IT UNIVERSITY OF COPENHAGEN

Vision Based Human Motion Analysis In Gymnastics

Sofus Sebastian Schou Konglevoll - sosk@itu.dk
Morten Holck Ertner - mert@itu.dk
Gustav Bakhauge - guba@itu.dk

Bachelor thesis
Data Science

Sofus Sebastian Schou Konglevoll
Morten Holck Ertner
Gustav Bakhauge

May 9, 2023
Supervisors: Sami Brandt & Stella Grasshof

ABSTRACT

Human action recognition (HAR) in sports is an evolving field with an abundance of possible use-cases. One such use case is in the field of gymnastics, where many applications using HAR could provide value for people engaged in the sport. Amateur gymnasts could use HAR to learn or improve upon their fundamental skill set, while professional athletes could use it to perfect their technique, by receiving fine-grained evaluations of their movement. To realize these applications, models that can identify sub-actions in a sequence of actions need to be implemented. Our thesis seeks to explore this underlying need.

Among others, we introduce a custom data set, containing homemade gymnastic videos from YouTube. The videos were manually labeled frame-by-frame and skeleton data was extracted by the use of human pose estimation methods. The extracted skeleton data was then fed to a convolutional neural network(CNN) and a long-short term memory(LSTM). We mainly evaluated and trained four models, a CNN ensemble, an LSTM ensemble, a CNN with averaged hyperparameters, and an LSTM with averaged hyperparameters, using a nested k-fold cross validation method.

We found that both architectures had difficulties differentiating between specific labels in our data, which we believe is a consequence of our labeling decisions. From our findings, we conclude that the LSTM architecture consistently outperforms the CNN architecture. Our findings also show a strong indication that this is due to the LSTM being able to capture a temporal aspect in the data, which the CNN fails to capture. Furthermore, we conclude that our ensembles generally outperform the models with averaged hyperparameters by a small margin, both in terms of scoring metrics and standard deviation.

CONTENTS

1 INTRODUCTION

Action recognition is the sub-field of computer vision that specifically focuses on the identification and classification of different human actions. It is an interesting and important field of research due to its broad range of use cases such as crowd dynamics, visual surveillance systems, and sports analysis. In the field of sports analysis, one area of study is gymnastics where the possibilities of action recognition can be leveraged to further develop the skill level of performing athletes. It can help expand their knowledge of how certain phases of a skill are performed by comparing their execution with other athletes. It can also be utilized in the competitive scene, where action recognition tools can be used to automatically detect and score the execution of skills, due to the presence of a strict scoring ruleset.

To develop models that are able to evaluate different actions in gymnastics, one must first identify sub-actions such as a take-off, a skill, or a landing in a video sequence. Similar research in other sports such as football [20] and basketball [15] have been made, where the focus is trying to classify sub-actions like *dribble*, *pass*, *shoot* etc. Both of these papers had to create or expand upon existing data sets, which seems to be a general issue in the field of fine-grained action recognition in sports. For coarse-grain action recognition, many good and comprehensive data sets exist, which contain many different types of actions. Data sets such as the NTU RGB+D [11] contain many different coarse-grain actions, spanning over multiple fields, while Sports-1M [8] and FSD-10 [13], has a specific focus on sports. In gymnastics, a data set called FineGym [16] exists, which provides a detailed level of fine- and coarse-grained actions. In this paper, we address sub-action classification in gymnastics. However, we were unable to access the FineGym data set, resulting in a lack of readily available and usable data.

Since no ideal data set were available to us, we deemed it necessary to create a custom data set, which consists of 202 videos, 20000 frames, with each frame labeled one of the following: idle, take-off, skill, or landing.
Many different visual modalities exist in the field of action recognition, each of these modalities comes with its own strengths and weaknesses and is suited for different tasks. The RGB modality is commonly used due to its flexibility and the relative ease of obtaining data. It does have some drawbacks due to its sensitivity to background and viewpoint, and therefore a more suited modality for human action

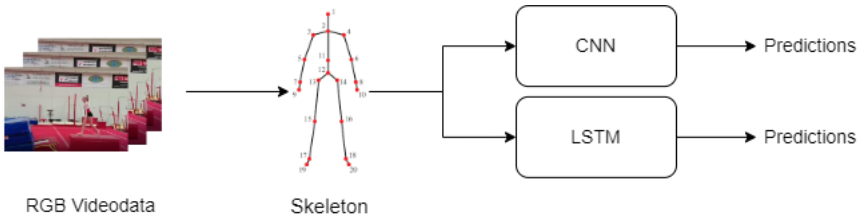


Figure 1: Pipeline showcasing how skeleton data is extracted from RGB videodata and fed into two neural networks.

recognition is the skeleton modality. The skeleton modality is, compared to the RGB modality, less sensitive to both viewpoint and background, resulting in a robust skeleton-representation of the human body. Obtaining this skeleton can be done with a method called human pose estimation (HPE), which extracts a vector representation of human body joints in a 2- or 3-dimensional space.
The data derived from the skeleton modality is not only sparse and view insensitive, but also captures geometric relations between body limbs, such as angles, which we believe is an important metric when it comes to action classification in gymnastics. Thus, we have chosen to use skeleton modality for our classification task.
We implemented two commonly used models for action recognition, namely a convolutional neural network (CNN) and a long-short term memory (LSTM). The models we deploy are of a simpler nature when compared to the current state-of-the-art models, this was a deliberate choice to limit the scope of our project. In figure 1 the pipeline of how RGB pictures are turned to predictions is shown.

This thesis will focus on sub-action classification on a custom-made data set of gymnastic videos from YouTube. More specifically, we will compare two machine learning models, a simple CNN, and a simple LSTM, on the task of frame-by-frame sub-action classification on skeleton data extracted from homemade gymnastic videos. Furthermore, amongst the two models themselves, we compare ensembles of base models, with *averaged models*, where use the averaged hyperparameters of the respective base models.
The rest of the paper is organized as follows. In chapter 2 we review different methods of human pose estimation and how different sophisticated RNNs and CNNs are used on skeleton data. In chapter 3 we explain the process behind creating our custom data set, as well as introducing the implementation of our two models. The procedure of how we conducted our experiments will be explained in chapter 4 together with a brief presentation of our results. Lastly, in chapter 5 we conclude our experiments, based on a discussion of our results and limitations, as well as propose suggestions for future work.

2 RELATED WORK

2.1 DIFFERENT MODALITIES IN HUMAN ACTION RECOGNITION

Human Action Recognition in short HAR, is the field of classifying human actions such as those shown in 1. Typically a machine learning model is trained on top of a data source to classify the human action. In HAR there are many different types of data, called modalities. The work of [18] gives an excellent overview of all the different modalities used in the field currently, such as depth, point cloud, radar, etc. A commonly known modality mentioned in the paper is the **Red-Green-Blue modality** (RGB), which is colored photos. RGB is widely used in applications such as visual surveillance, autonomous navigation and sports analysis[17]. However, for HAR, using RGB is often challenging due to colours, background, objects, viewpoint etc.
A modality that is not as susceptible to noise in the background, is the skeleton modality, which is created by applying human pose estimation methods on RGB images.

