STUTTGART MEDIA UNIVERSITY

MASTER THESIS

Applied Research of an End-to-End Human Keypoint Detection Network with Figure Ice Skating as Application Scope

---

Author:
Nadin-Katrin APEL

Supervisor:
Prof. Dr. J. MAUCHER
Prof. Dr. S. RADICKE

A thesis submitted in fulfillment of the requirements
for the degree

Master of Science

Computer Science and Media

June 2, 2020

Declaration of Authorship

I, Nadin-Katrin APEL, declare that this thesis titled, "Applied Research of an End-to-End Human Keypoint Detection Network with Figure Ice Skating as Application Scope" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at
- this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such
- quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was
- done by others and what I have contributed myself.

Signed:

_____

Date:

_____

"Data is a precious thing and will last longer than the systems themselves."

Tim Berners-Lee

STUTTGART MEDIA UNIVERSITY

Abstract

Computer Science and Media

Master of Science

Applied Research of an End-to-End Human Keypoint Detection Network with Figure Ice Skating as Application Scope

by Nadin-Katrin APEL

Human joint detection is a key component for machines to understand human actions and behaviors. Especially in figure ice skating this understanding is an indispensability, where there are many difficult figures and poses, even difficult to clearly understand for the professionalized jury. Herewith we present an end-to-end approach to detect the 2D poses of a person in images and videos. In the architecture we combine three branches: Image Segmentation, Body Part Detection, and Human joint detection. The applied research reveals multiple findings which outperform current existing main players with the special application scope of figure ice skating.

Acknowledgements

The acknowledgments and the people to thank go here, don't forget to include your project advisor

Contents

List of Figures

List of Tables

List of Abbreviations

LAH	**L**ist **A**bbreviations **H**ere
WSF	**W**hat (it) **S**tands **F**or

Physical Constants

use it Speed of Light $c_0 = 2.997\,924\,58 \times 10^8\,\mathrm{m\,s^{-1}}$ (exact)

List of Symbols

a	distance	m
P	power	W ($\mathrm{J\,s^{-1}}$)
ω	angular frequency	rad

For/Dedicated to/To my...

Chapter 1

Introduction

Human 2D pose estimation has gained more and more attraction in recent years. For example Facebook, one of the BigFive technology companies, has published 73 research paper targeting the problem of pose estimation in the last three years. The most popular ones are DensePose, VideoPose3d or Mask R-CNN [2, 16, 20, 34]. A company in Canada *wrnch.ai* even specialized on keypoint recognition from image and video data with a lot of product options [59]. Furthermore, many enterprises are becoming more and more interested in Sport Content Analysis (SPA) e.g. Bloomberg, SAP and Panasonic, just naming a few [14, 42].

But how got this topic into such a demanded focal point? Probably this is due to the various application areas in which Pose estimation can be encountered. Main fields are sports, visual surveillance, autonomous driving, entertainment, health care and robotics [33, 47, 64]. For example Vaak, a japanese startup, developed a software, which would detect shoplifters, even before they were able to remove items from a store. This yielded in a drastic reduction of stealing crimes in stores.

The exercise of sport not via visiting a sports course, gym or club became of fundamental severance in 2020, when the Coronavirus SARS-CoV-2 spread the entire world [39]. Many courses such as Yoga, Pilates or general fitness routines went online and were often conducted via Zoom, Instagram live or other video streaming technologies [3]. However, what participants were often missing, was the feedback of the coach on how the exercise was going, and whether it was done right or wrong. So in 2020 more than ever was missed a technology which is good at pose estimation, or even further, action recognition, in sports.

Most investigations in this field target usual activities not including complex poses which can be encountered in professional sport. This is why these architectures often fail when applied to more complex movements. For competitive sports there are various metrics of high interest depending on the environment. Competition and training can be differentiated as can be sports executed by multiple athletes versus single combats. Basketball or soccer as team sports for example are interested on predictions about how the other team behaves during the game and which would be the best reaction to their behaviour for winning the game. During practice 2d pose recognition can help to optimize the sports-person movements by taking the role of a coach. This could provide an answer to the question on how certain activities might be optimized? Single competitive sports with very complex movement routines are for example gymnastics and figure ice skating. Both sports include various artistic body movements, which are not part of daily activities. Even famous and well rated 2d pose recognition networks such as OpenPose or VideoPose3d fail to recognize these poses.

If this problem was solved it could help with action recognition and support during practice or relieve the jury on competitions. A predictor could for example suggest, what an athlete should do to land a certain jump. On the other hand jury is rare and the job sitting all day in the ice-rink on weekends with only a very small salary is not very attractive. Furthermore, people often complain scoring is not executed fairly.

Especially in figure ice skating an accurate 2d pose recognition could make a huge contribution. This is why this paper investigates 2d pose recognition with special focus on figure ice skating.

1.1 Motivation and Goals

A working 2d pose estimator could make a huge contribution to figure ice skating. Especially when building an action recognizer on top of it. However, as of today, this was not possible yet due to the complex poses and the different gliding movements on the ice. Especially spins with their fast rotation and stretching poses are of high complexity to these estimators. Such an estimator could support fair scoring during competitions or help to improve motion sequences during practice.

With the downward trend of jury staff and the increasing demand for more small competitions, jury is asked more and more in figure ice skating. Particularly the role of the technical specialist or controller diagnosing the individual elements on the ice is of high demand. Some competitions were event canceled in recent years, because they were not able to find the according jury. Furthermore, sitting all day in the cold ice rink for only a very low salary in not attractive at all. These long demanding days challenge concentration and many competition participants often complain about jury not rating fairly enough, completely forgetting the demanding work the jury has to do. Here a 2d pose estimator could contribute by recognizing the different elements or even scoring. This would not only relieve the jury but also could increase fairness.

During practice 2d pose estimators could examine the specific motions during elements and give hints how to improve these. Probably they could even suggest certain exercises to learn an element like a spin, jump or certain step. Additionally they could keep track of training and provide analysis metrics to the skaters and coaches.

All in all 2d pose estimation is very interesting not only because of all the possible different appliance possibilities in this sport, but as well because of the challenging task to build an according estimator, which was not possible until today.

1.2 Related Work

2D Pose estimation sets the baseline for machines to understand actions. It is the problem of localizing human joints or keypoints in images and videos. Many research studies explored and researched this topic already with the most popular ones being OpenPose and VideoPose3d [5, 34].

For 2D pose recognition there are mainly two general procedures: either top-down or bottom-up. Top-down first detects a person and then finds their keypoints. Whereas bottom-up first detects all keypoints in the image and then refers the corresponding people. For top-down it is argued, that if a person is not detected via a bounding box

or alike, no keypoints can be found. This would lead to more undetected keypoints in the frames of a video. When there are many people in the image with many occlusions the people often can not be detected. However, when a person is correctly detected, it is said that accuracy would be higher [32].

A famous top-down approach for example is Mask R-CNN developed by the Facebook AI Research team. It consists of three branches and two stages. The first stage presents the Region Proposal Network (RPN). It proposes candidate bounding boxes for objects. The second stage performs classification and bounding box regression by extracting features using region of interest pooling, which they refer as RoiPool. Additionally, Mask R-CNN predicts a binary mask for each ROI in the second stage. They receive top results in the COCO challenges for instance segmentation, bounding box object detection, and person keypoint detection [16]. Other famous top-down approaches include *Simple Baselines*, *the Cascaded Pyramid Network* or *Deep High-Resolution Learning* [8, 48, 60].

One of the most discussed and popular bottom-up approaches as of today is OpenPose. Their net predicts vector fields for the joint connections, which they call part affinity fields. Additionally, it estimates candidate keypoint locations via gaussian distributions. These they refer as part confidence maps. From these detections they refer the associate human poses. OpenPose shows very good results on the MPII and COCO challenges. They highlight their performance, which especially shows it's strength when detecting multiple people. The performance wouldn't change even if more and more people enter the scene. With the according hardware this would even show decent results in realtime [5]. In a newer research, they additionally pay attention to the temporal characteristic in video sequences in their work of Spatio Temporal Fields. Their approach is able to track multiple people's poses across frames being runtime-invariant to the number of people in the frames. Furthermore, they receive highly competitive results on the PoseTrack challenges [37]. Some other famous bottom-up approaches include *Convolutional Pose Machines* and *PifPaf* [23, 58].

The COCO, MPII and PoseTrack challenges lead to several studies in the pose estimation field. However, their dataset targets rather simple daily activities. There

have been only conducted a few studies on competetive sports such as basketball, ice-hockey or swimming [13, 32, 56]. For sports including full body flexibility or special unconventional jump or turn rotations as can be found in dance, gymnastics or figure ice skating, there have been only a few studies [6, 24, 62, 63]. These studies encounter three main problems in the figure skating domain: The first is domain knowledge. In C. Xu et Al. research they try to predict the technical and performance scores from video data with the only the ice skating progamm as video input and the judge scores as labels [62]. This will very likely not result in useful results since first, the figure skating judging system adjusts every year, second, there are always different judges on the competitions who all have their own rating style, and third however the skater is one of the always winning ones, this skater can do a good program falling at the main elements and still score very well, because the jury knows this skater and is just human in their judgement. This is one of the main controversies in the figure ice skating fairness of program judgement. Another problem is the missing dataset. There didn't exist a dataset with joint labeling until FSD-10 [24], which only came out at the beginning of the year. One very interesting study from Yu, Ri et Al. tries to create simulations from the figure ice skating elements. They exactly encountered the problem, that pose estimation currently does not work on difficult pose sequences such as spins or very flexible positions. However, they were able to successfully predict jumps and simple steps from videos into 3d simulation [60].

Goals

Our goal was to find a way on how to detect human poses in single skating, with an architecture which would even be possible to run on devices with lower computation power such as mobile phones. Important is, that only the main character in the image should be detected, and all background people would be neglected, so our network could later be used for action recognition and recommendation tasks during practice when there are multiple skaters on the ice.

Our Work

In our work we investigated the performance of VideoPose3d, OpenPose and wrnch.ai on figure ice skating elements. We elaborated the creation of a figure ice skating dataset with the help of the XSens motion capturing data in a real figure ice skating arena environment. Furthermore, we evaluated the creation of a dataset from these motion capturing data with the help of Blender and Makehuman. We created an end-to-end fully convolutional architecture consisting of three modules for background extraction, body part and keypoint detection. Since our work concentrates on pose estimation in single skating, the background extraction module is an indispensability. Multiple experiments with the network architectures, learning optimizers and loss functions resulted in decent results, which can be applied in figure ice-skating and run on usual hardware. At the end we elaborated several future thoughts were these studies can continue to.

Chapter 2

Figure Skating Pose Detection

2.1 Complexity of Figures

Figure ice skating includes very special movement sequences which stand out from other sports. Thanks to the surface of the ice, skaters perform gliding movements. Furthermore, their programs include highly artisitic movements with explosive take-offs, very high rotation speeds and flexible positions. Current pose estimators such as OpenPose, VideoPose3d or wrnch.ai all fail to correctly predict spins with their high rotation speed and difficult flexible positions. In this section we try to explain why especially figure ice skating means a challenge in human pose estimation.

In single figure ice skating, there are three main element types in a competitive program, which receive technical scores: jumps, steps, and spins. There are seven different listed jumps, which only differ in their take-off phase. Three of these are jumped just from one edge, the Axel, Loop and Salchow. The other ones additionally use the skate toe as a catapult. Furthermore, they can be distinguished by the edge that is last skated on the ice before the skater takes off. This is one of the reasons, why many skaters have problems with the Lutz and Flip jumps. For the jury it is often hard to tell as well, weather the jump should get an edge deduction. All the jumps can include a different amount of rotation in the air, whereas four rotations was the maximum a skater could perform until today. All these jumps can be combined in various manners, include features such as lifted arms or difficult steps before or after the element to increase the level of difficulty, resulting in higher scores.

For spins there are four main positions: upright, sit, camel and layback. These positions as well can be combined with various features, as for example jumps or difficult

elastic positions such as the famous Biellmann spin.

There are plenty of different step sequence elements including turns on the ice and steps. These as well can be combined with multiple additional features resulting in higher scores.

