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 *36411/*

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 29, 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 physical human actions and behaviors. Especially in figure ice skating, this understanding is an indispensability. There are many difficult figures and poses, even difficult to clearly understand for the professionalized jury. This thesis presents an end-to-end approach to detect the 2D poses of a person in images and videos. The underlying architecture combines three branches: image segmentation, body part recognition, and human joint detection. The applied research reveals multiple findings regarding different training settings, optimizing functions, and motion capturing techniques for the special application scope of figure ice skating. In particular, multiple elaborated concepts are meant to further spurn research of pose recognition in artistic sports.

STUTTGART MEDIA UNIVERSITY

Abstrakt

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

Die Erkennung von Gelenken beim Menschen dient als Fundament für maschinelle Algorithmen, um menschliche Bewegungsabläufe erkennen zu können. Dieses Verständins ist vor allem im Eiskunstlauf von großer Bedeutung. Ein Sport mit vielen complexen Figuren und Posen, welche sogar teilweise für die professionelle Jurie eine Herausforderung darstellen. Die folgende Thesis zeigt einen Ende-zu-Ende Vorgang auf für die Erkennung von 2d Posen einer Person in Bild- sowie Videodaten. Dabei werden verschiedene Erkenntnisse hinsichtlich Trainingseinstellung, Optimizer Funktionen, und Motion Capture Techniken näher untersucht und ausgewertet. Insbesondere verucht diese Arbeit mit dessen Ergebnissen und Erkenntnissen weitergehende Recherche in dem Feld der Posenerkennung in artistischen Sportarten voran zu treiben.

Acknowledgements

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

Contents

Declaration of Authorship **iii**

Abstract **vii**

Acknowledgements **xi**

Glossary **xv**

1 Introduction **1**
1.1 Motivation 2
1.2 Related Work 3
1.3 Purpose and Research Question 5
1.4 Approach and Methodology 6
1.5 Scope and Limitation 6
1.6 Target Group 6
1.7 Outline 6

2 Figure Skating Pose Detection **7**
2.1 Complexity of Figures 7
2.2 Distinct Rating System 8

3 Theoretical Background **11**
3.1 Architecture of Convolutional Neural Networks 11
3.1.1 Types of convolutional layers 11
3.1.2 Pooling Layers 12
3.1.3 Fully Connected layers 13
3.1.4 Non-Linearity Layer 13
Activation Functions 13
3.1.5 Regularization 15
3.1.6 Optimization Algorithms 16
Backpropagation Process 17
Stochastic Gradient Descent (SGD) 18
Adaptive Gradients (AdaGrad) 19
Root Mean Squared Propagation (RMSProp) 19
Adaptive Moment Estimation (Adam) 20
Nesterov-accelerated Adaptive Moment Estimation (Nadam) . 21
3.1.7 Cost Functions 22
3.1.8 Concepts 23
3.2 CNN Detection Tasks 25
3.2.1 Classification 25
3.2.2 Image Segmentation 25
3.2.3 Localization of Human Joints 26

4 Dataset **27**
4.1 Figure Skating Dataset 27
4.1.1 Motion capturing techniques 27
4.1.2 XSens: Inertial motion capturing recordings on the ice rink . . . 28
4.2 Synthetic Dataset: 3DPeople 31
4.3 Data Processing 32

5 Method **35**
5.1 Training Performance (human parts) 37
5.2 Inference Runtime Analysis 38
5.3 Implementation Details 39

6 Experiments **41**
6.1 Ablation Study 41
6.1.1 Body Parts Detection Module 41
6.1.2 Joint Detection Module 46
6.2 Performance of loss functions 51
6.2.1 Our custom loss function CILoss 52

7 Conclusion and future thoughts **55**

A Bibliography **59**

Glossary

AdaGrad	Adaptive Gradients xiii, 19–21
Adam	Adaptive Moment Estimation xiii, 18, 20, 21
BGD	Batch Gradient Descent 17
CNN	Convolutional Neural Network 11, 15, 23, 24
DSC	Depthwise separable convolutions 12
ELU	Exponential Linear Unit 14
FC	Fully connected layer 13, 24
ImageNet	ImageNet Large Scale Visual Recognition Challenge 23, 24
MSE	Mean Squared Error 22, 23
Nadam	Nesterov-accelerated Adaptive Moment Estimation xiii, 18, 21, 22
NAG	Nesterov accelerated gradient 21
PReLU	Parametric Rectified Linear Unit (ReLU) 14
ReLU	Rectified Linear Unit xv, xvii, 13–15, 24
RMSProp	Root Mean Squared Propagation xiii, 19–21
SCCE	Sparse Categorical Cross Entropy 23
SELU	Scaled Exponential Linear Units 14, 15
SGD	Stochastic Gradient Descent xiii, 17, 18
VGG	Visual Geometry Group 24