2.2 HUMAN POSE ESTIMATION

In recent years, deep learning algorithms has become the number one option for solving computer vision tasks due to its ability to generalize complex patterns, which is also why it is used for all modern human pose estimation algorithms. HPE generally works with three different body models. One is the planar model, which captures the shape of a human body by rectangles that approximate

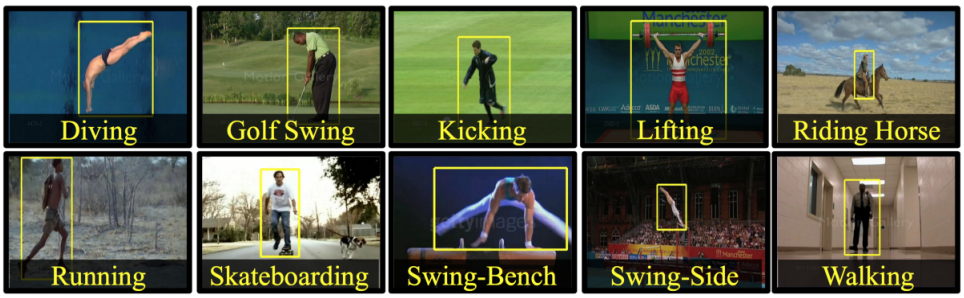


Figure 1: Example on Human Action Recognition (HAR). Source: https://www.crcv.ucf.edu/data/UCF_Sports_Action.php

the human body contours as seen in [7]. Another is the Volumetric model, which captures the shape of a human body, as well as capturing a more detailed representation of soft-tissue in human body parts. A common used model for this is he Skinned Multi-Person Linear model[14]. Lastly, the most commonly used model in HAR is the kinematic model, which creates a flexible graph-like skeleton representation of the body. Several opensource body detection systems are available and capable of creating a kinematic model. Such systems include OpenPose [4] that utilizes a bottom-up approach to create a realtime multi-person 2D-pose estimation. AlphaPose [6], on the other hand, utilizes a top-down approach to human pose estimation by firstly constructing bounding boxes for each human present in a frame, and then independently estimates the pose within each bounding box. Similarly, Blazepose [2] utilizes a two step pipeline to predict the 3-dimensional body estimation of a single person. First part of the pipeline detects a region of interest using a face detection algorithm called BlazeFace [3]. The second part of the pipeline is a neural network, which predicts 33 keypoints on the person in the region of interest.

2.3 MACHINE LEARNING ON SKELETON DATA

In general, common machine learning models used for HAR can be divided into 3 architectures: Convolutional Neural Network (CNN) based methods, Recurrent Neural Network (RNN) based methods and Graph Convolutional Network (GCN) based methods [19]. In this thesis we'll focus on the two former.

2.3.1 CNN-based Methods

Generally, CNN-based models are powerful for 2d image classification tasks, as they use convolutional layers to extract informational patterns from the image. However, a simple CNN is not able to capture temporal aspects between images, which is an important feature to have in HAR. To accommodate this issue, extensions to the basic CNN have been proposed. The following section will review some of the current research in the field of CNN-based models in HAR on skeleton data.
Applying CNN on temporal data can generally be divided into two methods. One method is to create pseudo-images by combining multiple frames into one. These pseudo images contain information about the join-trajectories. This methods is used in both [9] and [22], where the latter additionally to capturing joint-trajectories, also maps spec-

trum coding of body parts and joint velocity. Pseudo-images captured from both methods then encodes both spatial information in the frames, but also temporal information between frames.

The other method is using an extension of the simple CNN, namely the Temporal Convolutional Network(TCN). This method was proposed by Lea et al.[10], and utilizes a hierarchy of temporal convolutions to capture long-term relations in the data. Two different ways of achieving this was proposed, an encoder-decoder TCN (ED-TCN), which uses temporal convolutions, pooling and up-sampling, and a dilated TCN, which uses a deep stack of dilated convolutions to recognise longer temporal relations.

2.3.2 RNN-based Methods

The advantage of using an RNN over a regular neural network is its ability to learn the temporal aspect of the data, making it very suitable for HAR on videos. More specifically, one of its gated variants, the LSTM, is specifically good at capturing long term dependencies between data points. This is done with different gates, determining what information should be remembered and what should be forgotten. In the following section, we will briefly introduce some of the current RNN and LSTM models used for HAR on skeleton data.

Yong Du et al. [5] proposes an end-to-end hierarchical RNN, dividing the skeleton into five different parts: two legs, two arms and one trunk. These body parts are then fed to five bidirectional RNNs, after which the different body parts are connected hierarchically to form more complete representations of the skeleton throughout the deep network. Similarly, Zhang et al. [23] enumerates eight geometric features describing relationships between all limbs and inputting this information to a 3-layer LSTM.

Alternatively to alter the representation of the data, research have have also been done in altering the models themselves. Veeriah et al. proposes a differential RNN (dRNN)[21] as an alternative to using an LSTM to learn spatio-temporal features. The dRNN models dynamics of actions by computing different-orders of *Derivatives of States*, so that the models learns the dynamic saliency of the actions performed, unlike a regular LSTM where non-salient motion information is also learned. A variant of the standard LSTM is the Global Context-Aware Attention LSTM (GCALSTM) proposed by Liu et al. [12] where informative skeleton joints are identified and focused on, while less informative joints are ignored. To do this, a global context memory cell is introduced and fed to each step of the LSTM.

3 METHODS

3.1 DATA COLLECTION

To perform frame-by-frame HAR on gymnastics videos, we needed a data set containing many short clips, each containing one Skill, like somersaults, back-flip, etc, with each frame having a label. Furthermore, we wanted to perform our research on videos captured with regular-use cameras, as well as subjects who weren't necessarily professional athletes, to simulate a real-world scenario. Many of the data sets we found were either not publicly available or they were labeled on an action level instead of sub-action i.e each clip had only one given label. Due to this lack of data, we decided to create a data set. Criteria for the videos were 1. only one person in the frame, 2. the quality of the video had to be decent and 3. the camera had to be still. It was difficult finding videos that strictly followed these rules, however, we managed to find 8 videos on YouTube, that almost did (a few clips had multiple people in them or had handheld cameras). These eight videos were split into 202 clips, totaling in around 19680 frames, where each clip contains one person doing a single Skill. However, a few of the clips were filmed on trampolines, where a single clip could contain multiple Skills, and other clips had the subject performing a cartwheel before doing the Skill.