The above named elements only explain the absolute basics in figure ice skating. In practice scoring and the creation of ice skating programs is much more difficult. Nevertheless, this illustrates nicely, that however for the not professionalized audience many elements look the same, there are plenty of different metrics and elements. Moreover, each skater has their own style and performs these elements slightly different. This is what makes it so hard in machine-learning to correctly predict the elements. Which is why it is important that the basis, the keypoint recognition module, has to predict the poses correctly. Because otherwise the action recognition module has no chance to make correct estimations.

2.2 Distinct Rating System

- isu scores [53] - isu guidelines/ level of execution goe [52] - human struggle as well -> rating system with points, many abstractions, still often experienced as not fair

Sed ullamcorper quam eu nisl interdum at interdum enim egestas. Aliquam placerat justo sed lectus lobortis ut porta nisl porttitor. Vestibulum mi dolor, lacinia molestie gravida at, tempus vitae ligula. Donec eget quam sapien, in viverra eros. Donec pellentesque justo a massa fringilla non vestibulum metus vestibulum. Vestibulum in orci quis felis tempor lacinia. Vivamus ornare ultrices facilisis. Ut hendrerit volutpat vulputate. Morbi condimentum venenatis augue, id porta ipsum vulputate in. Curabitur luctus tempus justo. Vestibulum risus lectus, adipiscing nec condimentum quis, condimentum nec nisl. Aliquam dictum sagittis velit sed iaculis. Morbi tristique augue sit amet nulla pulvinar id facilisis ligula mollis. Nam elit libero, tincidunt ut aliquam at, molestie in quam. Aenean rhoncus vehicula hendrerit.

Chapter 3

Dataset

3.1 Figure Skating Dataset

The correct dataset with the according labels is what decides weather a neural network architecture will be able to sucessfully learn keypoints from image data or video sequence data. As of today, there does not exist a publicly available dataset which labels keypoints to ice skating performances or elements. Famous existing one such as FIS-V or the MIT datasets only include the total technical and performance scores of the complete performances, which are not applicable her as explained in chapter 2.

To create a reasonable dataset for figure ice skating there are various possibilities. Most straightforward would be to make use of the huge amout of vidos published everyday on online platforms such as YouTube and label the videos by hand. This however would cost a huge amount of time and includes outliers coming from human errors. Another possibility would be to work with motion capture.

3.1.1 Motion capturing techniques

There are three main methods for recording motion capturing data. Markerless capture is a method where videos are recorded via a single or multiple depth cameras. This comes with the advantage that the athletes are not distracted or feel uncomfortable by any markers [18, 25, 38]. When working with markers there are two sensoring types: optical or inertial. Optical markers are reflective markers, which are attached to characteristic part of the body. These reflections are tracked via multiple

cameras to make sure the motion movements are taken especially when a person is moving, so the markers are not concealed [55]. Inertial motion capture uses IMU sensors that are attached to body parts [31, 61]. 3D positions then are calculated via multiple sensor metrics. XSens system for example includes several trackers which contain Magnetometers, gyroscopes and accelerometers. Via their MVN Animate and Anlaze systems the tracked data is postprocessed on a biomechanical model and then can be used in game engines such as Unity or Unreal, or in 3D animation software such as Blender or 3DSMax [1].

In an environment such as the ice rink motion capture has to deal with several difficulties. For one, the lighting is often not smooth and there are several different light sources. Moreover, it is very difficult to get the whole ice rink for a recording, because most of the time the ice rink is fully booked with figure ice skating, ice-hockey, public skating or short track speed skating. Further, many ice rinks are governed by the local community [12], or are highly expensive to rent. This makes markerless and optical recordings very challenging. On the other hand inertial recordings as by the XSens technology seem very promising. Even when there are multiple skaters on the ice, recordings can be tracked in usual practice time slots.

3.1.2 XSens: Inertial motion capturing recordings on the ice rink

In our research study we tested recordings on the ice rink with the MVN Link product from XSens. This product includes 17 wired IMU motion trackers which can be either attached into a suit or onto straps. The sensors are connected via cables to a battery whit a life time of 9.5 h. These trackers are then connected via a router to their MVN Animate software. The wireless range of the trackers is 50 to 150 meters, however the trackers are able to buffer the recordings and transmit these later when there is a connection to the hub. Their MVN Animate software helps with calibration and shows the recording results as soon as the trackers are close enough for transmission. Furthermore, it processes the data and performs adjustments in a postprocessing step [29].

Our experiences with MVN Link from XSens

The overall setup was very time consuming. To correctly attach all the straps and align the cables took us a minimum of 20 minutes. Here we tried out the suit and the

straps with equal preparation times. Then the calibration on the ice with the MVN Analyze software was very time consuming again and took about 20 minutes. The callibration process had to be repeated several times, when the calibration failed. These problems probably came from the different gliding movement on the ice. So the skaters had to mimic usual floor movements for the calibration to work correctly. Since the recordings are very time consuming we could not work with young professional competitive skaters, because the setup was too drawn-out. We then took recordings from a senior competitive skater. The interial method was very beneficial here, because other skaters on the ice did not intervene in the recordings. Additionally, however sometimes the connection was lost to the hub, when the skater came back the recordings where transmitted correctly. The posprocessing from their software did work really good as well. What was disruptive however, where the sensors and battery, which the skater had to carry. This made the movements feel constrained. Further, in figure ice skating falls are common. A fall on the battery or one of the sensors would have been harmful, which again influenced the movement of the skaters. In comparison there are other products on the market, that promise a faster setup e.g. through a suit in which the sensors are integrated without cables [31]. These products would very likely better fit for figure ice skating motion captures.

Create datset with Blender and MakeHuman

We processed the recorded data from the XSens trackers in MVN Animate and then exported a short sequence as bvh file to test possible dataset creation procedures. Further, we used the open source MakeHuman [27] to create a figure skating avatar. Here we used the figure skating dress and ice skates, which are freely available from the MakeHuman community [54]. With the help of the MakeHuman plugin in Blender 2.7 [4] we then imported the created avatar from MakeHuman and retargetted the bvh motion capture file onto the rig. We then set up a default scene with lighting and a moving camera. To obtain the keypoints, we wrote a Python script in Blender to calculate the joint locations in relation to the camera view and exported the resulting keypoints into numpy files.

However, the above described process seemed to be smooth to conduct, we did face

a couple of challenges. For one, artistic elastic movements yielded in errors on the armature, since the joints were restricted to certain regions of movement, which seemed to work for usual daily activities but not figure ice skating ones. Another one was that with the moving camera the keypoints were not calculated correctly and drawn with a small offset. Recent research studies suggest to export the animation as bvh files and later calculate the joint positions with a program such as Matlab [7, 35, 44]. So this could improve the calculationof joint positions.

Our conclusions on creating a synthetic dataset in figure ice skating

We are convinced that the above described process to create a synthetic dataset for figure ice skating is a indispensability when it comes to figure ice skating, because of the following reasons: first: with inertial motion tracking is very accurate espeacially in an environment such as an ice rink when recording can be easily done during practice sessions. second: much less recordings are necessary thanks to the large arbitrary degrees of variation which is possible with an animation software such as blender third: the diversity of data is exceptionally vast. From one small motion capture file multiple different avatars can be retargetted, female and male, age variations, clothes and race variances. Further, multiple different light sources and camera views can be applied and the background randomly selected. fourth: calculated labels by the according animation software or post-processing program are just more exact then human labeled data.

All these points confirm the legitimacy of synthetic datasets in figure ice skating. Which is why we researched on already existing synthetic datasets since in the scope of this thesis creating an own dataset was not possible due to the huge time efforts. The systhetic datset 3DPeople as described in the next section correspond our expectations of a synthetic dataset.

3.2 Synthetic Dataset: 3DPeople

The dataset 3DPeople was created via created motion capture files. They took 70 realistic action sequences from Mixamo [28]. The sequences contain on average 110

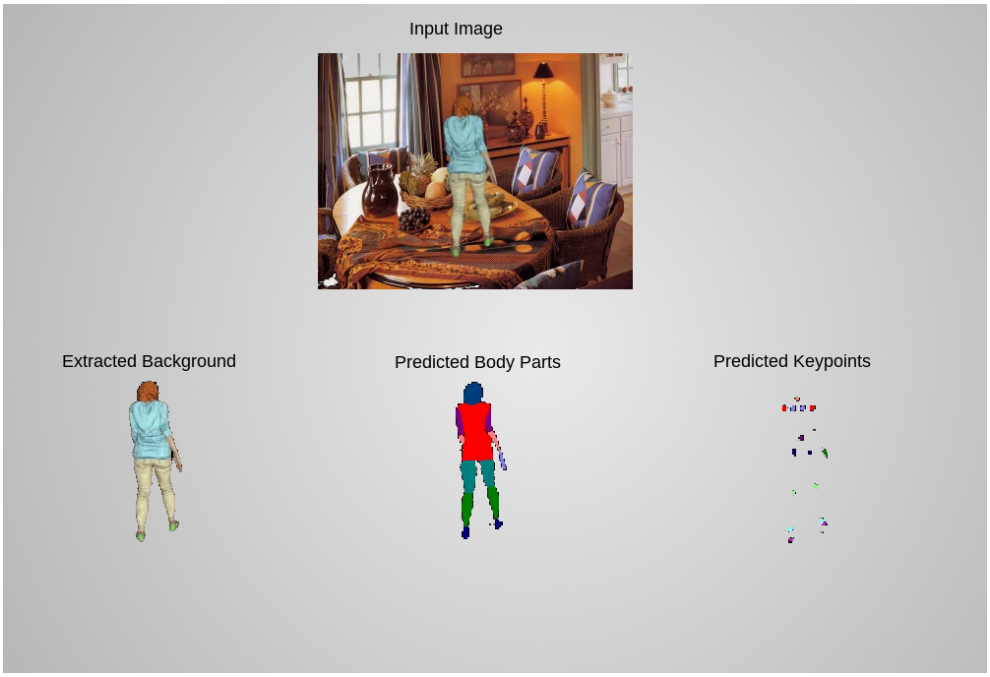

FIGURE 3.1: Learned labels from a image sequence of the 3DPeople dataset: Woman stiff walk

frames and range from usual activities with little motions such as driving or drinking until break dance moves or backflips. These actions they then retarget onto 80 armatures which are created with MakeHuman [27] or Adobe Fuse [26]. There are 40 female and 40 male characters with plenty of variation in body shapes, skin tones, outfits and hair. Furthermore, they record the actions with a projective camera and 800 nm focal length from four different viewpoints which are orthogonally aligned with the ground. The distance to the subjects vary arbitrary as do light sources and static background images. In sum the dataset includes 22,400 clips with the rendered video sequences as 640x480 pixel image frames, the according depth maps, optical flow and semantic information such as body parts and cloth labels [35].

We found this dataset highly sofisticated due to the wide variance and diversity of the data. Especially in terms of clothes it stands out from other famous datasets such as Human3.6M [45]. They use a variety of wide and tight clothes. For example do they include dresses and shorts. There variance is greater compared to their freely available combats [19, 45]. In Figure 3.1 we demonstrate the learned labels of our modules from a random image of the 3DPeople dataset.

```
_ Mapping to classes _
body_part_classes = {
    BodyParts.bg.name: 0,
    BodyParts.Head.name: 1,
    BodyParts.RUpArm.name: 2,
    BodyParts.RForeArm.name: 3,
    BodyParts.RHand.name: 4,
    BodyParts.LUpArm.name: 2,
    BodyParts.LForeArm.name: 3,
    BodyParts.LHand.name: 4,
    BodyParts.torso.name: 5,
    BodyParts.RThigh.name: 6,
    BodyParts.RLowLeg.name: 7,
    BodyParts.RFoot.name: 8,
    BodyParts.LThigh.name: 6,
    BodyParts.LLowLeg.name: 7,
    BodyParts.LFoot.name: 8
}
```

```
_____ Mapping to RGB values _____
segmentation_class_colors = {
    BodyParts.bg.name: [153, 153, 153],
    BodyParts.Head.name: [128, 64, 0],
    BodyParts.RUpArm.name: [128, 0, 128],
    BodyParts.RForeArm.name: [128, 128, 255],
    BodyParts.RHand.name: [255, 128, 128],
    BodyParts.LUpArm.name: [0, 0, 255],
    BodyParts.LForeArm.name: [128, 128, 0],
    BodyParts.LHand.name: [0, 128, 0],
    BodyParts.torso.name: [128, 0, 0],
    BodyParts.RThigh.name: [128, 255, 128],
    BodyParts.RLowLeg.name: [255, 255, 128],
    BodyParts.RFoot.name: [255, 0, 255],
    BodyParts.LThigh.name: [0, 0, 128],
    BodyParts.LLowLeg.name: [0, 128, 128],
    BodyParts.LFoot.name: [255, 128, 0]
}
```

3.3 Data Processing

The 3DPeople dataset can be recrested through their project homepage and is allowed for use in research enironments. Each data package consists of five batches for women and men. There are five data packages available for download. We created a *data_admin python class*, which takes care of downloading the data, processing and storing the data in memory agnostic and efficient access structured compressed numpy files, and delete the initial downloaded data. We included several processing steps: In sum we created four compressed numpy archives. Every archieve file included an array with all frames of one movement sequence to allow faster reading process for our neuronal network training later. The four archives consist of the usual RGB sequence frames, one archive without background, one for the body part labels, and one for the joint keypoints. The numpy archive without background only shows the actors in RGB colors. Therefore, we used the clothes labels, and replaced all colored pixels with black pixels when the pixels where not inside the mask. Initially we did this to test, weather our network ideas would converge with easier data. Later, when we trained our background-extractor module for getting an end-two-end architecture, this data served as labels.