List of Figures

3.1 ReLU activation function: $Y_i^l = max(0, Y_i^{l-1})$ 14
3.2 HRNetV3: Prediceted Feature Maps 16

4.1 3DPeople: Woman stiff walk (labeled) 31
4.2 RGB-Classes Mappings 33

5.1 HRNetV3: Our HRNet Architecture 35
5.2 HRNetV3: Predictions from Alena Kostornaia's Steps 36
5.3 HRNetv3 Training Process: Accuracy 38
5.4 HRNetv3 Training Process: Loss 38

6.1 Ablation Human Parts Module: Accuracy 44
6.2 Ablation Human Parts Module: Correct Pixel 44
6.3 Ablation Human Parts Module: Correctness Ratio 45
6.4 Ablation Human Parts Module: 1st Episode Predictions 45
6.5 Ablation Human Parts Module: 20th Episode Predictions 46
6.6 Ablation Keypoints Detection Module: Predicted Training Images . . . 48
6.7 Ablation Keypoints Detection Module: Loss FCN 49
6.8 Ablation Keypoints Detection Module: Loss Dense Layers 49
6.9 Experiments: Learning Rate SGD 49
6.10 Experiments: Accuracy 50
6.11 Experiments: Loss curves 51
6.12 Loss Functions SCCE: Accuracy 52
6.13 Loss Functions SCCE: predictions 52
6.14 Loss Functions CILoss: Calculation 54

List of Tables

6.1 Ablation Human Parts Module 42
6.2 Ablation Keypoint Module 46

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 *Big Five* 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, 18, 23, 41]. A company in Canada *wrnch.ai* even specialized on keypoint recognition from image and video data with a lot of product options [71]. Furthermore, many enterprises are becoming more and more interested in Sports Content Analysis (SPA) e.g. *Bloomberg, SAP*, and *Panasonic*, just naming a few [16, 49].

But how did this topic get into such a demanded focal point? One of the reasons are the various application areas to which pose estimation can be applied to. Main fields are sports, visual surveillance, autonomous driving, entertainment, health care and robotics [40, 54, 77]. 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 in stealing crimes in stores [58].

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 [46]. 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 [4]. 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 everyday activities not including complex poses, which can be encountered in professional sports. 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. Team sports, such as Basketball or soccer, are interested in predictions about how the other team behaves during the game and which would be the best reaction to their behavior 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. In fact, people often complain scoring is not executed fairly [34, 59, 61].

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

1.1 Motivation

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. Indeed, 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, the 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. In recent years, some competitions were even canceled, because they were not able to find the inevitable jury. In addition, sitting all day in the cold ice rink for only a very low salary is 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 on 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 appropriate 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* [6, 41].

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 to 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, people often can not be detected. However, when a person is correctly detected, it is said that accuracy would be higher [39].

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 to 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 [18]. Other famous *top-down* approaches include *Simple Baselines*, *the Cascaded Pyramid Network* or *Deep High-Resolution Learning* [9, 55, 72].

One of the most discussed and popular *bottom-up* approaches as of today is *OpenPose*. Their neural net predicts vector fields for the joint connections, which they call *Part Affinity Fields (PAF)*. Additionally, it estimates candidate keypoint locations via Gaussian distributions. These they refer to as *Part Confidence Maps*. From these detections, they refer to the associated 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 sufficient hardware equipment, this would even show decent results in realtime [6].
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. Besides, they receive highly competitive results on the *PoseTrack* challenges [44]. Some other famous *bottom-up* approaches include *Convolutional Pose Machines* and *PifPaf* [28, 70].

The COCO, MPII and PoseTrack challenges lead to several studies in the pose estimation field. Nevertheless, their dataset targets rather simple daily activities. There have been only conducted a few studies on competitive sports such as basketball, ice-hockey or swimming [15, 39, 67]. 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 [7, 30, 74, 75].
Studies on figure ice skating topics encounter three main problems: The first is domain knowledge. In C. Xu et Al.'s research, they try to predict the technical and performance scores from video data with only the ice skating program as video input and the judge scores as labels [74]. This will very likely not result in useful results since first, the figure ice 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 tends to be biased. This is one of the main controversies in the figure ice skating fairness of program judgment. Another problem is the missing dataset. There didn't exist a dataset with joint labeling until FSD-10 [30], 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 [72].