3.1.1 Annotating

After acquiring the clips, we needed to label them. We decided on four different sub-tasks: *Idle*, *Take-off*, *Skill* and *Landing*. Idle was determined to be whenever the subject did not do any of the other three sub-tasks, i.e if a person was standing, walking around or taking more than two steps after the Landing. Take-off is when the hands were raised synchronously and the knees bent, meaning the subject was ready for take off. In some cases the subject performs the movement of a Take-off followed by an Idle action, this in turn results in the sequence being labeled as Idle, since no Skill would follow the Take-off. A Skill starts after the subject leaves the ground and will generally continue until the subject lands again. In the cases where the subjects where on trampolines or doing a cartwheels before hand, the cartwheel and the multiple flips would all be classified as Skill. Lastly, Landing starts when the subject finishes their Skill sequence

(a) Idle (b) Take-off

(c) Skill (d) Landing

Figure 1: Example of body poses for the four different labels with the HPE skeleton applied

and ends in two ways: either if the subjects takes two steps after the Landing, after which the rest of the clip would be labeled Idle, or if the clip ends. This limit of two steps was chosen to resemble the conditions found in the competitive scene. Figure 1 shows the four labels on an example clip.
Each clip were manually annotated frame by frame according to the above specifications. This process was done by all of us individually, and for each frame, a majority vote of the three votes was used to decide on a final label. In the few cases (284 frames) where we all disagreed, we annotated it "Unlabeled". For each clip, a file containing labels for each frame was made.

3.1.2 Applying HPE

For extraction of the skeleton data, we used the lightweight convolutional neural network architecture for human pose estimation BlazePose[2] proposed by Bazarevsky et al.. This architecture utilizes a two-step pipeline; First, it roughly detects a region of interest (ROI) within the frame. The ROI is found, unlike many other HPE algorithms, by using a face detection algorithm, BlazeFace [3] in this case, as a proxy for a person-detector. Furthermore. This face detector also acts as a predictor for alignment parameters such as the middle point between the person's hips, the size of the circle circumscribing the whole person, and incline, which is the angle between the lines connecting the two

mid-shoulder and mid-hip points. The second part of the pipeline is a neural network that predicts 33 key points of the person, based on the person alignment provided by the first part of the pipeline. Prediction of the key points is done using a combination of heat map-, offset- and regression approaches. Here, a heat map and offset are used only during training, after which the corresponding output layers are removed before running inference.
To make the network faster and require less computational power, data augmentation is performed. Firstly the person is centered such that the point between the hips is located at the center of the square image created by the person alignment in the first part of the pipeline. Secondly, the person is rotated such, that the line, $\mathbf{l}$, between mid-hip and mid-shoulder points are aligned with the Y-axis. Furthermore, occlusion augmentation is simulated during training, to allow constant tracking of a person, even in cases of significant occlusion.
We chose BlazePose as it has been proven to work great on dance and HIIT (high intensive interval training) [1] data, which has resembles to gymnastics data with fast and intricate movement. Additionally, it has a publicly available, easy-to-implement Python library. Lastly, its ability to do pose estimation in real-time, makes it a good candidate, if one were to further develop this project into a real-time gymnastics coach.

We applied this HPE algorithm to our data, which left us with a skeleton representation containing x- and y-coordinates, scaled to be between 0 and 1, for each of the 33 body joints for every frame in every clip. The algorithm uses a detection certainty threshold for each point, to determine if the pose estimation is accurate. If the estimation is deemed inaccurate, zero was given as a value for each key point. In figure 1 , we show four frames after applying the HPE algorithm to it and plotting the skeleton on top of it.

3.1.3 Pre-processing

Before feeding our skeleton data to our two machine learning models, we had to clean the data. After joining our annotations with the skeleton data, we removed all frames the pose estimation deemed inaccurate, which were 1871 frames. These frames carried no information about the skeleton and would only function as noise to the model. Likewise, frames that were labeled "Unlabeled" were removed since they didn't carry any salient information about the Skill and it would create another layer of complexity by adding a new label. This left us with a total of 17525 frames.

[1] https://google.github.io/mediapipe/solutions/pose.html

3.2 MACHINE LEARNING MODELS

In the following section, we explain the two models we use in our thesis, as well as the architectures we implement.

3.2.1 CNN

Convolutional Neural Network is a variant of a neural network that is particularly good at image recognition. This is highly due to its ability to recognize local patterns in subsequent parts of the image. A CNN uses **convoutional layers** to mask the input image into different representations. Each convolutional layer applies a filter with randomized values that maps the original features into new features. That way, each filter will be trained to detect different basic structures such as "lines" or "circles". First a input images is put through a convolutional layer. The output of this layer, $G[m, n]$ called a convolution, can be calculated as the dot product between the filter and the window the filter is applied on.

$$G[m, n] = \sum_j \sum_k h[j, k] f[m - j, n - k] \tag{1}$$

where f is a matrix describing the input image, h is the filter, n and m are indexes of resulting matrix, j and k are the indexes of the input matrix.

Then an activation function is applied to the convolution to increase non-linearity. After which, a **pooling layer** is applied to further reduce the convolution into a smaller dimension. Two commonly pooling methods are max- and average-pooling. Max-pooling returns the maximum value in a given matrix, whereas average-pooling returns the average.
This process can be repeated N times depending on how many convolutional layers there exist in the CNN. The more convolutional layers, the deeper the network is, and the filters will be trained to detect more complex structures. Lastly, the output is sent to a **fully connected layer** that will apply weights, where after an output layer applies an activation function to predict an output. From this output, a loss is calculated, and weights and filters are updated using backpropagation. A set of hyperparameters can be set including *stride*, *padding* and *kernel size*. Stride is the step size of the filters, padding is how much padding is wrapped around the images, and kernel size is the dimension of the filters.

additionally, the output dimension can be calculated as follows:

$$n_{out} = \frac{n_{in} + 2p - f}{s} + 1 \tag{2}$$

where p is the padding, f is the filter dimension and s is stride.

3.2.1.1 Architecture of the CNN

For our thesis, we designed a CNN to input skeleton data as pseudo images. The dimension of the input data is (2x33) since there are 33 body joints each with x- and y-coordinates. We designed the CNN with 2 convolution layers and 2 max-pooling filters. Since our data structure has the dimension (2x33), the dimension of our convolutional- and pooling filters were limited to (2x2). However, the ambition is that it could learn different local structures along one dimension capturing relations between body parts. In both convolutional layers, we set padding to 1. Additionally, stride was set to 1 on the first convolutional layer and 2 on the second, as this showed better performance. For activation functions, we used RELU, as loss function we used NNLOSS, and a final Softmax layer was added since we are doing multi-class classification. As our optimizer, we use Stochastic Gradient Descent.

3.2.2 Long-Short Term Memory

Recurrent neural networks, are superior to a normal forward neural network when it comes to sequential or time-series data. This is because RNN retains information from previous input and uses this to create relationships over time. For this thesis, we use an LSTM as one of our classifiers to capture temporal dependencies between frames in a video. An LSTM consists of multiple coherent LSTM-cells, each using different gates to remember, forget and update information coming from the previous cell and send it to the next one. Each cell gate gets at time t, a *cell state* from the previous cell, C_{t-1}, a *hidden state* from previous cell, h_{t-1}, and an input x_t. One LSTM cell operates with 3 different gates, namely a *forget gate*, an *input gate* and an *output gate*.