Furthermore, we cleaned up the body masks and mapped the resulting three dimensional pixels to one-dimensional class values Figure **??**. The borders of the body-parts often contained a mixture of RGB values, which where of no use to our network.

Chapter 4

Method

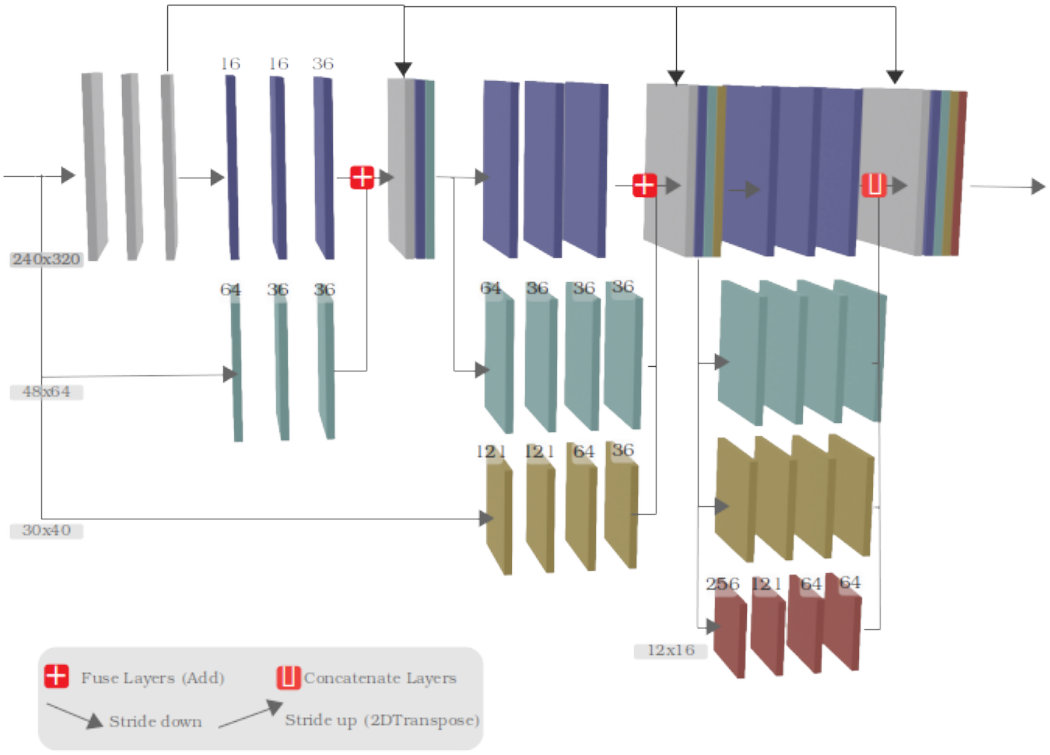

FIGURE 4.1: High-to-low representations network architecture

For our end-to-end keypoint recognition architecture we have created three modules to extract the background, find the body parts in the image and detect the human joints or keypoints. However, to extract the backround was not important for other keypoint recognition architectures such as OpenPose [5] or VideoPose3D [34], it is important in our architecture, since we wanted this recognition architecture to be able to be used during practice, when there are multiple skaters on the ice, but only track the focused skater. With the body part detection module we altered the well established approach from OpenPose not to recognize vector fields, which the

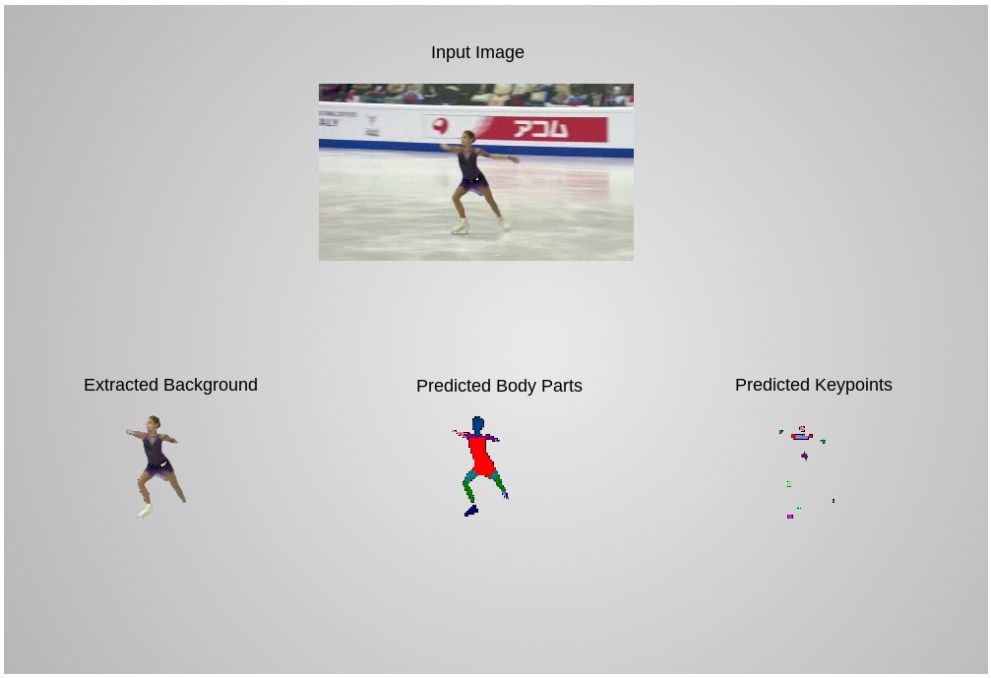

FIGURE 4.2: Learned labels by the three modules: extracted background, human part detection and keypoint detection *skater: Alena Kostornaia, 2020 European champion[51]*

joints connect,but the visible body parts. We assume this method to work seamlessly, and estimate that this approach was not chosen before due to the missing accurate labels for body part detection. For the keypoint recognition module, we calculate the gaussian with a radius of three pixel and a standard deviation of three. In Figure 4.2 we demonstrate an example frame labeld by our three modules showing Alena Kostornaia, the 2020 European champion during her program.

All in all we have build a fully convolutional architecture with three networks, that all are based on high-to-low representation learning. Our architecture consists of one input block $\mathcal{N}_I$ and three subsequent blocks $\mathcal{N}_L$, $\mathcal{N}_M$,$\mathcal{N}_S$ and $\mathcal{N}_{XS}$. These subsequent blocks combine feature maps with lower coarser representations with the original sized input image feature maps and thus learn the features of the different levels equal to the HRNet strategy [21, 57].

However, different from the HRNetV1 and HRNetV2 we do not use pooling to decrease or increase the size of the feature maps. We decrease the feature maps with usual convolutions and accoring strides $\mathfrak{s}$ and kernel sizes $\mathfrak{k}$, with $\mathfrak{s} = \mathfrak{k}$. To increase the feature maps we use transposed convolutions with again $\mathfrak{s} = \mathfrak{k}$ according to the strided down convolutions. This allows the network to learn additional weights

for the upward and downward convolutions and improve these level exchange processes.

Furthermore, we fuse the layer blocks in the first and second stage to combine the mentioned feature levels. In the third stage however, we concatenate all feature levels to fully exploit the multi-resolution convolutions as argued in HRNetV2 [21].

As visualized in 4.1 we adjusted the amount of feature maps for the different blocks. Moreover, in the first stage we use only three convolutional layers for one block int the other stages we use four layers for the lower levels and only in $\mathcal{N}_L$ we use only three convolutional blocks throughout the network.

Another adjustment is, that we use the input image as initial input for all our block levels but the $\mathcal{N}_{XS}$ block, wich uses the fused layers of all the other levels as input. To every stage we add the input block $\mathcal{N}_I$.

This network architecture resulted after several experiments and investigations of the estimated feature maps of the different blocks and stages. Every block is completed with batch normalization and a *selu* activation functions. The output of the network is predicted by a linear softmax activation function. For the background-extraction and keypoint detection network we use mean-squared-error as loss functions and in the human part detection network we use a custom loss function chapter 5 to optimize the network weights. In sum our network comprises 3,008,562 parameters of which 3,003,930 can be learned. The amount of all layers for one network is 156.

4.1 Training Performance (human parts)

We trained the human parts module with the Adam optmizer for 5556 episodes, a batch size of 3 and 64 steps per epoch. One training epoch took on average 85.43 seconds and took about 5.5 days. For optimization we have build a custom loss function as explained in chapter 5 to better deal with the class invariance of the occurring pixel labels. The Figure 4.3 shows, that the accuracy of our network Figure 4.3 rises

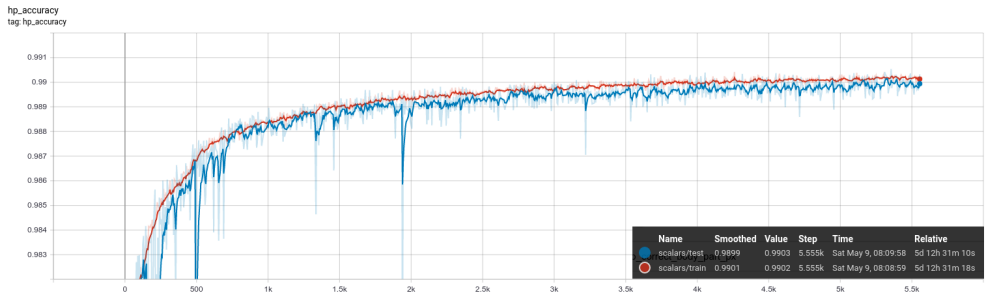

FIGURE 4.3: Learned labels by the three modules: extracted background, human part detection and keypoint detection *skater: Alena Kostornaia, 2020 European champion*[51]

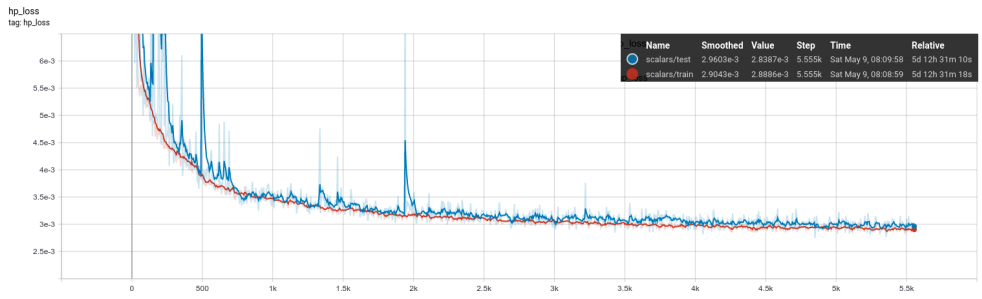

FIGURE 4.4: Learned labels by the three modules: extracted background, human part detection and keypoint detection *skater: Alena Kostornaia, 2020 European champion*[51]

very steep until the 500th episode and then flattens more and more until the last episode, where the network reaches an excelent accuracy level of 0.99. Furthermore, we devided our data in a distinct 90 percent train and 10 percent test dataset. The figure shows equal trends for both datasets, which means the network did not overfit on the data.

The loss function in Figure 4.4 shows equal opposite trends to the accuracy graph. It steeply decreases until the 500th episode and then starts to flatten. Again train and test dataset do not cross each other but show similar courses.