1.3 Purpose and Research Question

Our goal was to find a way on how to detect human poses in single figure ice 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.

1.4 Approach and Methodology

1.5 Scope and Limitation

1.6 Target Group

1.7 Outline

In our work we tested the performance of *VideoPose3d, OpenPose,* and *wrnch.ai* on figure ice skating elements, but the result showed a lot of failed frames, especially spins, such as the artistic Biellmann spin. This motivated us to further look into datasets, dataset creation and suitable neural network architectures for labeling figure ice skating videos, having in mind a good performing architecture, which would be able to run on mobile phones.

In this regard, 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 in chapter 4. In detail, we evaluated the creation of a dataset from these motion capturing data with the help of Blender and Makehuman in subsection 4.1.1. We created an end-to-end fully convolutional architecture consisting of three modules for background extraction, body part and keypoint detection in chapter 5. Since our work concentrates on pose estimation in single figure ice 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, which we demonstrate in chapter 6. At the end, we elaborate several concept, where these studies should continue to in chapter 7.

Chapter 2

Figure Skating Pose Detection

2.1 Complexity of Figures

Figure ice skating includes very special movement sequences that 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 keypoints for 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. In fact, 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, whether the jump should get an edge deduction. All the jumps can include a different amount of rotation in the air, whereas four rotations were 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 position.

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 may look the same, there are many different metrics and components. Moreover, each skater has their own style and performs these elements slightly different. This is what makes it so hard for machine-learning algorithms to correctly predict the elements. Which is why it is important that the basis, the key-point recognition module, has to predict the poses correctly. Because otherwise the action recognition module has no chance to make correct estimations [24, 63].

2.2 Distinct Rating System

First reported figure ice skating competitions with a registered scoring system took place in Vienna in 1881. The International Skating Union (ISU) regulates figure ice skating competitions since 1892 and is responsible for the construction and further development of its judging system. The old, still very well known judging system *6.0* was officially used until the world championships in 2004, and is still used very rarely at tiny competitions or fun competitions, due to no special technical equipment requirements. This system allows grades from 0-6 with 6 standing for best outstanding skating. Further refinements could be rated via decimal numbers. A program receives grades *A* and *B*, with the A-Grade rating technical skills and the B-Grade judging performance gratitude and expression. Both grades combined resulted in the placement of the skater, where placement proposals of the judges where combined via the majority principle, resulting in the final placement.

Because of a rising number of complaints [34], a new system was developed to increase fairness, today referenced as ISU Judging System. This new system was adjusted several times since it's first introduction in 2004, and only recently in 2019 got updated fundamentally in it's GOE scores. It assigns a base value to every jump, step sequence, or spin type in the still existing technical *A-Grade* part. Furthermore, several features rise the GOE for jumps, as for example lifted arms or the base value as for spins. The step sequences as well include special regulations to achieve certain levels and gain a higher base score. These base scores can become higher or lower via deductions from the grade of execution. For example, a fallen jump gets its jump value with a GOE of -5.
The *B-Grade* judges the skating skills, program transitions, performance or execution, choreography and interpretation. Also costume choice and interpretation of music play part in this grade. Great about the new system is, that the programs are recorded via video, and the technical specialists can review, slow down and zoom into the element. The GOEs then are given by separate judges.

Another benefit of this new system is the comparability between competitions and the transparent display of how a certain final score was achieved. Skaters can even collect points from several competitions to get special sponsorships as for example from the squad.

One drawback however is the complexity of the new judging system. Moreover, it requires special technical equipment calculating all the points and for reviewing the programs. Additionally today, all programs have very specific rules, which elements are allowed or have to be performed. Which is why some people complain about the resemblance of the different programs and the missing own interpretation freedom.

Even when, the traceability of scores rose, there is still a very high degree of subjectivity in the results, about which many people complain. This starts from the first stage of the technical specialist rating a jump with certain deductions or not. Then these subjective results are given to the judges, who pass in their GOE estimations for the elements. The *B-Grade* still remains very subjective and thus unfairness is a topic at almost every competition [24, 34, 63].

