrated with skeleton-based time-series data as an input. This thesis also includes a demonstration of neuron activations usable with time-series data, uncommon in relevant literature processed by the author. As the author himself is a regular practitioner of this sport, the motivation for experimenting with non-invasive and accessible technology exists.

Although an end-to-end solution is described in this paper, there is much room for improvement to bring this solution into everyday use by athletes. It should also be noted, that most of the computational work was done in a cloud computing environment with professional tier graphical processing units. Before field testing this solution, the data pre-processing strategies developed and used in this thesis also require optimization. The data pre-processing strategies were needed to balance the shortcomings of *OpenPose*. The constructed dataset should also be extended with more relevant actions and angles of samples for better reusability.

Advancements in the accessible and non-invasive pose estimation and human action recognition could bring the technology closer to be used for feedback in live sports events, automated training session documentations, and relevant metrics acquisition from a distance.

Acknowledgments

I want to thank my supervisor Sven Nõmm for his feedback, encouragement, and motivation. Last but not least, I would like to express my deepest gratitude to my girlfriend, friends, and my family for unconditional support and keeping me on track during the 2020's virus pandemic.

Bibliography

[1] Shae Donham-Foutch. Teaching skills and health-related fitness through a preservice gymnastics program. *Journal of Physical Education, Recreation & Dance*, 78(5):35–44, 2007.

[2] Robyn Burgess and Guillermo J. Noffal. Kinematic analysis of the back salto take-off in a tumbling series: Advanced vs. beginner techniques. 2001.

[3] Cheol-Hee Park and Young-Kwan Kim. Kinematic comparisons between yang-1 and yang-2 vaults in men's artistic gymnastics. *Korean Journal of Sport Biomechanics*, 24(4):317–327, 2014.

[4] Ayaka Yamada, Yosuke Ikegami, Tomoyuki Horikawa, Hiroki Obara, Yoshihiko Nakamura, and Mutsumi Harada. Dynamics analysis of artistic gymnastics with video motion capture: Pommel horse and vault. In *2019 58th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*, pages 1093–1098. IEEE, 2019.

[5] Michalis Vrigkas, Christophoros Nikou, and Ioannis A. Kakadiaris. A review of human activity recognition methods. *Frontiers in Robotics and AI*, 2:28, 2015. ISSN 2296-9144. doi: 10.3389/frobt.2015.00028. URL `https://www.frontiersin.org/article/10.3389/frobt.2015.00028`.

[6] Acrobatic gymnastics program rules and policies. `https://usagym.org/PDFs/Acro/Rules/2019_rules_policies.pdf`, last accessed on 3 Aug 2020, 2019.

[7] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. *CoRR*, abs/1611.08050, 2016. URL `http://arxiv.org/abs/1611.08050`.

[8] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *CoRR*, abs/1812.08008, 2018. URL `http://arxiv.org/abs/1812.08008`.

[9] Ahmed Elhayek, Edilson de Aguiar, Arjun Jain, Jonathan Tompson, Leonid Pishchulin, Micha Andriluka, Chris Bregler, Bernt Schiele, and Christian Theobalt. Efficient convnet-based marker-less motion capture in general scenes with a low number of cameras. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[10] I. H. López-Nava and A. Muñoz-Meléndez. Wearable inertial sensors for human motion analysis: A review. *IEEE Sensors Journal*, 16(22):7821–7834, 2016.

[11] Spiros Prassas, Young-Hoo Kwon, and William A. Sands. Biomechanical research in artistic gymnastics: a review. *Sports Biomechanics*, 5(2):261–291, 2006. doi: 10.1080/14763140608522878. URL `https://doi.org/10.1080/14763140608522878`. PMID: 16939157.

[12] Nvidia t4 tensor core gpu for ai inference — nvidia data center. `https://www.nvidia.com/en-us/data-center/tesla-t4`, last accessed on 3 Aug 2020.

[13] Mark D Sleeper, Lisa K Kenyon, and Ellen Casey. Measuring fitness in female gymnasts: the gymnastics functional measurement tool. *International journal of sports physical therapy*, 7(2):124, 2012.

[14] German Vicente-Rodriguez, C. Dorado, I. Ara, J. Perez-Gomez, H. Olmedillas, S. Delgado-Guerra, and J. A. L. Calbet. Artistic versus rhythmic gymnastics: Effects on bone and muscle mass in young girls. *International Journal of Sports Medicine*, 28, 2007. doi: 10.1055/s-2006-924397.

[15] Gabriella Penitente, Franco Merni, and William Sands. Kinematic analysis of the centre of mass in the back handspring: A case study. *Gym Coach*, 4:1–11, 01 2011.

[16] Mickael Begon, Pierre-Brice Wieber, and Maurice Raymond Yeadon. Kinematics estimation of straddled movements on high bar from a limited number

of skin markers using a chain model. *Journal of biomechanics*, 41(3):581–586, 2008.

[17] Diogo C Luvizon, David Picard, and Hedi Tabia. 2d/3d pose estimation and action recognition using multitask deep learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5137–5146, 2018.

[18] Tomas Pfister, James Charles, and Andrew Zisserman. Flowing convnets for human pose estimation in videos. 06 2015. doi: 10.1109/ICCV.2015.222.