4.2 Inference Runtime Analysis

The time to predict one frame of size 640x480 pixel takes about 0.3 seconds run on a Quad-Core Intel i7 CPU with 2.20 GHz, a simple laptop cpu. Predictions and training can run on simple hardware as the above mentioned cpu. The training speed will increase if the minimum version of Cuda 3.5 is suported by the system, since then tensorflow 2 is able to run on the gpu instead of the cpu. Due to the

age of the named laptop cuda 3.0 was the highest to be supported. Since recent developments targeting AI chips in the mobile world by the common companies Apple, Samsung or Huawei, we are very confident that inference of our network does work as well on common sold mobile phones today [30]. This sets apart our architecture from OpenPose, which we could not use for inference on the mentioned laptop.

4.3 Implementation Details

We build our Network upon the high-level API TensorFlow 2 which is based on the Keras API [50]. Our training run on the AI server *J.A.R.V.I.S.* provided by the Stuttgart Media University. The server contains four Nvidia Titan Xp GPUs with 12 GB RAM and a i7 8-core CPU with 3.2 GHz. We trained each experiment on a separate GPU. The training was conducted in docker containers using the TensorFlow *latest-devel-gpu* [49] docker build. The latest TensorFlow docker container did not support Python 3 at the time of this research, but only Python in version 2, which is why we had to use the before mentioned descendant.

Chapter 5

Experiments

We conducted several experiments on our human parts and keypoint detection module. We applied the resulting network architecture from the human parts module later to the background extraction module. In sum we investigated eight network architecture variants differentiating in commonly discussed parameters. For the keypoint detection module we conducted tests with and without Dense layers. On the human parts detection module we tested the three state-of-the-art optimizers Adam, Nadam, and SGD and experimented with little modifications to the SGD optimizer. To overcome class invariance, we tried out three different loss functions under which one we have come up with ourselves. In the following section we will present our experimental setup, results and constructive thoughts behind the studies.

5.1 Ablation Study

5.1.1 Body Parts Detection Module

In 5.1 we demonstrate our network architectures, which range from 88 until 177 layers and are all fully convolutional networks. The HRNet *(traditional)* 5.1.1 we slightly deviate the HRNetV2 presented in [21] by replacing the pooling layers with strided-down and transposed convolutional layers. The amount of levels and sizes of the feature maps correspond to 4.1. As in HRNetV2 we use 4 stages with a filter size of 64 for each stage. We add one level per stage starting with just the highest level $\mathcal{N}_L$ and concatenate the levels from stage 2 ongoing as demonstrated in HRNetV2.
For the HRNet *(filters)* 5.1.2, we adjust the filters in a way, so that higher levels use less filters then lower levels. Additionally, we decrease the filters from the convolutional layers in the level blocks, starting from the highest filter amount reducing until 36. All levels are concatenated with a filter size of 36.

TABLE 5.1: Ablation Human Parts Module: Network Architecture Comparison

	Name	Parameter Amount	Trainable Parameters	Layers	Training Time/Epoch
5.1.1	HRNet *- traditional*	5,595,221	5,589,237	171	84.76s
5.1.2	HRNet *- adjusted filter*	4,936,997	4,933,029	171	66.23s
5.1.3	HRNet *- 3 stages*	848,409	846,441	100	82.33s
5.1.4	HRNet *- stride-down-up*	4,953,269	4,949,157	177	63.21s
5.1.5	HRNet *- stride-down-up-input*	4,185,605	4,181,493	177	80.57s
5.1.6	HRNet *- add-input*	4,185,605	4,181,493	177	80.57s
5.1.7	HRNet *- add-depthwise-conv*	3,182,342	3,177,654	156	83.10s
5.1.8	UNet	656,389	654,261	88	85.37s
5.1.9	HRNet *- v3*	3,008,562	3,003,930	156	85.43s

Subsequent architectures all make use of the adjusted filter amount.

For the HRNet *(3 stages)* 5.1.3, we use only the three first stages from 5.1.2 and omit $\mathcal{N}_{XS}$. In HRNet *(stide-down-up)* 5.1.4, we stride down the Input from 240x320x3 to 120x160x36 and use this layer as Input layer and highest Level $\mathcal{N}_L$. Lower layers scale relatively to $\mathcal{N}_L$. The HRNet *stride-down-up-input* 5.1.5 uses the intially strided down input layer as input for every stage instead of the concatenated results for lower levels.

As presented in the mobilenet paper [46] in 5.1.7 we experiment with depth-wise convolution and replace the concatenation layers after each stage with add layers.

In the UNet 5.1.8 we use all levels and combine them via concatenation according to the traditional UNet [41].

Finally, we present our resulting high-to-low resolution network HRNet *(v3)* 5.1.9, where we fuse the first stages and only concatenate the last stage. Furthermore, we conducted some adjustments to the layer amount in the levels and stages as presented in chapter 4.

The HRNet stride-down-up 5.1.4 seems to train the fastest, when looking at 5.2 the 4.705kth step, it is only at three days and 13h closely followed by the traditional HRNet with the adjusted filters 5.1.7 which is at 3 days and 13h. All other trainings took more than four days at that time. However, one has to keep in mind that these both trainings where conducted at the same dates, and the other dates differ. So the server

could have had other workload, nevertheless, our trainings were the only ones running on the server most of the time. This occasion as well is reflected in the training time of one epoch. There we measured the third epoch to omit starting warm up phases. There as well both the HRNet *stride-down-up* and HRNet *adjusted-filter* only took about 63, 66 seconds, where all the others took more then 80 seconds. However, we would have expected the U-Net or HRNet *3 stages* to be the fastest, since they have the smallest amount of layers and trainable parameters. Further, the HRNet *stride-down-up-input* contains less trainable parameters then *stride-down-up*, so we would have expected a faster training process here. This verifies that the measurements are dependent from various circumstances.

Comparison of network architectures

The U-Net 5.1.8 has the smallest amount of layers with 88 and 654,261 trainable parameters. The HRNet *stride-down-up* 5.1.4 composes most trainable parameters with 4,949,157 and 177 layers, since the input layer is first strided down and at the end transoped up again. On the other hand due to the reduces feature map sizes this reduces the computation effort as well. All comparison figures show very similar curves. Since we could only run our experiments one complete training each, the results must be viewed circumspectly. All figures **??** show very steep curves until the 500th episode and start to flatten then.

When looking at 5.4 and 5.5 this steep initial increase is visually reflected. When after the first episode the circumferences of the body shapes are predicted vaguely, after the 20th episode the networks have already learned to predict the locations of body parts relatively close. Of course here has to be regarded, that the different predictions were made on different poses, which might differ in their complexity. Nevertheless, the UNet 5.1.8 and the HRNetV3 4.3 seem to accomplish superior performance. After the first episode, the body shape is already estimated precisly close to the input image. HRNetV3 even shows correct class estimations for different body parts such as torso, head, arms and legs already. The 20th episode then the labels seem to be estimated very close to the true labels.

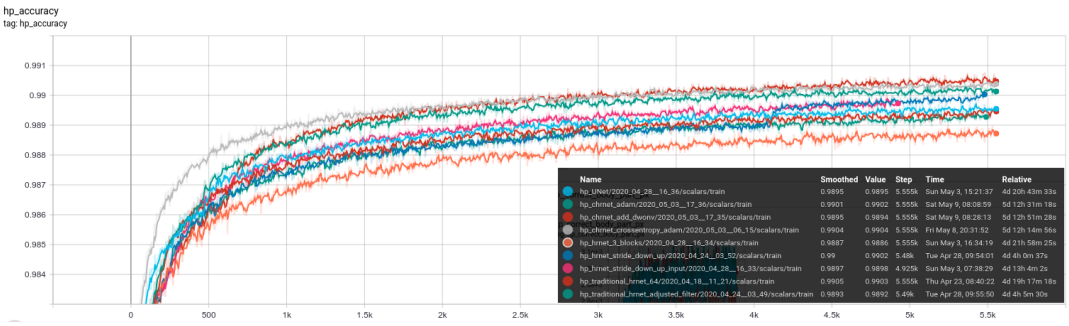

FIGURE 5.1: Ablation Human Parts Module: Accuracy Comparison

The accuracy graph shows that the HRNet *traditional* and HRNetV3 get the best results from epoch 500 onwards. At step 55.55k they reach 99 percent. However, the HRNet *stride-down-up* has a lower courve until the 4.3kth epoch, it then makes a jump in accuracy and at the end reaches 99 percent as well. Interestingly the U-Net with the by far fewest paramets and layers does receive very good results as well making almost 99 percent in accuracy at the end. The HRNet with only three stages showes the lowest curve and reaches only 98.87 percent points on accuracy at the end.

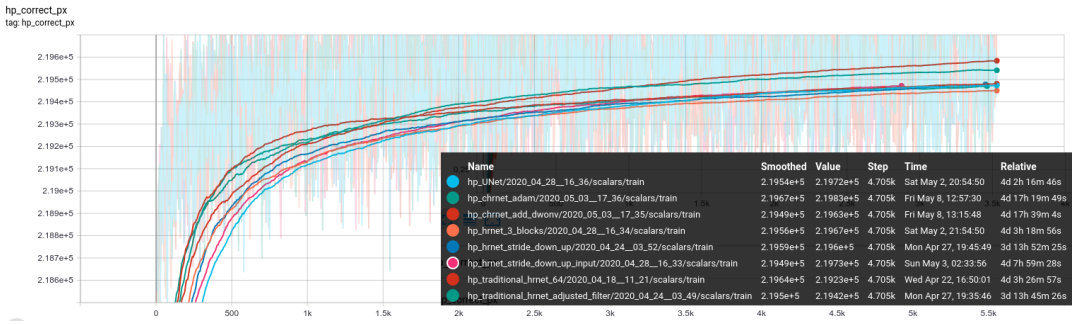

FIGURE 5.2: Ablation Human Parts Module: Correct Pixel

The correct pixel graph 5.2 measures how many pixels of the 240x320 image where labeled correctly in one batch. The maximum which could be reached here is $240 \times 320 \times 3 = 230\,400$. Since the single epochs vary a lot it makes more sence to look at the smoothed results. The courves again are very close to each other, with the best results coming from the HRNetV3 with umprint219670 and the HRNet *traditional* with 219 640. The lowest curve again shows the HRNet *3 stages*.

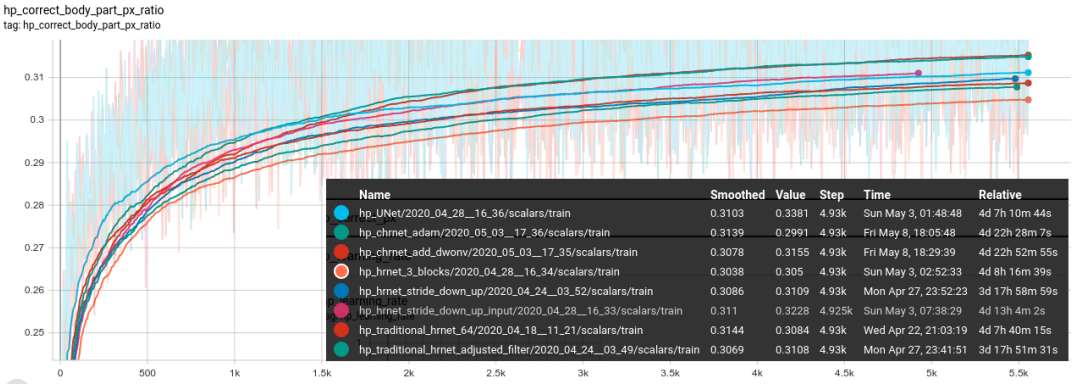

FIGURE 5.3: Ablation Human Parts Module: Correct Pixel Ratio

For 5.3 we summed up the amount of body part pixels and divided by the amount of correctly predicted pixels for one training batch. This figure as well displays high varieties for the different epochs, which is why it is important to take the smoothed values into account. We receive similar courses as already inspected in 5.2. The HRNet *traditional* and HRNetV3 take the lead wih the *traditional* being slightly above the HRNetV3 with 0.05 percent points. With 31.44 percent the HRNet *traditional* shows best results, whereas HRNet *3 blocks* performs worst with 30.38 percent at step 4.93k.