Chapter 3

Theoretical Background

- How to choose the correct dataset?
- Which base architecture does learn human keypoints?
- How can Neural Networks understand image- and video data?
- How to optimize the inference of neural networks?

3.1 Architecture of Convolutional Neural Networks

Convolutional Neural Networks (CNN) plural help machines to understand image and video data. The main building blocks of CNNs are convolutional, pooling and fully connected layers.

3.1.1 Types of convolutional layers

The Convolutional Layers learn to detect specific features in the image data. They do this with learned filters and activation functions presented in **??**. The learning process will be deeper discussed in subsection 3.1.6 and subsection 3.1.7. Via the stride s and kernel size k the feature maps can be reduced in size resulting in learning coarser features. Adding padding p accordingly this decrease in size can be countervailed. Given an input of size I and filter of size F the size of the output O can be calculated as follows:

$$O_{conv} = \frac{I + 2p - F}{s} + 1 \tag{3.1}$$

On the other hand, transposed convolution increases the size of the feature map. It implements convolution with a modified input slice or convolutional filter, where this modification leads to an increase of the feature map sizes [76]. Depthwise separable convolutions (DSC)s combine depthwise and 1x1 pointwise convolutions, which can reduce the calculation efforts and increase training and inference performance. Having an input layer with three channels and a 3x3 kernel, the kernel is applied to each channel separately depthwise on the input. Now, the scalars on these three 3x3x1 filters are calculated and stacked together. Afterwards, pointwise convolution is applied to each pixel of all channels. The amount of times this 1x1 convolution is conducted on the input accounts for the number of output channels. As an easy example, the kernel is presumed to be able to move 4x4 times across the image and expand the input to 256 output channels. This would lead to the following calculation efforts for the original convolution $256x3x3x3x4x4 = 110.592$. When using the depthwise convolution for the three input channels the 3x3x1 kernel is moved 4x4 times with $3x3x3x1x4x4 = 432$ calculations. The pointwise convolution is applied 256 times according to the required output channels resulting in $256x1x1x3x4x4 = 12.288$ calculations. Summing up both calculations this gives $432 + 12.288 = 12.720$, which means 11 times less calculations are needed for the depthwise convolution. Replacing convolutional layers with DSC can speed up performance [65]. One network which intensively makes use of this trait, for example, is the MobileNet [53].

3.1.2 Pooling Layers

Pooling layers can be used to increase the receptive field of deeper neurons and reduce the size of the filters thus save calculation effort and speedup performance. As for the convolution, the pooling operation is applied on the input in regard to s. The size of each grid on which the pooling operation is calculated on the input slice is defined by the kernel k. In contrast to the convolutional layers, no filters are learned. The output size O of the pooling operation can be calculated as follows:

$$O_{pool} = \frac{I - F}{s} + 1 \tag{3.2}$$

There are three very common pooling operations:

- **Max pooling**: the maximum of each of the receptive fields is chosen.
- **Average pooling**: the average of each grid is calculated

- **Global Average Pooling (GAP)**: the average is calculated on the whole feature map with $I = F$

Pooling operations are usually combined with convolutions [65].

3.1.3 Fully Connected layers

Fully connected layers (FC) connect all input pixels with each output. In a convolutional network it can be used to refer the target labels from the learned features of the final convolutional or pooling layers of the network. Classification is one typical application for these layers, which will be further explained in subsection 3.2.1.

3.1.4 Non-Linearity Layer

In the non-linearity layer an activation function f performs an element-wise operation on the input feature map $Y_i^l - 1$ by creating the output activation map Y_i^l:

$$Y_i^l = f(Y_i^l - 1) \tag{3.3}$$

with,

$$Y_i^l \in \mathbb{R}^{m_1^l \cdot m_2^l \cdot m_3^l} \tag{3.4}$$

$$Y_i^l - 1 \in \mathbb{R}^{m_1^{l-1} \cdot m_2^{l-1} \cdot m_3^{l-1}} \tag{3.5}$$

$$m_1^l = m_1^{l-1} \wedge m_2^l = m_2^{l-1} \wedge m_3^l = m_3^{l-1} \tag{3.6}$$

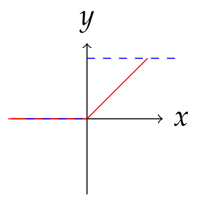
Where m_1 is the number of two dimensional feature maps or filters in convolutional layer $l - 1$ with size $m_2 x m_3$. As the calculus suggests, this layer is combined with a convolutional layer, where the filters ought to be learned [60]. In the following commonly used activation functions for convolutional networks will be presented.