[19] Posenet. `https://github.com/tensorflow/tfjs-models/tree/master/posenet`, last accessed on 3 Aug 2020.

[20] wrnchai. `https://wrnch.ai/technology`, last accessed on 3 Aug 2020.

[21] Openpose requirements and dependencies. `https://github.com/CMU-Perceptual-Computing-Lab/openpose/blob/master/doc/installation.md#requirements-and-dependencies`, last accessed on 3 Aug 2020, .

[22] Openpose python pose extraction module. `https://github.com/AllarVi/openpose-extract`, last accessed on 3 Aug 2020, .

[23] Yong Du, W. Wang, and L. Wang. Hierarchical recurrent neural network for skeleton based action recognition. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1110–1118, 2015.

[24] Alex Hernández-Garc´a and Peter König. Data augmentation instead of explicit regularization. *arXiv preprint arXiv:1806.03852*, 2018.

[25] Juan C. Núñez, Raúl Cabido, Juan J. Pantrigo, Antonio S. Montemayor, and José F. Vélez. Convolutional neural networks and long short-term memory for skeleton-based human activity and hand gesture recognition. *Pattern Recognition*, 76:80 – 94, 2018. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2017.10.033. URL `http://www.sciencedirect.com/science/article/pii/S0031320317304405`.

[26] Farzan Majeed Noori, Benedikte Wallace, Md. Zia Uddin, and Jim Torresen. A robust human activity recognition approach using openpose, motion features, and deep recurrent neural network. In Michael Felsberg, Per-Erik Forssén, Ida-Maria Sintorn, and Jonas Unger, editors, *Image Analysis*, pages 299–310, Cham, 2019. Springer International Publishing. ISBN 978-3-030-20205-7.

[27] G. Kurillo R. Vidal F. Ofli, R. Chaudhry and R. Bajcsy. Berkeley mhad: A comprehensive multimodal human action database. In *IEEE Workshop on Applications on Computer Vision (WACV)*, 2013.

[28] Chinmay Sawant. Human activity recognition with openpose and long short-term memory on real time images. Technical report, EasyChair, 2020.

[29] Zachary Chase Lipton. A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019, 2015. URL `http://arxiv.org/abs/1506.00019`.

[30] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *CoRR*, abs/1409.2329, 2014. URL `http://arxiv.org/abs/1409.2329`.

[31] Douwe Osinga. *Deep Learning Cookbook*. OŔeilly Media, Inc., 2018. `https://www.oreilly.com/library/view/deep-learning-cookbook/9781491995839`.

[32] Monica Bianchini, Marco Gori, and Marco Maggini. On the problem of local minima in recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(2):167–177, 1994.

[33] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012.

[34] Ralf C Staudemeyer and Eric Rothstein Morris. Understanding lstm–a tutorial into long short-term memory recurrent neural networks. *arXiv preprint arXiv:1909.09586*, 2019.

[35] Estimating 3d poses of athletes at live sporting events. `https://rd.nytimes.com/projects/estimating-3d-poses-of-athletes-at-live-sporting-events`, last accessed on 3 Aug 2020.

Appendix A

A.1 Visualizing recurrent network activations with time-series skeleton data

```
def get_activations(model, example_sample):
    n_timesteps, n_features = example_sample.shape[0],
        example_sample.shape[1]

    x = np.zeros((1, n_timesteps, n_features))

    for t, timestep in enumerate(example_sample):
        for f, feature in enumerate(timestep):
            x[0, t, f] = example_sample[t][f]

    output = model.get_layer('simple_rnn_1').output

    f = K.function([model.input], [output])

    return f([x])[0][0]

activations = get_activations(example_model, example_sample)

def get_image(img, n_timesteps, img_idx, cell_size=48):
    img_width = n_timesteps * 25
    cell_size = int(img_width / n_timesteps)

    pil_image = PILImage.fromarray(img.astype(np.uint8))
    resized_pil_image = pil_image.resize((img_width, cell_size))

    draw = ImageDraw.Draw(resized_pil_image)

    for n_timestep in range(n_timesteps):
        text = str((img_idx * 30) + n_timestep)
        xy = (n_timestep * cell_size, 0)

        draw.text(xy, text)

    f = BytesIO()
    resized_pil_image.save(f, 'png')
    return Image(data=f.getvalue())

def visualize_neurons(act, cell_size=48):
    n_neurons = act.shape[1]
    n_timesteps = act.shape[0]

    fill_value = 128

    img = np.full((n_neurons + 1, n_timesteps, 3), fill_value)

    # add 1 to each value in matrix and then divide by 2
    scores = (act[:, :].T + 1) / 2

    img[1:, :, 0] = 255 * (1 - scores)
    img[1:, :, 1] = 255 * scores

    first_hs_img = img[:, :30, :]
    second_hs_img = img[:, 30:60, :]
    third_hs_img = img[:, 60:90, :]
    fourth_hs_img = img[:, 90:, :]

    imgs = [first_hs_img,
            second_hs_img,
            third_hs_img,
            fourth_hs_img]

    actual_imgs = []
    for i, img in enumerate(imgs):
        n_img_timesteps = img.shape[1]

        actual_imgs.append(get_image(img, n_img_timesteps, i))

    return actual_imgs

example_sample_imgs = visualize_neurons(activations)

for img in example_sample_imgs:
    display(img)
```