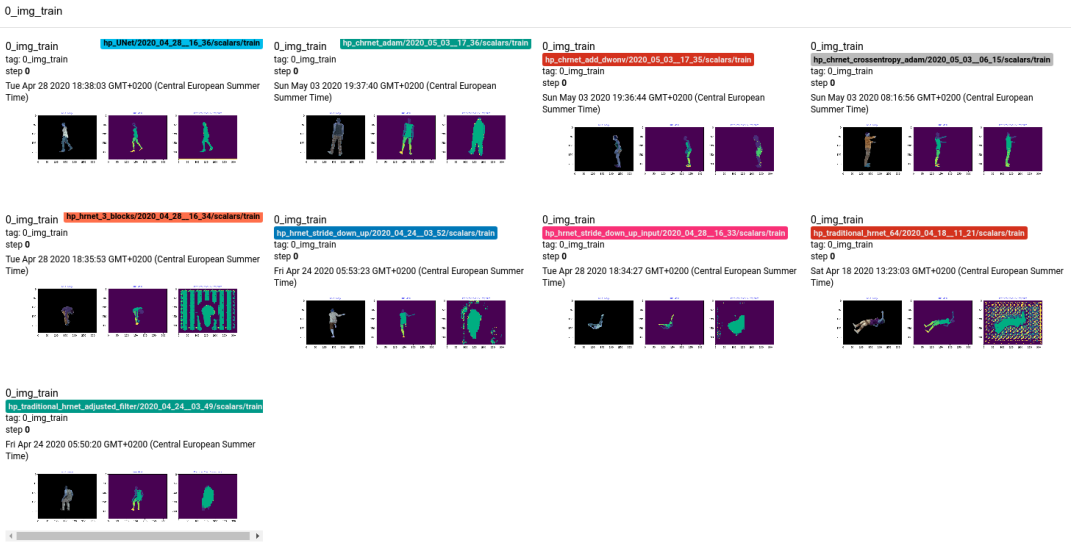

FIGURE 5.4: Predicted images of network architectures after first episode

FIGURE 5.5: Predicted images of network architectures after 20th episode

Summary

In summary our experiments show that our constructed HRNetV3 performs best, and the HRNet with only three stages performs worst. Interesting here would be different variations in the level of feature map sizes. For example what would happen if the level $\mathcal{N}_{XS}$ was maintained as smallest level and the level $\mathcal{N}_{M}$ was strided down slightly more. Insightful as well it the performance of the UNet, which has less than half of trainable parameters and layers compared to the HRNet *traditional* or HRNetV3, but does not perform much worse.

Additionally, further tweaks and studies in these network architectures applied to our different modules would be greatly enlightening.

5.1.2 Joint Detection Module

TABLE 5.2: Ablation Keypoint Detection Module: Network Architecture Comparison

	Name	Parameter Amount	Trainable Parameters	Layers	Training Time/Epoch
5.2.1	KPS-Dense - *block_L*	221,748,579	218,735,769	51	68.75s
5.2.2	KPS-Dense	23,097,980	20,086,328	13	67.05s
5.2.3	KPS-FCN	4,119,529	1,109,991	41	63.78s

Network Architectures

We created three variations of keypoint detection network architechtures of which two**??** utilize Dense layers to estimate the keypoint locations and KPS-FCN 5.2.3 implements the third stage of our build HRNetV3.

KPS-Dense 5.2.1 additionally to the dense stage utilizes the third stage of the HRNetV3.

The dense stage first uses max pooling to reduce the feature map size from 240x230x9 to 120x160x1, 58x78x32 and then 28x38x64. The resulting pooling layer is flattened and followed by two dense layers, which are combined with AlphaDropout and batch normalization. The layers include 1024 and 512 units. The first dense layer is conneted to a softmax activation function, the second to a linear activation function. The model estimates 38 x,y coordinates for the keypoint locations. The networks loss function calculates the distance from the predicted keypoint locations to the true locations and optimizes the networks via mean-squarred-error.

For the KPS-FCN 5.2.3 only uses convolutional layers. The true labels calculate gaussians with a radius size of three for the locations of the keypoints. We combined the keypoint classes for equal body locations such has legs and arms, resulting in 11 joint classes. This network predicts 11 different labels for the classes. The network is optimized as well with mean-squarred-error.

All networks are trained with an Adam optimizer and a learning rate of 0.001 and run for 5556 epochs.

FIGURE 5.6: Predicted images of keypoint detection networks after first, 20th and last episode

Both figures 5.7, 5.8 show similar curves. Until the 500th episode they decrease steeply and then they start to flatten. KPS-Dense 5.2.2 has the fewest layers with 13, however KPS-FCN 5.2.3 contains only 1,109,991 trainable parameters, the dense networks train 20/ 218 times more parameters. The networks only differ slightly in their training epoch time and are all between 63 and 69 seconds.

The main difference though is visible in the predicted training images in 5.6. The keypoint detection modules seem not to learn the locations of the body parts, but that the probability of the location of a keypoint being at the center of the image is much more likely than outer locations. Very interesting about the KPS-FCN is that this network first learns the locations of the feet, and then slowly goes upward per epoch learning the other keypoint locations in a continuous way.

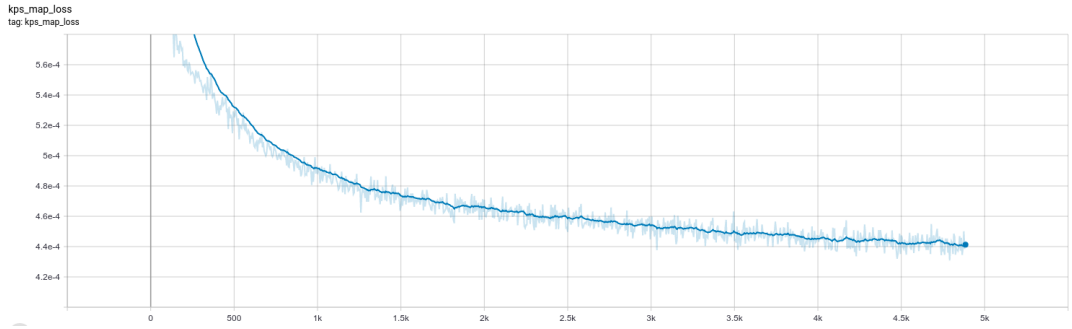

FIGURE 5.7: Predicted images of keypoint detection networks after first, 20th and last episode

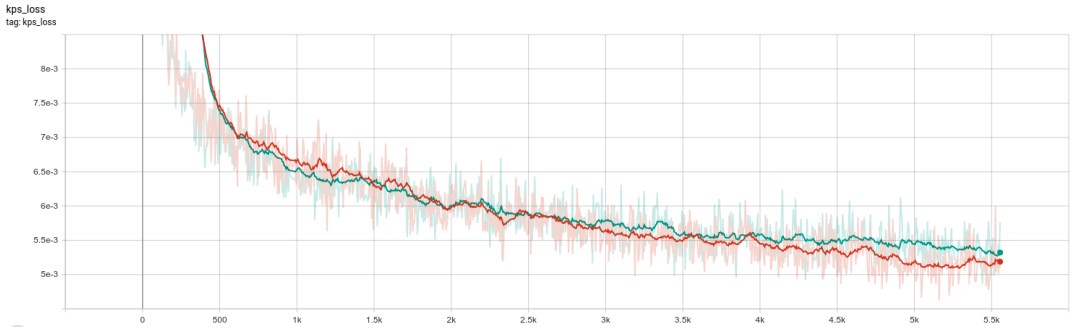

FIGURE 5.8: Predicted images of keypoint detection networks after first, 20th and last episode

Fully Convolutional

5.2 Comparison of Optimizer Algorithms

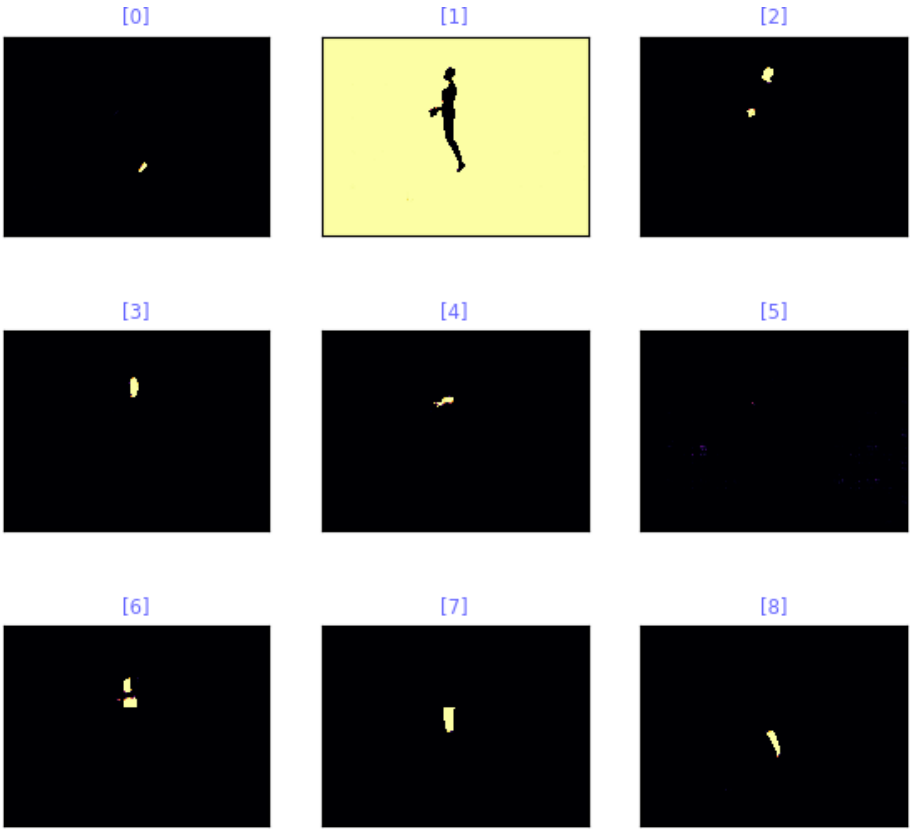

FIGURE 5.9: HRNetV3: Predicted 9 feature map classes of the output layer of the human part detection module

Optimizing the weights of a neuronal network is a crucial step in the training life cycle of deep learning. When our neural network has processed the input image and calculated several feature maps throughout the process, at the end it predicts 9 feature maps of body parts for example for head, arms, torso and legs for the human parts detection module 5.9. With the help of a loss function we then calculate the error for the predicted pixels in comparison to the truly labeled pixels. The optimizer updates the network parameters $\theta \in \mathbb{R}^d$, which result in learned filters predicting several feature maps at the different stages of the network. The updates are performed accordingly to the calculated error from the loss function $\mathcal{J}(\theta)$ via the so called Backpropagation process, since the updates are passed backward through the network. This backward optimization process is performed by the calculations of the chosen optimizer. The goal of the optimizer is to minimize the error. There, it

makes use of gradient descent to update the network's weights, biases and activation functions, in opposite direction of the gradient. Figuratively speaking, a weight could be imagined as a hill. The updates then will follow the direction of the slope downhill until the valley is reached. Whereas the valley is equal to a local minimum [43].

There are three variants for gradient descent:
Batch Gradient Descent (BGD), which calculates the gradients θ for the entire training dataset. The problem here is, that most of the time this would not fit into the RAM. We for example had problems with data overflow when we trained with more than 12 images at a time, however we even reduced the image size to 320x240. Another drawback is, that gradients for similar examples must be recalculated before each update, leading in redundant computations.
Stochastic Gradient Descent (SGD) confronts this problem by performing updates for each training example input image x_i and mask label y_i. This optimization process is much faster. Moreover, the updates are of more variance when random training examples are chosen. A drawback however is, even if jumping around may result in finding another better local minimum, it may also lead to constantly overshooting the exact minimum. Against this problem, slowly decreasing the learning rate may help and lead to similar convergence as for BGD.
Mini-Batch Gradient Descent is the most common variant for gradient descent since it combines the aspects of BGD and SGD. The updates are conducted for every mini-batch of n training samples. This reduces the variance of parameter updates and leads to less jumping around, which all in all results in more stable convergence [43].

In the following section we will present an overview on optimizers targeting the optimizers we experimented on, which are SGD, Adam and Nadam. We conducted our training with Mini-Batch Gradient Descent with a batch size of three. In the upcoming section we will present some mathematical backgrounds of optimizer algorithms and explain further aspects of SGD in more detail.