Forget Gate decides what information needs to be forgotten in the cell state. This is done by passing the information from the previous hidden state and the input through a Sigmoid-function along with an associated weight W_f and an associated bias b_f:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3}$$

The output of this is a matrix, f_t, with numbers between 0 and 1 for each number in C_{t-1}. The closer to 0 a number is the less important it is, and the closer to 1 the more it should be "remembered". f_t is then multiplied with C_{t-1}, filtering C_{t-1} according to the values found in

f_t.

Next step is the **Input Gate**. This gate decides what new information should be stored in the cell state. This consists of two steps: First a Sigmoid.layer i_t, determines what is to be forgotten, and secondly a *tanh layer* $\widetilde{C}_t$, determines what values might be added to the cell state:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{4}$$

$$\widetilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{5}$$

The outputs of these two layers are multiplied together and added to the cell state, updating it once again.

The final equation for the cell state at time t is:

$$C_t = C_{t-1} * f_t + i_t * \widetilde{C}_t \tag{6}$$

C_t is the cell state which the subsequent LSTM-cell will receive.

Lastly, the **Output** h_t, of the cell needs to be determined (The output also acts as the hidden state given to the next LSTM-cell). This is determined by C_t, as well as a Sigmoid layer acting as a filter. The filter, o_t, will be running h_{t-1} and x_t through a Sigmoid layer:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{7}$$

while C_t will go through a tanh layer. The output will be calculated as follows:

$$h_t = o_t * tanh(C_t) \tag{8}$$

From the output of each LSTM-cell as loss is calculated and weights are updated using stochastic gradient descent and backpropagation.

3.2.2.1 Architecture of the LSTM

For this thesis, we used a bi-directional, stacked LSTM consisting of two bi-directional LSTMs, with a Softmax output layer. The reason for using bi-directional instead of the regular forward LSTM, is to better capture the potential temporal aspects of our data, which we believe are present. Similarly, we believe a deeper network would capture these temporal aspects better, which is why we use a stacked LSTM. The downside of such a network, is the possible computational time, but due to our relative small amount of data, as well as no

considerable time constraint, we decided on this trade-off. We chose a Softmax output layer, as we are doing multi-class classification.

3.2.3 Loss and Optimizer Functions

Softmax outputs a probability distribution between the different classes, and is calculated as follows:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{9}$$

Where z_i is a value from the vector z.
Since we used a Softmax output layer for both of our machine learning models, we chose to use the negative-log likelihood loss function(NLL):

$$L(y) = -\ln(y) \tag{10}$$

where y is the probability of the correct classification.
For optimizing our models, we chose to use Stochastic Gradient Descent (SGD). Gradient Descent aims to find a set of input variables that minimize a function, such as our loss function, by computing the gradient of the input vector. To decrease computational time, SGD uses an approximation of the gradient, based on batches of the data. The gradient is calculated for each weight in our models, in a backward order starting at the output. This method is called back propagation. By using the chain rule from calculus, we make a "chain" of sub-functions, and calculate the gradients recursively backward throughout the model, while updating the weight along the way to minimize the loss.

3.2.4 K-fold Cross Validation

K-fold cross validation is a non-exhaustive cross validation technique, that iterates k times over a data set and splits said data set into k folds for each iteration. Each of these iterations will have a unique combination of train and test data. A common split distribution of the data is 70-80% of the data as training data and 20-30% of the data as testing data.

Using K-fold cross validation gives a better and more robust estimate of the performance of a model, as the model is evaluated by several combinations of train and test set. Furthermore, it also reduces the chance of overfitting a model.

4 EXPERIMENTS

4.1 K-FOLD CROSS VALIDATION

To conduct our experiments, we use a nested K-fold cross validation technique to achieve the most consistent results, with the lowest amount of noise. The structure of our nested K-fold is as follows:
A standard 5-fold cross validation algorithm splits our data into 5 equally sized folds, with around 80% of the data being used for training and 20% used for testing. This continues for 5 iterations, with a new combination of folds forming the train and test set in each iteration. We call these five different combinations $[parent_1, ..., parent_k]$. For each parent, the training set is furthermore split by a 4-fold cross validation method, into 4 equally sized folds, with a 75/25 train and validation split. This leaves us with 4 combinations of validation and train data for each parent, which we call $[child_{k1}, ..., child_{k4}]$. In total this leaves us with 5 parents and 20 children as seen in the figure 1, therefore the term nested K-fold. Additionally, before running our nested cross validation algorithm, we shuffled the data by clips, i.e frames would not be shuffled to keep the temporal aspect, but the order of clips would be. This was done to ensure a more equal distribution of clips, and thereby reduce the possibility of overfitting.

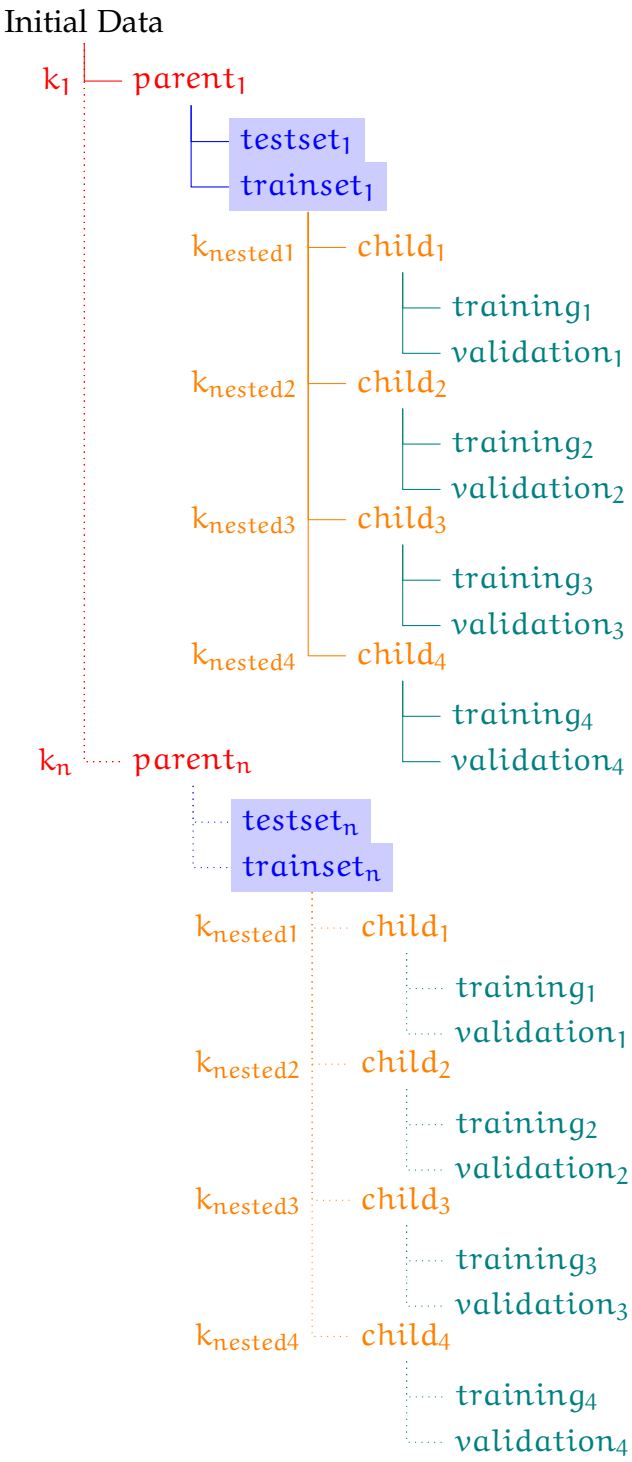
4.2 EXPERIMENT PIPELINE

In the following section, we will walk the reader through how we conducted our experiments. First, we will briefly explain how we tune our hyperparameter. Then we will explain how we deploy different variations of our models.