Activation Functions

ReLU is a very popular activation function applied in the non-linearity layer. With a threshhold at zero it is a piecewise linear function:

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

FIGURE 3.1: ReLU activation function: $Y_i^l = max(0, Y_i^{l-1})$



$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \end{cases}$$

Reasons for its popularity are first the efficient propagation of the gradient, which helps with vanishing gradients. Second, since negative values become zero, the output volume is much more sparse, which provides robustness to small changes and helps with noisy input data. This is a great benifit for pictures, since many pixel values are not of high importance in reference to the target label. Third, the efficiency towards computation effort, thanks to the simple operations [60, 65].

However, one common problem is dying ReLUs. This happens when a weights' change is very high in Y_i^{l-1} and in the next iteration Y_i^l become very small and thus stuck at the left side of ReLU. As a result learning stagnates and weights stop adjusting properly in the backpropagation process. Some modifications open up this strict threshold and help with this problem.

As examples can be named Leaky ReLU, Parametric ReLU (PReLU), Exponential Linear Unit (ELU) and Scaled Exponential Linear Units (SELU) for easing up this threshold below zero by making the function return values between $[0; -\infty[$.

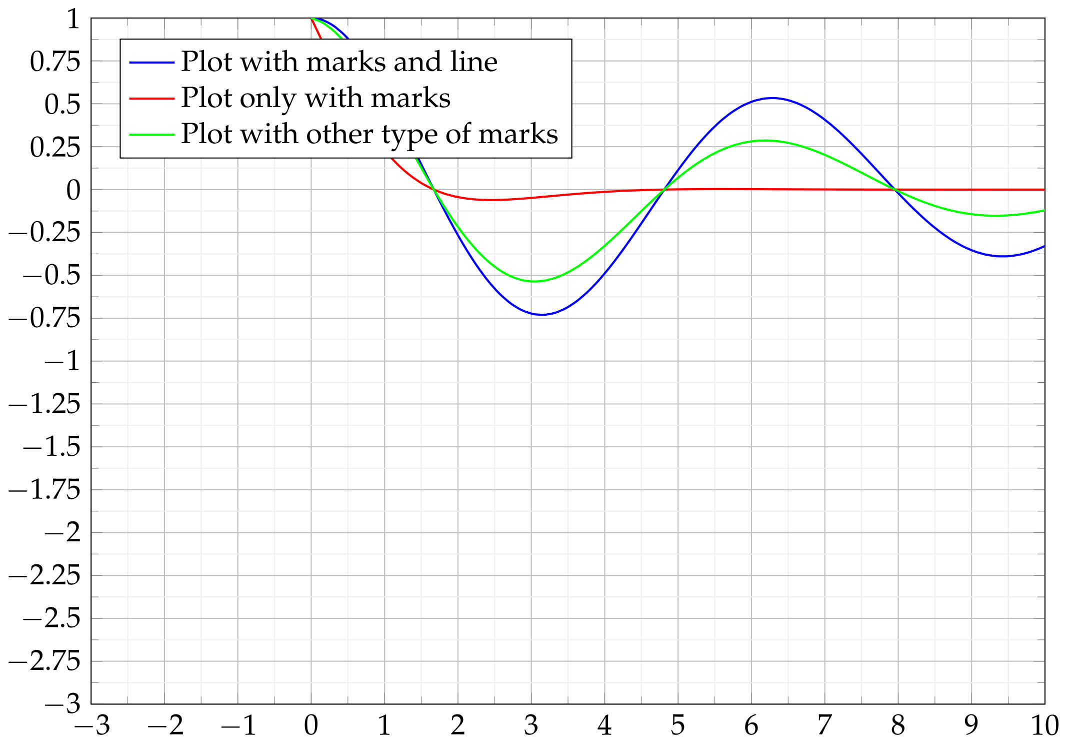
Since the presented experiments in chapter 6 make use of the SELU activation, the next section further explains this function.