Overview of our used optimizers

Stochastic Gradient Descent (SGD) is one of the first optimizers and still commonly in use. Already in 1951 it was publicly reported by Robins and Sutton et Al [40]. Now given a neural networks parameters θ, such as weights, biases and activation functions, here via an input image $\mathfrak{x}$ and according labels $\mathfrak{y}$ the Backpropagation process is defined under the gradient ∇ from the calculated loss function $\mathcal{J}$. The adaption of the parameters then is performed stepwise by the learning parameter η.

$$\theta = \theta - \eta \cdot \Delta_\theta \mathcal{J}(\theta; x; y) \tag{5.1}$$

The SGD optimizer updates a weight w in a way to reduce the loss function $\mathcal{J}(\mathrm{w})$ and find the local minima of the parameters. As already mentioned this is one of the problems with pure SGD, it will very likely become stuck in a local minimum or saddlepoint, where one slope goes up and the other one down. Furthermore, it converges slower compared to newer optimizers [15, 43].

Momentum, first proposed in 1999 by Qian et Al. [36], can help to get faster to the local minimum. This is accomplished by an additional term which is added to SGD. A constant fraction γ is multiplied with the last parameter update $\mathfrak{v}_t$ to θ.

$$\theta = \theta - \mathfrak{v}_t \tag{5.2}$$

$$\mathfrak{v}_t = \gamma \mathfrak{v}_{t-1} + \eta \cdot \nabla_\theta \mathcal{J}(\theta) \tag{5.3}$$

Since the momentum term includes all previous updates 5.3, it allows the optimizer to accelerate in terms of speed towards the local minimum and thus converge faster.
This can be associated with a ball rolling down a hill. The momentum term increases as the ball rolls downhill accelerating in terms of speed, for subsequent gradients which pointing in the same direction. When the direction changes, the term decreases, the ball slows down. All in all this results in faster convergence and less oscillation or jumping around the minimum [43].

Adaptive Gradients (AdaGrad), a newer algorithm proposed in 2011 by Duchi et

Al. [11], adapts the size of an update to the importance of the individual parameters. This is a great benefit for the occurrence of sparse data, which is the case for image data or word embedding tasks. It adjusts the learning rate term, by dividing the learning rate through the sum of previous updates w.r.t the network parameter θ_i at time step t. An additional smoothing term η prevents the division by zero.

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{\epsilon + \sum_{t=1}^{t}(\nabla_\theta \mathcal{J}(\nabla_{\theta_{t,i}}))^2}} \cdot \nabla_\theta \mathcal{J}(\theta_{t,i}) \tag{5.4}$$

This results in the learning rate decreasing when approaching a local minimum, meaning the overshooting problem is eased off. However, one weakness is the growing dominator, which makes the learning rate shrink to a infinitesimally small number until the step size almost dissolves to zero.

Root Mean Squared Propagation (RMSProp) extends AdaGrad's root squared loss function. It is an unpublished optimization technique proposed by Hinton et Al. [17]. The dominator takes, additionally to the past squared gradients, the current gradient at the current time step t is taken into account, weighting the last update more than the current. Hinton suggests to weight the last update with 90 and the current with 10 percent.

$$\theta_t = \theta_{t-1} - \frac{\eta}{\epsilon + E[g^2]_t} \cdot \nabla_\theta \mathcal{J}(\theta_{t-1}) \tag{5.5}$$

$$E[g^2]_t = (1-\gamma)g^2 + \gamma E[g^2]_{t-1} \tag{5.6}$$

$$g = \nabla_\theta \mathcal{J}(\theta_{t,i}) \tag{5.7}$$

Unlike AdaGrad, RMSProp does not decrease monotonously, but can adapt the size of adaption. Now larger or smaller updates are possible in reference to their impact. Nevertheless, the issue with diminishing learning rates remains.

Adaptive Moment Estimation (Adam) [22], as well as AdaGrad or RMSprop, calculates adaptive learning rates w.r.t. the the parameters θ_i. As an extension it keeps track of an exponentially decaying average of past gradients m_t, as was already done in momentum.
Coming back to the visual anecdote, the ball has a lot of weight and is very heavy [43]. It prefers flat minima.

Past decaying average m_t and past suqared errors v_t estimate the first momentum (the mean) and the second momentum (the uncentered variance).

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{5.8}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{5.9}$$

Since these terms are biased with zeroes, the initial updates are very small. Which is why $\hat{m}_t$ and $\hat{v}_t$ bias these terms:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{5.10}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{5.11}$$

This results in an update rule which reassembles RMSProp:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{5.12}$$

In particular, Adam makes use of two decay parameters β_1 and β_2, referred as exponential decay rates. These parameters β_1 and β_2 are close to the constant γ term used in RMSprop and Momentum. The great advantage here is, that the learning rate can adabt in both directions solving the issue from AdaGrad or RMSProp of vanishing learning rates.

Nesterov-accelerated Adaptive Moment Estimation (Nadam), now replaces Momentum with the better performing Nesterov accelerated gradient (NAG). It was first presented in 2016 by Dozat et Al [10].

NAG allows to approximately calculate the future position of the parameters. Referring back to Momentum, NAG is like a smarter ball, which knows that it has to slow down and not naively go up an upcoming slope again just to roll back down. NAG's first step is according to the last parameter update and taking the fraction γ

into account. The second step approximates the future position of the parameters by calculating the loss function from θ w.r.t. the previous update step. This step can be interpreted as a correction step.

$$\theta = \theta - \mathfrak{v}_t \tag{5.13}$$

$$\mathfrak{v}_t = \gamma \mathfrak{v}_{t-1} + \eta \cdot \nabla_\theta \mathcal{J}(\theta - \gamma \mathfrak{v}_{t-1}) \tag{5.14}$$

To combine Adam and Nadam, some steps must be done. We can write the gradient term as g_t and recall the Momentum update and parameter update function θ_{t+1} with:

$$g_t = \nabla_{\theta_t} J(\theta_t) \tag{5.15}$$

$$m_t = \gamma m_{t-1} + \eta g_t \tag{5.16}$$

$$\theta_{t+1} = \theta_t - m_t \tag{5.17}$$

The 5.17 m_t term can be extended with 5.16 resulting in **??**:

$$\theta_{t+1} = \theta_t - (\gamma m_{t-1} + \eta g_t) \tag{5.18}$$

Momentum takes one step into the direction of the previous momentum update and another step into the direction of the current gradient. However, instead of the past Momentum vector, the current update rule is used in order to look ahead. Furthermore, expanding term 5.12 $\hat{m}_t$ with 5.10 and 5.10 m_t with 5.8 results in the following:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} (\frac{\beta_1 m_{t-1}}{1 - \beta_1^t} + \frac{(1 - \beta_1) g_t}{1 - \beta_1^t}) \tag{5.19}$$

Where $dfrac\beta_1 m_{t-1} 1 - \beta_1^t$ is the correction step of the first step from NAG. This term again, can be substituted by $\hat{m}_{t-1}$ resulting in:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} (\beta_1 \hat{m}_{t-1} + \frac{(1 - \beta_1) g_t}{1 - \beta_1^t}) \tag{5.20}$$

As a final step the bias-corrected estimate of the momentum vector of the past time step $\hat{m}_{t-1}$ is replaced with the bias-corrected estimate of the current momentum $\hat{m}_t$ resulting in the optimization function for Nadam [43]:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\epsilon + \sqrt{\hat{v}_t}} \cdot (\beta_1 \hat{m}_t + \frac{(1-\beta_1)g_t}{1-\beta_1^t}) \tag{5.21}$$

5.2.1 Experiments with optimizers

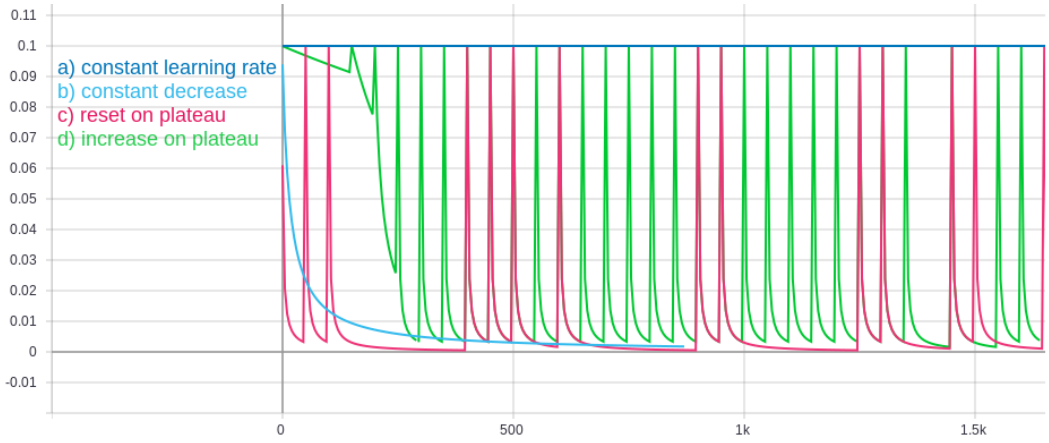

FIGURE 5.10: Learning Rate SGD.

In our experimants we tested SGD, Adam and Nadam all with intial learning rates of $\eta = 0.001$. We configured SGD to use Nesterov Momentum with a value of $\gamma = 0.9$, Adam with epsilon $\epsilon = 1e-7$ and AMSGrad. Nadam used the exponential decay parameters $\beta_1 = 0.9; \beta_2 = 0.999$ and epsilon $\epsilon = 1e-7$. Additionally, we conducted several tests with the learning rate in SGD and it's influence on the training process. One SGD configuration is with a constant learning rate η one is decaying by $\eta = 0.01$ and one learning rate is dacaying with $\eta = 0.01$ but will reset to the initial learning rate, when no significant learning or changes regarding the loss value is observed over the last 50 epochs anymore. Another experiment with SGD increases the ascent of decay when a plateau is hit with increasing decay values $[1e-5, 1e-4, 1e-3, 1e-2]$ as visualized in 5.10.

Comparison of Adam, Nadam and SGD

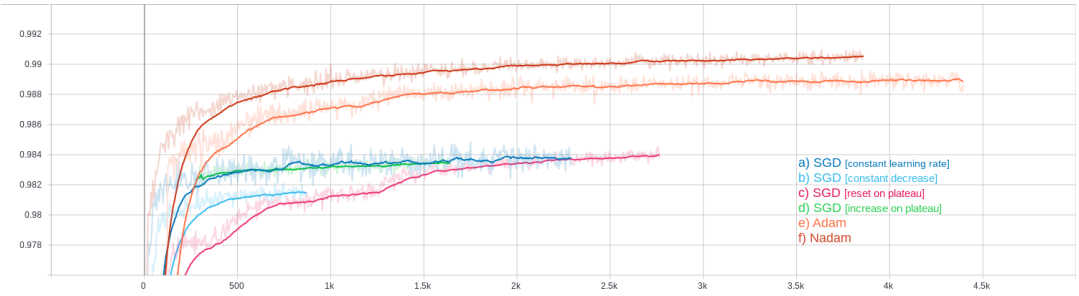

FIGURE 5.11: Accuracy

The Figure on accuracy 5.11 shows that our network makes huge learn progress in the first 200 to 300 epochs with an ascent of almost 90 degrees. However, the graph shows that SGD performs worse than Adam or Nadam, all networks reach a commendable accuracy above 98 percent, whereas Nadam even gets over the 99 percent mark. Adam and Nadam start to flatten a little bit later than SGD about 100 epochs. Furthermore, their curves flatten more evenly, whereas SGD seems to flatten more abruptly. Interestingly, the SGD optimizer, where we reset the learning rate on plateau, shows jumps in it's curve. There one can observe the typical flattening of SGD and then, when a reset took place, the learning process starts again. This indicates that SGD hit local minima from which it seems to jump out and re-trigger the adjustment of θ.