4.2.1 Hyperparameter Tuning

For hyperparameter tuning we did a manual rough estimation of parameters, and then an in-depth analysis using the state of the art hyperparameter tuner Optuna [1]

Figure 1: Structural view of our data, after the k-fold algorithm has been used on our initial dataset



4.2.1.1 Manual Tuning

For both CNN and LSTM, we evaluated both the SGD and the ADAM optimizer. We found the performance to be somewhat similar, and since ADAM automatically tunes learning rate and momentum, we chose to use SGD. This enables us to create an *averaged model* using the average of hyperparameters found with SGD, and compare that with base models and ensemble models.

In the CNN architecture, we experimented with a different number of filters and found the best performance with 8 filters in the first convolutional layer and 64 in the second. Furthermore, we tried different values of batch size and found the best performance with a batch size of 64.
For the LSTM, we experimented with a different number of layers but arrived at 2 achieving the best results.

4.2.1.2 Optuna and Tuning Analysis

We use Optuna on a range of tests for tuning the learning rate and momentum for the SGD optimizer in both CNN and LSTM. Optuna is a hyperparameter tuning tool, that offers fast parallel searching and pruning strategies. For each model we create 15 tests, meaning we test 15 combinations of hyperparameters in the range [0.01 ... 0.1] for learning rate and [0.1 .. 0.9] for momentum, and the model in each test used 15 epochs. The parameters were evaluated on the validation set in each *child*, using F1-score as the deciding metric.

4.2.2 Deploying the Models

Base Models. For each **child** we deploy an LSTM and a CNN, trained and hyper-parameter tuned on its respective train and validation data. After finding the hyperparameters that maximize the F1-score on the validation set, we evaluate the models on the test set found in the *parent*. The result of this is 20 LSTM and 20 CNN base models.

Ensemble Models. For each *parent* we create two ensembles of base models based on its four *children*. One, CNN_E consists only of CNN base models, while the other, LSTM_E, consists only of LSTM base models. For the LSTM base models, we trained them using 100 epochs, while the CNN base models used 150 epochs. The ensembles make use of a soft vote that combines the probability of each prediction and then picks the class with the largest probability. The ensembles are evaluated on the test set from the corresponding *parent*, The end result is five LSTM_E and five CNN_E

Averaged Models. Lastly, we create the averaged models, $\overline{\text{CNN}}$ and

$\overline{\text{LSTM}}$. For each *parent* we use the average learning rate and momentum of its four *children* to create a set of new hyperparameters, which we use to train an averaged model on the train set of the *parent*. After training, the model is evaluated on the test set of its corresponding *parent*. We end up with five $\overline{\text{CNN}}$ models and five $\overline{\text{LSTM}}$ models.

Table 1 Results

K	Model	LSTM		CNN	
		Accuracy(%)	F1-score(%)	Accuracy(%)	F1-score(%)
1	Base 1	**85.6**	**83.5**	64.4	63.6
	Base 2	79.1	76.8	63.2	62.0
	Base 3	79.8	77.9	61.3	61.4
	Base 4	77.3	75.8	65.9	65.8
	Ensemble	83.6	81.5	69.0	69.8
	$\overline{\text{Model}}$	77.9	76.6	**71.8**	**71.8**
2	Base 1	81.9	78.9	73.3	69.8
	Base 2	80.7	78.2	74.3	69.3
	Base 3	80.5	78.8	71.8	69.8
	Base 4	79.6	77.1	74.4	71.9
	Ensemble	**85.2**	**83.0**	**77.6**	**74.3**
	$\overline{\text{Model}}$	84.9	82.3	75.4	74.2
3	Base 1	35.6	19.3	66.2	65.0
	Base 2	83.3	81.9	65.2	63.1
	Base 3	82.4	80.8	64.6	59.6
	Base 4	81.1	79.1	63.3	60.9
	Ensemble	**84.0**	**82.1**	**70.0**	**67.6**
	$\overline{\text{Model}}$	80.1	77.2	65.2	64.5
4	Base 1	83.5	81.0	68.5	67.3
	Base 2	85.5	82.6	69.4	65.9
	Base 3	80.2	76.3	73.1	69.7
	Base 4	80.3	77.1	68.8	65.4
	Ensemble	**87.3**	**84.8**	**73.1**	**70.1**
	$\overline{\text{Model}}$	85.6	83.0	68.2	65.8
5	Base 1	81.1	78.2	68.2	65.7
	Base 2	80.8	77.8	65.2	63.6
	Base 3	81.9	78.7	71.5	**70.0**
	Base 4	82.7	79.5	69.0	63.0
	Ensemble	**83.0**	**80.1**	**72.6**	69.8
	$\overline{\text{Model}}$	80.3	76.5	70.7	66.4

Table 2 Comparison of Ensembles and averaged models for both architectures

Model	Accuracy		F1-score	
	μ	σ	μ	σ
CNN_E	72.46 %	3.0	70.32 %	2.18
$\overline{CNN}$	70.26 %	3.43	68.54 %	3.77
$LSTM_E$	84.62 %	1.52	82.3 %	1.57
$\overline{LSTM}$	81.76 %	2.98	79.12 %	2.90

4.3 RESULTS

Our results are shown in table 1, where we compare base models, ensembles, and averaged models in the 5 folds. Table 2 shows the average performance of the ensembles and averaged models, as well as their standard deviation. Furthermore, we show the average confusions matrices of our ensembles and average models in 2. In the following section, we'll briefly lay out our results.

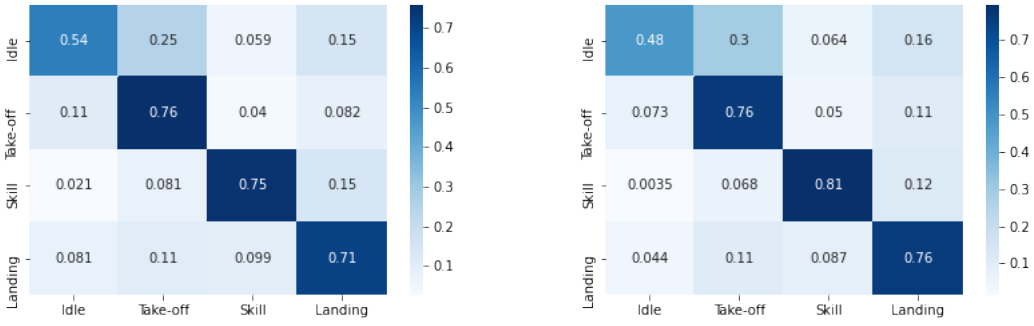
4.3.1 CNN

Table 1 show that generally the ensemble is the best performing classifier, except in $K = 1$ where $\overline{Model}$ is best. We see that $\overline{CNN}$ performs decent, with accuracy and F1-score generally a bit above the average performance of the base models.
In table 2 we see the average F1-score and accuracy of the CNN_E is 70.32% and 72.46% with 2.18 and 3.0 standard deviation respectively, besting the $\overline{CNN}$ in both performance and on how accurate it is. Furthermore, for the CNN_E the standard deviation of its F1-score is lower than of its accuracy.
In fig. 2 we see both CNN models generally having a hard time predicting 'idle', often confusing it for take-off, 25% and 30% percent of the time respectively. The CNN_E is best at prediction skill, with an average accuracy of 81%, while the $\overline{CNN}$ is best at predicting take-off with an average accuracy of 76%.