In 2017 Klambauer et Al. first introduced SELUs in their research work of *Self-Normalizing Neural Networks* in which they successfully confront the named problem of dying ReLU [27]. In comparison to the prior named activation function, SELU additionally introduces normalization. Each layer preserves the mean and variance from the previous layer, since positive and negative output values are possible. These enable a shift process of the mean. An extra parameter λ scales values larger than zero. The paper by Kambauer et Al. suggests to use a value slightly above one as 1.0507. Due to the fact, that this parameter is larger than one, the gradient becomes larger than one as well, enabling the activation function to increase the variance. On the other hand a gradient very close to zero decreases variance. Altogether, SELUs confront the problem of vanishing gradients and in comparison to ReLUs they do not die.

Another activation function, which is usually used for the output layer of classification tasks in convolutional networks is **softmax**.

$$f(z_i) = \frac{exp(z_i)}{\sum_j = 1^n exp(z_j)} \tag{3.7}$$

Since this function outputs values between zero and one and the sum of z is equal one $\sum_j = 1^n exp(z_j) = 1$ it will distribute the probabilities of the output neurons according to the amount of classes. This means that this function will assign the probabilities to the given classes in a sequential order and help to transfer the target labels.



3.1.5 Regularization

Dropout and Batch normalization are commonly used regularization techniques for CNNs. With Dropout neurons are randomly dropped during training. The amount of dopped neurons is specified via the dropout rate. With this techniques the network is trained to not rely too much on the connected neurons. In fact, every mini-batch training trains a different network architecture, since some neurons are excluded.

Batch normalization is applied to the hidden layers of a network. It ensures that for every mini-batch the mean activation value is close to zero and the standard deviation close to one. With this it ensures that the activations will not explode or become to small. Furthermore, overfitting is less likely to take place [65].

3.1.6 Optimization Algorithms

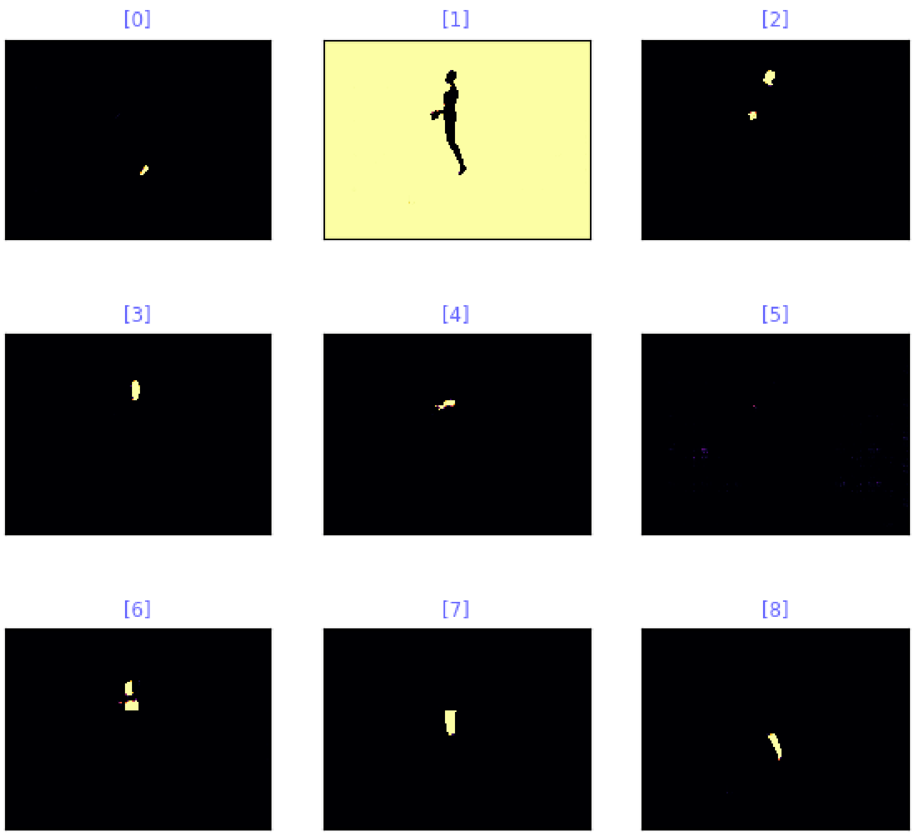


FIGURE 3.2: HRNetV3: Predicted 9 feature map classes of the output layer of the human part detection module

Optimizing the weights of a neuronal network is a crucial step in the training life cycle of deep learning. When the in chapter 5 presented 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 the head, arms, torso, and legs for the human parts detection module 3.2. With the help of a loss function, the error is then calculated for the predicted pixels in comparison to the truly labeled pixels.