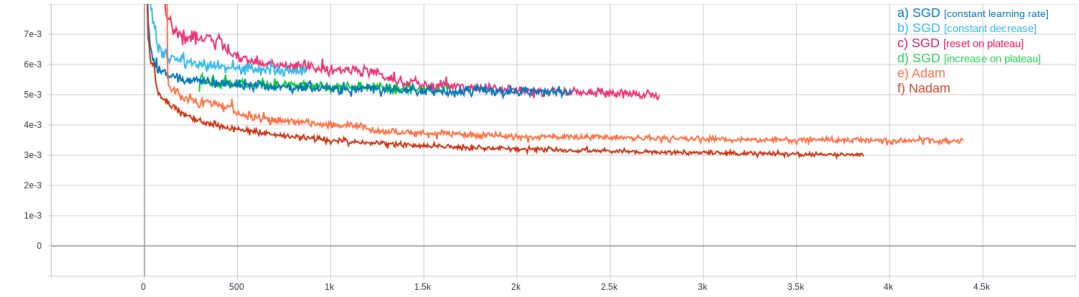

FIGURE 5.12: Loss

Similar process curve trends can be observed in 5.12, while the curves are mirrored at the x axis. This correlation of loss and accuracy trend proves the according learning process for the different optimizers.

Conclusions

Our results are reflected by the research from Choi et Al [9]. They conducted a very interesting research with the thesis that more general optimizers would never underperform the ones they approximate. Meaning RMSProp, Adam and Nadam would always perform at least as good if not better than Momentum, Nesterov or SGD. They impose to carefully tune the hyperparameters and not just conclude an optimizer would work better than another one. This is why we used very commonly recommended hyperparameters. A continuing grid search could help to find an even more efficient way for training and probably even result in better learned network parameters θ.

5.3 Performance of loss functions

As already mentioned in section 5.2 loss functions $\mathfrak{J}$ closely work together with optimizer algorithms. The loss function is used in the optimizer algorithm as seminal feedback deciding about the success of the learning process of a neural network. It calculates how close a prediction y_p from a neural network is to the true label y_t.

5.3.1 Commonly used loss functions

One classical loss function is Mean Squarred Error (MSE). The error represents the difference between y_t and y_p. MSE then squares the error to even negative results and calculates the sum of all these errors. For our fully convolutional network each pixel presents a certain class. As visualized in 5.9, our Human Parts module predicts nine classes. This means, there are nine label possible for each pixel. Now the neural network estimates with a certain probability how likely a certain class would be true for a specific pixel. The error then calculates the difference between the estimation or predicition and the true label, which inludes a value of 1 only for the true class. Additionally to the sumation of squarred errors, MSE divides this sum with the number of all squares resulting in the mean value:

$$MSE(y_t, y_p) = \frac{1}{n} * \sum_{[} i = 1]^{n} (y_t - y_p)^2 \tag{5.22}$$

We trained the Keypoint Detection and Body Part Extraction modules with MSE.

However, with our Body Part Detection Module we intially had problems with class inbalances. First we conducted our training with Sparse Categorical Cross Entropy (SCCE), which is a common loss function used in image segmentation. Wrong predictions are weighed harder, especially the ones with a great wrong probability value. This is accomplished by the calculating the logarithm of the predicted values. Since the logarithm function $log(x)$ is negative for $x \in \Re and [0 > x > 1]$ and input values closer to zero mean an exponential decrease towards $-\infty$ and there is just one true label for the different classes of one pixel, if the prediction is clearly the wrong class, the cost will be exponentially higher.

$$SCCE(y_t, y_p) = -\sum_{i,j=1}^{i,j} y_{t_{i,j}} * log(y_{p_{i,j}}) \tag{5.23}$$

Here *i,j* stands for the column and row pixels of the image.

Nevertheless, our first results 5.14 made us believe that with SCCE our net was not able to learn the correct labels for the pixels. Especially the prediction after the 60th epoch shows, that the net mainly has learned to classify all body parts with the torso class. We assumed this was the case, because the liklyness of the pixel to be of class torso is higher than for example foot. Furthermore, however the accuracy graph shows descent results over 94 percent it started to flatten more and more becoming almost flat after the 50th epoch already.

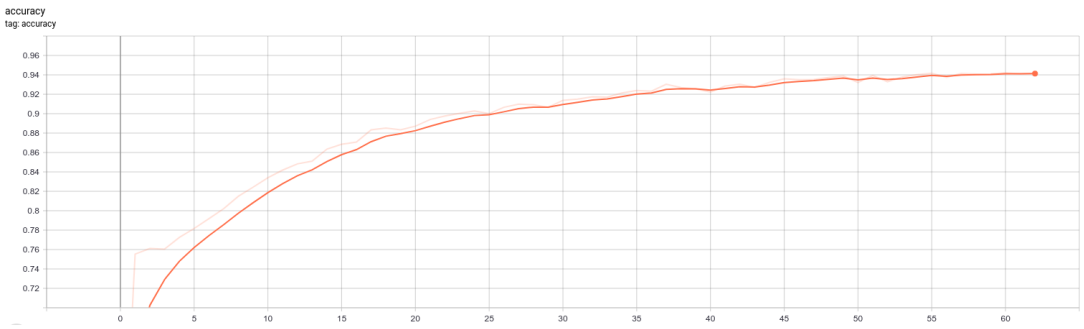

FIGURE 5.13: Accuracy for training with Sparse Categorical Cross Enntropy loss function

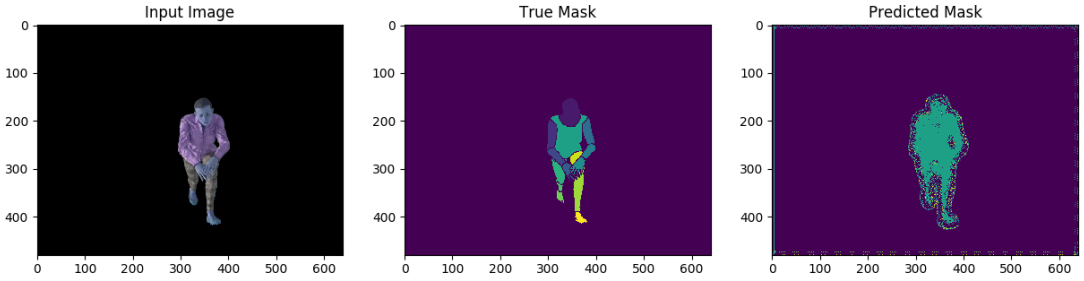

FIGURE 5.14: Predicted images for training with Sparse Categorical Cross Entropy loss function

We than created our own loss function for this class inbalance problem as explained in subsection 5.3.2. Furthermore, we reduced the amount of classes from 14 to nine accumulating left and right body parts such as foot and arm. A later experiment however shows, that our build network HRNetv3 even learns slightly better than with our own created loss function CILoss 5.1. However, this of course could be by chance due to current training settings.

5.3.2 Our custom loss function CILoss

This loss function confronts the problem of class imbalance, which especially occurs in body part recognition. The background pixels appear most often, and the different body part classes occur much more less often and they even differ a lot in their relative occurrence.

We try to confront this problem with a weighed map μ, which takes the body parts as a graph and calculates the distances from each body part b_x to all other body parts b_n, and store this data inside a table. This weighed map μ is applied to the true labels

of y_{true} so that wrong predictions further away from the true class will be punished more. For example if the network predicts hand instead of lower arm, the error will be less as if the network predicts foot.

Additionally this weight map is evened out with a multiplier to reduce the distances and facilitate the learning process for the network.

As in MSE we calculate the difference $\mathcal{E}$ between y_t and y_p, but do not square the result, we just use the absolute value. We multiply the resulting error $\mathcal{E}$ with our weighed map receiving δ. To calculate the loss we then sum the error and delta pixel-wise:

$$\mathcal{E} = y_t(x) - y_p(x)$$

$$\delta = \theta * \mu[argmax(y_t)]$$

$$L = \sum_{i=0}^{n} \mathcal{E}_i + \delta_i$$

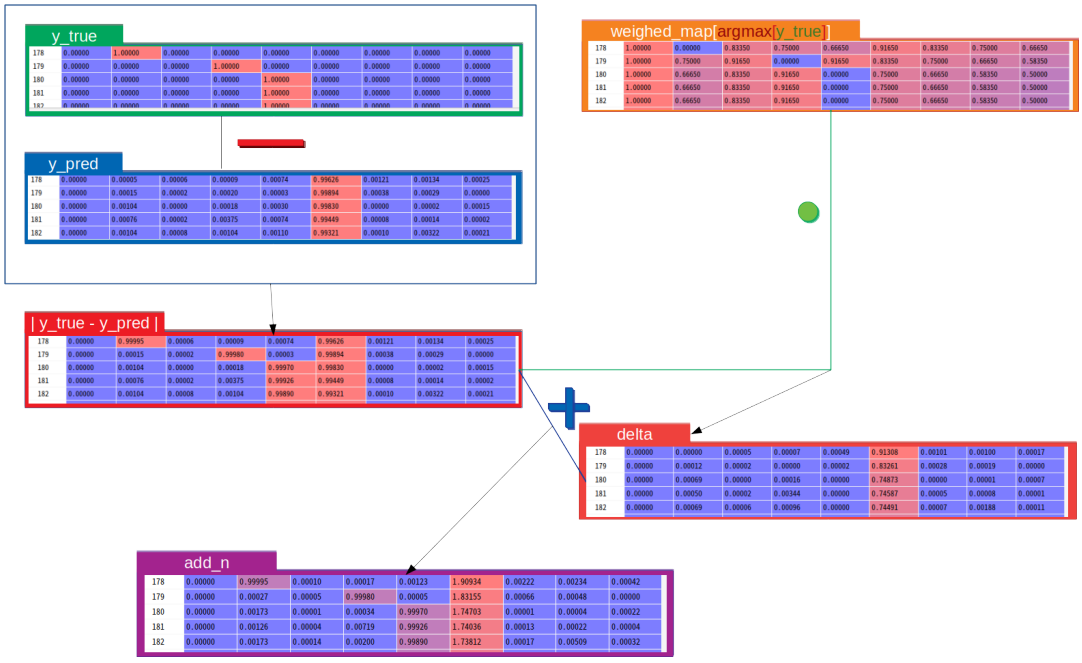

FIGURE 5.15: Visualization of our custom loss function CILoss

Chapter 6

Conclusion and future thoughts

Inspired by popular ressearch from OpenPose, VidePose3d or Simulation of figure skating [5, 34, 63] which all have problems to correctly predict keypoints for spins in figure ice skating, we covered several analytical aspects such as motion capture, network architectures and the analysis of optimizer and loss functions.

We successfully created a new high resolution fully convolutional neural network HRNetv3, which includes state-of-the-art research findings from i.a. HRNet, HRNetV3, MobileNet [21, 46, 57].

Our architecture succeeded in predicting keypoints by learning from the synthetic dataset 3DPeople [35]. In sum we have built three modules for background extraction, human parts detection and keypoint recognition. These modules partly correspond to concepts used in other state-of-the-art research such as OpenPose, however our single ice skating domain specific background extraction module stands out other architectures.

For our applied research we have created a Python 3.7 project with Tensorflow 2 [50] as core library. We conducted our experiments in multiple Docker containers with the Tensorflow:*latest-devel-gpu* as base image [49] Moreover, we have spend lots of efforts to write good readable code with a decent OO-style following the SOLID principle. So our Github repository, which we plan to make publicly available, might help to promote further research in this topic. Especially, our feature driven approach making use of Github's feature pull request style and the formulation of proper commits helped us to iteratively improve our project architecture.

Since the data is one of the main aspects deciding weather a neural network learns the desired predictions, we looked into motion capturing as well, since we could not find an approriate figure ice skating dataset for keypoint recognition. We took some captures with te inertial motion capture set from XSens. The setup felt very time-consuming and the cables all over the body connected to the battery and sensors felt very uncomfortable for recording on the ice. Additionally, multiple callibrations had to be done, probably because the system was not prepared for sliding movements. However, after the imposing capture, the integration into Blender and use of MakeHuman for creating a synthetic dataset became very promising. We especially value the diversity of data or video sequences that are possible from just one motion capture recording, which is why we are absolutely convinced that this is the way to gain decent data for an artistic sport such as figure ice skating. Yet, we think that in comparison to 3DPeople where random background images are put behind the person in action, even better results would be possible if an ice arena simulation was added to blender producing even more realistic scenes. Another caveat we encountered when looking closely at the spins from figure ice skaters winning world championchips such as Evgenia Medvedovy or Alina Zagitova, we observed that these spins include a high level of motion blur. Nevertheless, our data from 3DPose included only image sequences without any blurriness, which is why we believe that predictions in the wild on spins such as the Biellmann with a lot of blurriness, keypoints could not be predicted accurately.