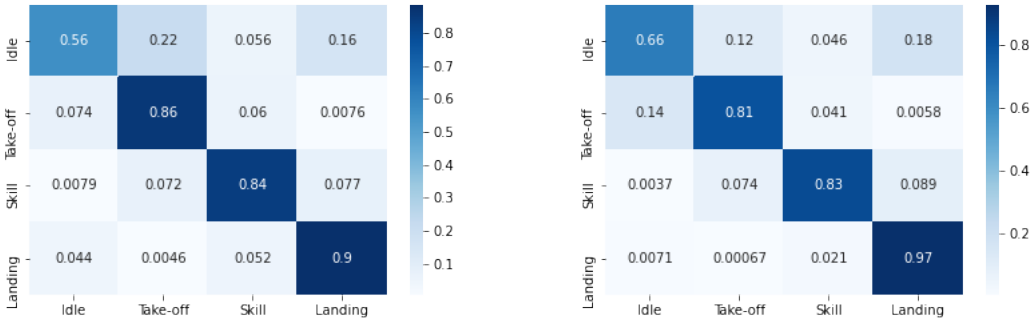
4.3.2 LSTM

From table 1 we see that the $LSTM_E$ generally outperforms all the other models. However, In $K = 1$ we see that Base 1 performs the best, with an accuracy of 85.6% and an F1-score of 83.5%, beating the ensemble model for that fold by 2% in each category. We also observe

the $\overline{\text{LSTM}}$ performing similar or worse than the base models in most cases, despite being trained on more data. Interestingly, Base 1 in $K = 3$ is by far the worst performer with an accuracy of 35.6% and an F1-score of 19.3%. When looking at the loss function for this particular model in figure 2 in the Appendix, we see that it ends its training on a spike in the loss function. Generally, the loss function for the LSTM has many spikes, and it seems the model can be "unlucky" and end training on a spike, which would result in bad predictions.
In table 2 we observe that our two LSTM models generally outperform their CNN counterparts, by around 10-12% in both accuracy and F1-score. We also observe that the standard deviation is significantly smaller for the LSTM classifiers. Additionally, we see that LSTM_E beats the $\overline{\text{LSTM}}$ and is the best performing model with an average F1-score of 82.3% and a standard deviation of 1.57
In fig.2 we see that similar to the CNN, both LSTMs still have difficulties predicting "Idle". $\overline{\text{LSTM}}$ confuses idle mostly with take-off, 22% of the times, while LSTM_E confuses it more with landing, a value of 18%. Both LSTMs perform extremely well on predicting landing with an average accuracy of 97% and 90% respectively but also perform well on take-off and skill.



(a) $\overline{\text{CNN}}$ **(b)** CNN Ensemble



(c) $\overline{\text{LSTM}}$ **(d)** LSTM Ensemble

Figure 2: Confusion matrices over the averaged results of the respective CNN & LSTM models.

5 CONCLUSION

5.1 DISCUSSION OF RESULTS

5.1.1 Ensemble Models vs Averaged Models

From the confusion matrix and the tables presented in chapter 4 we can see that the difference between the $\overline{\text{CNN}}$ and $\overline{\text{LSTM}}$ and their counterparts, the CNN_E and the LSTM_E, are quite significant. When looking at table 2 we can see the standard deviation σ, in both accuracy and f1-score, being almost twice as large for the $\overline{\text{LSTM}}$ when compared to the LSTM_E. This in turn means the $\overline{\text{LSTM}}$ model, on average, is more unstable when it comes to predicting the correct label. While not as prevalent, the same trend can be seen when comparing the $\overline{\text{CNN}}$ to the CNN_E, both of which perform worse on average when compared to the $\overline{\text{LSTM}}$ and LSTM ensemble. From table 1 we can see a direct comparison between all the models in each iteration of K and a clear trend is seen for the LSTM models. The different LSTM_E always outperform the $\overline{\text{LSTM}}$s in the respective K. The same is also true for the CNN ensemble when compared to $\overline{\text{CNN}}$ in all iterations of K, except for K = 1.
Since the best hyperparameters of the different base models can be far from each other, an average is not guaranteed to perform well.
With this in mind, we chose to focus more on the ensemble models in our discussion, due to the general increase in performance. This is also reflected in table 2, where the ensemble for the CNN and the LSTM outperforms their respective counterpart.

5.1.2 LSTM Ensemble vs CNN Ensemble

Looking at table 1 and 2, it is evident that the LSTM_E outperforms the CNN_E in the task of frame-by-frame sub-action classification. We hypothesize the reason for this to be due to the LSTM considering temporal aspects, whereas the CNN does not. The LSTM takes a whole clip as input, hence capturing any possible temporal aspects of the sequence, whereas a CNN takes only one frame at a time as input. This means that our CNN can only classify based on the body position in that exact frame, disregarding whatever came right before or what comes after, while our LSTM takes this temporal aspect into account.

This difference is also seen in 2, where we see actions with similar body poses, such as Idle, Take-off, and Landing, being mistaken for each other. This is especially noticeable for our CNN, where we see that it predicts Landing and Take-off on 46% of the Idle frames. The LSTM predicts Take-off on 12% of the Idle frames, which is significantly less than the CNN, which predicts Take-off on 30% of the Idle frames. We believe this discrepancy appears because the LSTM knows the temporal aspect of Idle often appearing before Take-off. When it comes to predictions on frames labeled Take-off, the CNN predicts Landing on 11% of the frames and Idle on 7.3%, whereas the LSTM rarely predicts Landing, but predicts Idle more often than the CNN. We suppose this to be a result of the temporal aspects captured by the LSTM. It knows that Landing would not come after Take-off, so even though the body pose of Take-off and Landing might look similar, it would rarely predict Landing on Take-off frames.
We see that both the CNN and LSTM perform well on frames labeled Skill, 81% and 83% accuracy respectively, suggesting that Skill has a unique body pose. This is in line with what we expected, as the body pose when doing a Skill would often be more curled up and positioned differently, with the action points being in more unique positions when compared to the poses of the other labels, such as being upside-down. We hypothesize the unique body pose of the Skill label is the deciding factor for the high accuracy, 97%, of the LSTM when predicting the Landing frames. As previously mentioned, the body pose of Landing is not particularly unique, but because Skill is, and the LSTM has learned Landing comes after Skill, our LSTM can differentiate between Landing, Idle, and Take-off on Landing frames.
One should note the loss for the different base models of the CNN ensemble, in 1 in the appendix, has yet to converge to a stationary point. This is not the case with our LSTM, where the loss seems to be closer to convergence. This suggests that perhaps running the CNN with more epochs would yield a lower loss and a more accurate model. Even though increasing the number of epochs could yield better results for the CNN, we still believe the LSTM would reign supreme in this task since it captures temporal aspects.