Backpropagation Process

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 backwards 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 represents a local minimum [50].

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. For example, there were problems with data overflow when training was conducted with more than 12 images at a time, however, the image size was even reduced to 320x240. Another drawback is, that gradients for similar examples must be recalculated before each update, leading in redundant computations.
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 both 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 [50].

In the following section an overview of optimizers will be presented in reference to the used ones in chapter 6, which are SGD, Adam and Nadam.

Stochastic Gradient Descent (SGD)

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 [47]. Now given the parameters of a neural network θ, such as weights, biases and activation functions, here via an input image $\mathfrak{x}$ and associate 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.

$$\theta = \theta - \eta \cdot \nabla_\theta \mathcal{J}(\theta; x; y) \tag{3.8}$$

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, since 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 [17, 50].

Momentum, first proposed in 1999 by Qian et Al. [43], can help to get faster to the local minimum. This is accomplished by an additional term that is added to SGD. A constant fraction γ is multiplied with the last parameter update $\mathfrak{v}_t$ to θ.

$$\theta = \theta - \mathfrak{v}_t \tag{3.9}$$

$$\mathfrak{v}_t = \gamma \mathfrak{v}_{t-1} + \eta \cdot \nabla_\theta \mathcal{J}(\theta) \tag{3.10}$$

Since the momentum term includes all previous updates 3.10, 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 point in the same direction. When the direction changes, the term decreases, and the ball slows down. All in all, this results in faster convergence and less oscillation or jumping around the minimum [50].

Adaptive Gradients (AdaGrad)

AdaGrad, a newer algorithm proposed in 2011 by Duchi et Al. [13], 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.

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{\epsilon + \sum_{t=1}^{t} (\nabla_\theta \mathcal{J}(\nabla_{\theta_{t,i}}))^2}} \cdot \nabla_\theta \mathcal{J}(\theta_{t,i}) \tag{3.11}$$

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 denominator, which makes the learning rate shrink to an 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. [20]. The denominator takes, additionally to the past squared gradients, the current gradient at the current time step t, 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.

$$\theta_t = \theta_{t-1} - \frac{\eta}{\epsilon + E[g^2]_t} \cdot \nabla_\theta \mathcal{J}(\theta_{t-1}) \tag{3.12}$$

$$E[g^2]_t = (1-\gamma)g^2 + \gamma E[g^2]_{t-1} \tag{3.13}$$

$$g = \nabla_\theta \mathcal{J}(\theta_{t,i}) \tag{3.14}$$

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)

Adam [26], 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 [50]. It prefers flat minima.
Past decaying average m_t and past squared errors v_t estimate the first momentum (the mean) and the second momentum (the uncentered variance).

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{3.15}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \tag{3.16}$$

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:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{3.17}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{3.18}$$

This results in an update rule which reassembles RMSProp:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{3.19}$$

In particular, Adam makes use of two decay parameters β_1 and β_2, referred to 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 adapt 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 [12].

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.

$$\theta = \theta - \mathfrak{v}_t \tag{3.20}$$

$$\mathfrak{v}_t = \gamma \mathfrak{v}_{t-1} + \eta \cdot \nabla_\theta \mathcal{J}(\theta - \gamma \mathfrak{v}_{t-1}) \tag{3.21}$$

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

$$g_t = \nabla_{\theta_t} J(\theta_t) \tag{3.22}$$

$$m_t = \gamma m_{t-1} + \eta g_t \tag{3.23}$$

$$\theta_{t+1} = \theta_t - m_t \tag{3.24}$$

The 3.24 m_t term can be extended with 3.23 resulting in 3.25:

$$\theta_{t+1} = \theta_t - (\gamma m_{t-1} + \eta g_t) \tag{3.25}$$

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 to look ahead. Furthermore, expanding the term 3.19 $\hat{m}_t$ with 3.17 and 3.17 m_t with 3.15 results in the following:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}(\frac{\beta_1 m_{t-1}}{1-\beta_1^t} + \frac{(1-\beta_1)g_t}{1-\beta_1^t}) \tag{3.26}$$

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

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}(\beta_1 \hat{m}_{t-1} + \frac{(1-\beta_1)g_t}{1-\beta_1^t}) \tag{3.27}$$

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 [50]:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\epsilon + \sqrt{\hat{v}_t}} \cdot (\beta_1 \hat{m}_t + \frac{(1-\beta_1)g_t}{1-\beta_1^t}) \tag{3.28}$$

3.1.7 Cost Functions

As already mentioned in subsection 3.1.6 loss functions 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.