Throughout our research we spurned further ideas on how to continue in this topic of keypoint recognition. First, creating a decent synthetic dataset for figure ice skating adding motion blurriness and improving the recorded scene in Blender. Furthermore, test other motion capturing methods as for example the Awinda set from XSens, which could be much more comfortable and faster to apply, because the sensors are wireless, and no cables restricting movements on the ice, must be delt with. Additionally, tests could be conducted with a markerless system such as Vicon [55].

Our network architecture could became more compact and better in performance by applying a grid search with multiple different parameters. Furthermore, the temporal information from a video could add additional information, allowing to faster and more efficiently predict videos as already argued in [37].

All in all the here presented research is ment to serve as foundation for further investigations towards action recognition in sports, especially artistic ones such as figure ice skating. Moreover, a very up-to-date topic on the writing of this paper are restrictions during the lock-downs of cities due to the COVID-19 (Corona) virus. Highly promising topics include as well physio therapy or feedback during fitness and dance routines.

Bibliography

[1] 2017. URL: https://base.xsens.com/hc/en-us/community/posts/115003027934-How-do-the-Xsens-suits-actually-work (visited on 05/17/2020).

[2] Iasonas Kokkinos Rĩ za Alp Güler Natalia Neverova. "DensePose: Dense Human Pose Estimation In The Wild". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2018).

[3] Sabrina Barr. "CORONAVIRUS: FROM YOGA TO BARRY'S BOOTCAMP — BEST EXERCISE CLASSES ON ZOOM, INSTAGRAM AND YOUTUBE". In: *Independent* (2020). URL: https://www.independent.co.uk/life-style/health-and-families/coronavirus-home-workout-exercise-class-yoga-dance-kids-elderly-joe-wicks-a9421126.html (visited on 05/11/2020).

[4] *Blender.today - daily art & development live streams*. URL: https://www.blender.org/ (visited on 05/17/2020).

[5] Z. Cao et al. "OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019), pp. 1–1.

[6] Daniel Castro et al. "Let's Dance: Learning From Online Dance Videos". In: (2018).

[7] Christoforos Charalambous and Anil Bharath. "A data augmentation methodology for training machine/deep learning gait recognition algorithms". In: (Oct. 2016).

[8] Y. Chen et al. "Cascaded Pyramid Network for Multi-person Pose Estimation". In: (2018), pp. 7103–7112.

[9] Dami Choi et al. "On Empirical Comparisons of Optimizers for Deep Learning". In: (2020). URL: https://openreview.net/forum?id=HygrAR4tPS.

[10] Timothy Dozat. "Incorporating Nesterov momentum into Adam". In: *ICLR Workshops* (2016).

[11] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research* 12 (July 2011), pp. 2121–2159.

[12] *Eiswelt Stuttgart - Stadt Stuttgart*. URL: https://www.stuttgart.de/eiswelt (visited on 05/17/2020).

[13] M. Fani et al. "Hockey Action Recognition via Integrated Stacked Hourglass Network". In: (2017), pp. 85–93.

[14] David Fox. "Video-based sports analytics system from SAP and Panasonic announced at IBC". In: *SVG europe* (2014). URL: https://www.svgeurope.org/blog/headlines/video-based-sports-analytics-from-sap-and-panasonic-announced-at-ibc/ (visited on 05/11/2020).

[15] Casper Hansen. *Optimizers Explained - Adam, Momentum and Stochastic Gradient Descent*. 2019. URL: https://mlfromscratch.com/optimizers-explained/#/ (visited on 06/02/2020).

[16] K. He et al. "Mask R-CNN". In: (2017), pp. 2980–2988.

[17] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. *Lecture 6a:Overview of mini-batch gradient descent*. URL: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf (visited on 06/02/2020).

[18] https://thecaptury.com/. *THE CAPTURY - Markerless Motion Capture*. 2020. URL: https://thecaptury.com/ (visited on 05/17/2020).

[19] *Human3.6M*. URL: http://vision.imar.ro/human3.6m/description.php (visited on 05/17/2020).

[20] Facebook Artificial Intelligence. *Feacebook publications with topic pose estimation*. 2020. URL: https://ai.facebook.com/results/?q=pose%20estimation&content_types[0]=publication&years[0]=2020&years[1]=2019&years[2]=2018&sort_by=relevance&view=list&page=1 (visited on 05/03/2020).

[21] Sun ke et al. "High-Resolution Representations for Labeling Pixels and Regions". In: (Apr. 2019).

[22] Diederik Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations* (Dec. 2014).

[23] S. Kreiss, L. Bertoni, and A. Alahi. "PifPaf: Composite Fields for Human Pose Estimation". In: (2019), pp. 11969–11978.

[24] Shenlan Liu et al. "FSD-10: A Dataset for Competitive Sports Content Analysis". In: (2020).

[25] Z. Liu et al. "Template Deformation-Based 3-D Reconstruction of Full Human Body Scans From Low-Cost Depth Cameras". In: *IEEE Transactions on Cybernetics* 47.3 (2017), pp. 695–708.

[26] *Make custom 3D characters for your Photoshop projects*. URL: https://www.adobe.com/products/fuse.html (visited on 05/17/2020).

[27] *MakeHuman - Open Source tool for making 3D characters*. URL: http://www.makehumancommunity.org/ (visited on 05/17/2020).

[28] *Motion Pack*. URL: https://www.mixamo.com/#/?page=1&type=Motion%2CMotionPac (visited on 05/17/2020).

[29] *MVN Animate*. URL: https://www.xsens.com/products/mvn-animate (visited on 05/17/2020).

[30] Neuromation. *What's the deal with "AI chips" in the Latest Smartphones?* 2018. URL: https://medium.com/neuromation-blog/whats-the-deal-with-ai-chips-in-the-latest-smartphones-28eb16dc9f45 (visited on 05/22/2020).

[31] Perception Neuron. *Perception Neuron Motion Capture*. URL: https://neuronmocap.com/ (visited on 05/17/2020).

[32] Takuya Ohashi, Yosuke Ikegami, and Yoshihiko Nakamura. "Synergetic Reconstruction from 2D Pose and 3D Motion for Wide-Space Multi-Person Video Motion Capture in the Wild". In: (2020).

[33] Paritosh Parmar and Brendan Tran Morris. "Learning to Score Olympic Events". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2017), pp. 76–84.

[34] D. Pavllo et al. "3D Human Pose Estimation in Video With Temporal Convolutions and Semi-Supervised Training". In: (2019), pp. 7745–7754.

[35] Albert Pumarola et al. "3DPeople: Modeling the Geometry of Dressed Humans". In: (Apr. 2019).

[36] Ning Qian. "On the momentum term in gradient descent learning algorithms". In: *Neural networks : the official journal of the International Neural Network Society* 12 (Feb. 1999), pp. 145–151. DOI: 10.1016/S0893-6080(98)00116-6.

[37] Y. Raaj et al. “Efficient Online Multi-Person 2D Pose Tracking With Recurrent Spatio-Temporal Affinity Fields”. In: (2019), pp. 4615–4623.

[38] Radical. *The Body in Motion - No suits. No hardware.* 2020. URL: https://getrad.co/ (visited on 05/17/2020).

[39] RKI. “SARS-CoV-2 Steckbrief zur Coronavirus-Krankheit-2019 (COVID-19)”. In: *Robert Koch Institut* (2020). URL: https://www.rki.de/DE/Content/InfAZ/N/Neuartiges_Coronavirus/Steckbrief.html (visited on 05/11/2020).

[40] Herbert Robbins and Sutton Monro. “A Stochastic Approximation Method”. In: *Annals of Mathematical Statistics* 22 (Sept. 1951), pp. 400–407. DOI: 10.1214/aoms/1177729586.

[41] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *LNCS* 9351 (Oct. 2015), pp. 234–241. DOI: 10.1007/978-3-319-24574-4_28.

[42] Bloomberg Press Room. “Bloomberg Sports Launches “Stats Insights,” Sports Analysis Blog”. In: (2012). URL: https://www.bloomberg.com/company/press/bloomberg-sports-launches-stats-insights-sports-analysis-blog/ (visited on 05/11/2020).

[43] Sebastian Ruder. “An overview of gradient descent optimization algorithms”. In: *Insight Centre for Data Analytics, NUI Galway* (Sept. 2016).

[44] Nikolaos Sarafianos et al. “3D Human Pose Estimation: A Review of the Literature and Analysis of Covariates”. In: *Computer Vision and Image Understanding* 152 (Sept. 2016). DOI: 10.1016/j.cviu.2016.09.002.

[45] Leonid Sigal, Alexandru Balan, and Michael Black. “HumanEva: Synchronized Video and Motion Capture Dataset and Baseline Algorithm for Evaluation of Articulated Human Motion”. In: *International Journal of Computer Vision* 87 (Mar. 2010), pp. 4–27. DOI: 10.1007/s11263-009-0273-6.

[46] Debjyoti Sinha and Mohamed El-Sharkawy. “Thin MobileNet: An Enhanced MobileNet Architecture”. In: (Oct. 2019), pp. 0280–0285. DOI: 10.1109/UEMCON47517.2019.8993089.

[47] Waqas Sultani, Chen Chen, and Mubarak Shah. “Real-World Anomaly Detection in Surveillance Videos”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), pp. 6479–6488.

[48] K. Sun et al. "Deep High-Resolution Representation Learning for Human Pose Estimation". In: (2019), pp. 5686–5696.

[49] Tensorflow. *dockerhub - tensorflow*. 2020. URL: https://hub.docker.com/r/tensorflow/tensorflow/tags (visited on 05/22/2020).

[50] Tensorflow. *TensorFlow Core*. 2020. URL: https://www.tensorflow.org/overview/ (visited on 05/22/2020).

[51] ISU International Skating Unigion. *ISU European Figure Skating 2020 - Ladies - Result*. 2020. URL: http://www.isuresults.com/results/season1920/ec2020/CAT002RS.htm (visited on 05/22/2020).

[52] International Skating Uniion. "Levels of Difficulty and Guidelines for marking Grade of Execution and Program Components, Season 2020/21". In: *Communication No. 2324: SINGLE & PAIR SKATING* (2020). URL: https://www.isu.org/inside-isu/isu-communications/communications/24332-2324-sp-levels-of-difficulty-and-guidelines-for-marking-goe-final/file.

[53] International Skating Uniion. "Scale of Values, Levels of Difficulty and Guidelines for marking Grade of Execution, season 2018/19". In: *Communication No. 2168: SINGLE & PAIR SKATING* (2018). URL: https://www.isu.org/docman-documents-links/isu-files/documents-communications/isu-communications/17142-isu-communication-2168/file.

[54] *User contributed assets*. URL: http://www.makehumancommunity.org/content/user_contributed_assets.html (visited on 05/17/2020).

[55] Vicon. *Award Winning Motion Capture Systems*. 2020. (Visited on 05/17/2020).

[56] B. Victor et al. "Continuous Video to Simple Signals for Swimming Stroke Detection with Convolutional Neural Networks". In: (2017), pp. 122–131.

[57] J. Wang et al. "Deep High-Resolution Representation Learning for Visual Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020), pp. 1–1.

[58] S. Wei et al. "Convolutional Pose Machines". In: (2016), pp. 4724–4732.

[59] "wrnchAI, BUILT FOR ALL YOUR HUMAN VISION NEEDS". In: (2020). URL: https://wrnch.ai/product (visited on 05/12/2020).

[60] B. Xiao, H. Wu, and Y. Wei . "Simple Baselines for Human Pose Estimation and Tracking". In: (2018).

[61] XSens. *Xsens 3D motion tracking*. URL: https://www.xsens.com/ (visited on 05/17/2020).

[62] C. Xu et al. "Learning to Score Figure Skating Sport Videos". In: *IEEE Transactions on Circuits and Systems for Video Technology* (2019), pp. 1–1.

[63] Ri Yu, Hwangpil Park, and J. Lee. "Figure Skating Simulation from Video". In: *Computer Graphics Forum* 38 (Oct. 2019), pp. 225–234. DOI: 10.1111/cgf.13831.

[64] Nan Zhao et al. "See your mental state from your walk: Recognizing anxiety and depression through Kinect-recorded gait data". In: *PLoS ONE* 14 (2019).