5.1.3 Data

Consequence of Labelling Choice While the LSTM achieves higher performance than the CNN, its accuracy in the classification of Idle still lacks behind when compared to the other three labels. We presume this has more to do with the choices in our labeling strategy, rather than a shortcoming of the model itself.
When labeling our video data, we discovered that in some cases the subjects would be hesitant to commit to a Skill and therefore they

would perform a Take-off motion, but not follow up said motion by performing a Skill, and would instead return to an Idle state.
We made an active choice in our labeling process, such that there could only be one single instance of Take-off present in a clip since if there was no skill as a follow-up, it would technically not be a Take-off. This resulted in some videos consisting of multiple fake Take-offs, where we changed the label of the action to Idle.
We believe this is reflected in our findings in figure 2 where our models struggle to separate Take-off from Idle, when the true label is Idle. This is especially apparent for the CNN_E, where the model could be presented with two almost identical frames, but labeled differently depending on what comes after.

5.2 LIMITATIONS

In this section, we will describe some of the limitations of our work, mainly limitations in of the chosen models and limitations of our proposed data set.

5.2.1 Models

5.2.1.1 CNN

The different models mentioned in sec.2.3.1 are all different variations of a vanilla CNN with alterations and additions to the model and the input data. These different alterations made the model more receptive to sequences as input, instead of just one single instance. Since we implemented a relatively simple CNN, we could only input one frame at a time. This has the downside of not being able to consider any temporal aspects when making predictions on the data. Furthermore, a vanilla CNN normally takes pictures as input, making use of the different convolutional and pooling layers to extract and make information denser. We gave the skeleton position of one frame, containing x and y coordinates for 33 key points, as input to our CNN, resulting in it interpreting this information as a "pseudo-picture" of the size 2x33. We suspect that the CNN wouldn't be able to utilize its different layers optimally when presented with such a small "picture".

5.2.1.2 LSTM

The vanilla LSTM seems to perform well when handling the task of frame-by-frame sub-action classification since it can look at a whole sequence and capture temporal aspects. Although our results show that temporal aspects are important for frame-by-frame classification

of sub-tasks, an LSTM might still be more complex than necessary, since it excels in capturing longer relationships. Looking at the data, we don't have many intricate and long term-relationships between frames, since we have such a small "vocabulary" of classes. The relationships we have is that a sequence of Take-offs is always followed by a sequence of Skills, which is always in between a sequence of Take-offs and a sequence of Landings. A sequence of Landings always appears after a sequence of Skills. None of these relationships is long-term as they always appear subsequently. If we instead had a bigger set of sub-actions, such as *take-off type*, *skill type* etc., there could be more long-term relationships. Many of the proposed variation of LSTM in sec.2.3.2 is trained and tested on data, which is, first of all, labeled on action-level, but also has 100+ labels. In such data, we would assume that small and intricate relationships, across the whole sequence, would be a determining factor for some classes.

5.2.2 Data

The data available for sub-action classification was very limited and it forced us to gather and annotate our own, which was a long and tedious task. This presented us with several limitations. As the process of finding and splitting videos into individual clips took a lot of time, we ended up with only around 200 videos. Due to us using deep learning models, which are data-hungry by nature, more data would most likely have resulted in better performance. Furthermore, we only used three people (the authors of this paper) to annotate the clips frame-by-frame, making the data prone to inconsistency in labeling. These inconsistencies would be more evident in the "border-regions", i.e where the video would change from one label sequence to another such as Skill -> Landing. Using more annotators could possibly have made these "border-regions" more consistent.
Moreover, we used homemade videos, where the quality and frame rate varied, despite us trying to choose videos with decent video quality. We suspect these variations were reflected in our pose estimations. Videos with a lower resolution would have less precise estimations, whereas videos with a higher resolution would have more precise estimations. We think this difference in quality affected the performance of the models.

Only having videos of consistently high quality would have resulted in more precise and consistent body pose estimations, which we assume would have had a positive impact on the models' performance and our findings.

5.3 CONCLUSION ON OUR EXPERIMENTS

In this thesis, we set out to explore how two deep learning models, CNN and LSTM, perform in the task of frame-by-frame sub-action classification of homemade gymnastics videos. Furthermore, we wanted to see how CNN ensembles and LSTM ensembles, compared to CNN's and LSTM's with averaged hyperparameters. With no other data set being publicly available to allow for the classification of sub-actions, we propose our own

This data set consists of 202 clips, of people doing somersaults, back-flips, and other similar flips, with each frame labeled one of four: Idle, Take-off, Skill, and Landing.
Our results show that a simple LSTM performs better than a simple CNN, both as ensembles, base models, and models with average hyperparameters when classifying sub-actions. We believe that this is due to the temporal aspect of the data, which the LSTM can capture, while the CNN is not. Moreover, for our subgoal, we found that using an ensemble generally performs better than each base model it is composed of, as well as models with averaged hyperparameters. Finally, we found that using models with averaged hyperparameters doesn't consistently perform better or worse than the base models.

5.4 FUTURE WORKS

As mentioned in the limitations section, our models were not tangent to the state-of-the-art models, which most likely impacted the results in a way where we did not maximize the possible scores, given our data set. For future works, implementing different state-of-the-art models, such as those mentioned in sec. 2.3.1 and sec. 2.3.2 and analyzing their performance in classifying sub-actions in our video data, would be a desired direction for further research. Especially the TCN would be interesting to implement since it encodes both the spatial and the temporal aspects, using ideas from LSTM implemented in a CNN. Another interesting model to use is the GCN. This seems to be suitable for our task since the skeleton data is seen as a graph-based structure.
Furthermore, going from single modality to multi-modality by incorporating modalities such as the **RGB modality** as input data for our models, and making use of methods such as *fusion* and *co-learning*, would also be an interesting direction for future analysis.

Given that the size of our data was relatively small and the quality of our scraped YouTube videos mediocre, future work could expand upon the scope of our data collection, as well as the labeling of said

data. Currently we have 4 classes (idle, take-off, skill, landing) and moving forwards one could expand the Skill label to contain the different types of skills, eg. a back flip or a front somersault, to make the data set more fine grained. The same could be done for the Idle label, which currently does not differentiate between actions such as running, walking or standing.

For future iterations, and the initial motivation behind this bachelors project, we would also have liked to have our data scored by professional judges according to the official rule book in TeamGym. With this data we would have liked to see if we could mimic a professional judge, by implementing a neural network that would be able to automatically score a gymnastics skill based on the video data in question.

BIBLIOGRAPHY

[1] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[2] V. Bazarevsky, I. Grishchenko, K. Raveendran, T. Zhu, F. Zhang, and M. Grundmann. BlazePose: On-device Real-time Body Pose tracking. *arXiv:2006.10204 [cs]*, June 2020. arXiv: 2006.10204.

[3] V. Bazarevsky, Y. Kartynnik, A. Vakunov, K. Raveendran, and M. Grundmann. BlazeFace: Sub-millisecond Neural Face Detection on Mobile GPUs. *arXiv:1907.05047 [cs]*, July 2019. arXiv: 1907.05047.

[4] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh. OpenPose: Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(1):172–186, Jan. 2021. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[5] Y. 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.

[6] H.-S. Fang, S. Xie, Y.-W. Tai, and C. Lu. RMPE: Regional Multi-person Pose Estimation. *arXiv:1612.00137 [cs]*, Feb. 2018. arXiv: 1612.00137.

[7] S. Ju, M. Black, and Y. Yacoob. Cardboard people: a parameterized model of articulated image motion. In *Proceedings of the Second International Conference on Automatic Face and Gesture Recognition*, pages 38–44, Killington, VT, USA, 1996. IEEE Comput. Soc. Press.

[8] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, 2014.

[9] T. S. Kim and A. Reiter. Interpretable 3D Human Action Analysis with Temporal Convolutional Networks. *arXiv:1704.04516 [cs]*, Apr. 2017. arXiv: 1704.04516.

[10] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager. Temporal Convolutional Networks for Action Segmentation and Detection. *arXiv:1611.05267 [cs]*, Nov. 2016. arXiv: 1611.05267.

[11] J. Liu, A. Shahroudy, M. Perez, G. Wang, L.-Y. Duan, and A. C. Kot. NTU RGB+D 120: A Large-Scale Benchmark for 3D Human Activity Understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(10):2684–2701, Oct. 2020. arXiv: 1905.04757 version: 2.

[12] J. Liu, G. Wang, L.-Y. Duan, K. Abdiyeva, and A. C. Kot. Skeleton-Based Human Action Recognition with Global Context-Aware Attention LSTM Networks. *IEEE Transactions on Image Processing*, 27(4):1586–1599, Apr. 2018. arXiv: 1707.05740.

[13] S. Liu, X. Liu, G. Huang, L. Feng, L. Hu, D. Jiang, A. Zhang, Y. Liu, and H. Qiao. FSD-10: A Dataset for Competitive Sports Content Analysis. *arXiv:2002.03312 [cs]*, Feb. 2020. arXiv: 2002.03312.

[14] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black. SMPL: a skinned multi-person linear model. *ACM Transactions on Graphics*, 34(6):1–16, Nov. 2015.

[15] A. Nagpal. Fine grained action recognition in sports videos. page 6.

[16] D. Shao, Y. Zhao, B. Dai, and D. Lin. FineGym: A Hierarchical Video Dataset for Fine-grained Action Understanding. *arXiv:2004.06704 [cs]*, Apr. 2020. arXiv: 2004.06704.

[17] K. Soomro and A. R. Zamir. Action recognition in realistic sports videos. In T. B. Moeslund, G. Thomas, and A. Hilton, editors, *Computer Vision in Sports*, pages 181–208. Springer International Publishing. Series Title: Advances in Computer Vision and Pattern Recognition.

[18] Z. Sun, Q. Ke, H. Rahmani, M. Bennamoun, G. Wang, and J. Liu. Human action recognition from various data modalities: A review.

[19] Z. Sun, J. Liu, Q. Ke, H. Rahmani, M. Bennamoun, and G. Wang. Human action recognition from various data modalities: A review. *CoRR*, abs/2012.11866, 2020.

[20] T. Tsunoda, Y. Komori, M. Matsugu, and T. Harada. Football Action Recognition Using Hierarchical LSTM. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 155–163, Honolulu, HI, USA, July 2017. IEEE.

[21] V. Veeriah, N. Zhuang, and G.-J. Qi. Differential Recurrent Neural Networks for Action Recognition. *arXiv:1504.06678 [cs]*, Apr. 2015. arXiv: 1504.06678.

[22] P. Wang, Z. Li, Y. Hou, and W. Li. Action recognition based on joint trajectory maps using convolutional neural networks. In *Proceedings of the 24th ACM international conference on Multimedia*, pages 102–106. ACM.

[23] S. Zhang, X. Liu, and J. Xiao. On Geometric Features for Skeleton-Based Action Recognition Using Multilayer LSTM Networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 148–157, Santa Rosa, CA, USA, Mar. 2017. IEEE.

A APPENDIX

A.1 GITHUB

Link to Github Enterprise:
https://github.itu.dk/guba/BACHELOR-ITU-2022

Link to Github:
https://github.com/flexfalk/visionbased_human_motion_analysis_in_gymnastics.git

A.2 LOSS OVER ITERATIONS ON TRAINING

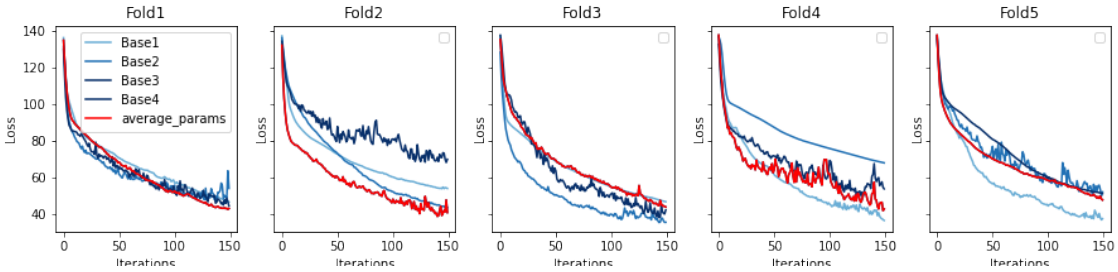


Figure 1: Loss for CNN on training data

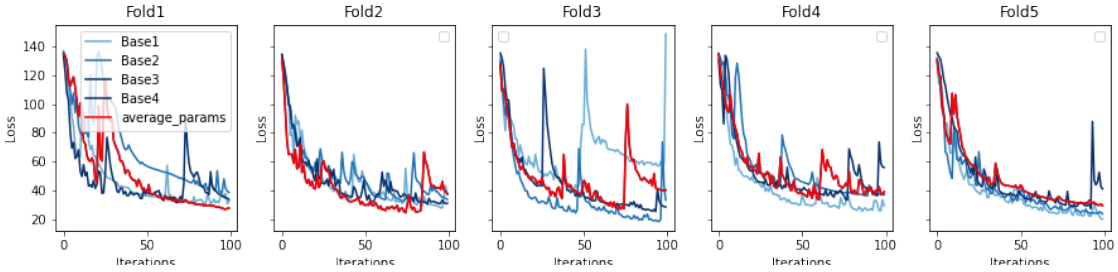


Figure 2: Loss for LSTM's on training data