One classical loss function is Mean Squared Error (MSE). The error represents the difference between y_t and y_p. MSE then squares the error to even out negative results and calculates the sum of all these errors. For the fully convolutional network in the body part detection module, each pixel presents a certain class. As visualized in 3.2, nine classes are predicted. This means, there are nine labels possible for each pixel. Now, the neural network estimates with a certain probability of how likely a certain class would be true for a specific pixel. The error function then calculates the difference between the estimation or prediction and the true label, which includes a value of 1 only for the true class. Additionally to the summation of squared errors, MSE divides this sum with the number of all squares resulting in the mean value:

$$MSE(y_t, y_p) = \frac{1}{n} \cdot \sum_{[} i = 1]^n (y_t - y_p)^2 \tag{3.29}$$

Another commonly used cost function in image segmentation is Sparse Categorical Cross Entropy (SCCE), which is a common loss function used in image segmentation. It is said to help with class imbalances. Wrong predictions are weighed harder, especially the ones with a great wrong probability value. This is accomplished by 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.

$$SCCE(y_t, y_p) = -\sum_{i,j=1}^{i,j} y_{t_{i,j}} \cdot log(y_{p_{i,j}}) \tag{3.30}$$

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

3.1.8 Concepts

AlexNet, a convolutional network, which won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC, or short ImageNet) competition in 2012 set a milestone in the development of CNNs in Computer Vision. It achieved 37.5% top-1 and 17% top-5 error rates, compared to the second place which received a top-5 error of 26.2%. The ImageNet is a popular competition which "evaluates algorithms for object detection and image classification at large scale" [29] since 2010. It provides the ImageNet dataset with more than 15 million images labeled with 22 thousand classes. The ImageNet competition though uses a subset of this database. Competitors strive to receive good top-1 and top-5 error rates. The numbers of the error rates stand for the amount of occurrence of a specific label. With top-5 not being one of the model's five most likely labels.

AlexNet uses five convolutional layers with max pooling resumed by three FC layers, of which the last one is connected to a softmax which predicts the likelyness of 1000 classes. What was revolutionary about the their network was the use of ReLU in the non-linearity layer instead of tanh, which was common at that time. With this modification they were able to train six times faster. Additionally, they introduced multiple GPU training, placing half of the model's neurons on one and the other half on another GPU. This allowed training bigger models and speeding up training. Overlapping pooling leaded to an error reduction of 0.5% plus overfitting would less likely take place. Moreover, they confront the overlapping problem with data augmentation and dropout [65, 69].

Another popular finding in the history of convolutional networks is presented by the Visual Geometry Group (VGG). Same as the AlexNet, they combine convolutional layers with max pooling, and round up their network with three FCs layers and a softmax at the end. What is new though, is that they prove in their research that large filter sizes can be replaced with multiple layers of smaller filter sizes, namely factorized convolution. Before, convolutional networks had rather large receptive fields, as for example the AlexNet with 11x11. Via factorized convolution 7x7 layers can be replaced with three 3x3 layers. This results in a reduction of trainable weights and operations and thus speeds up performance. OpenPose as well makes use of these findings with their second paper on human pose recognition in image and video data [6]. The increased volume depth in deeper layers of 128, 256, 512 is still commonly used today. Their famous networks VGG-16 and VGG-19, named after the amount of weighted layers, are commonly used as benchmark or backbone of other networks [65]. As already mentioned OpenPose uses the 10 first layers of the VGG-19 network with additional fine-tuning [6].

Another discovery in the history of CNNs are residual blocks presented 2016 in the ResNet publication [19]. A residual block consists of the block itself and a skip connection from the input layer of the block to the output layer. The block includes two or three sequential convolutional layers. The first layer is a pointwise downsample layer, followed by a convolutional layer with an applied 3x3 filter, the so called bottleneck and concluded with a upsample 1x1 layer.

The special feature of this residual block is the skip connection. There, an element-wise sum is applied to the input and output layer of the block. With this, the network itself can decide whether to skip blocks, leading to a reduction of depth of the network. Moreover, the propagation of the unmodified signal and learned features of the block is closer to the original input. In Tensorflow this element-wise sum corresponds to the **tensorflow.keras.Add** layer and is used in the developed method of this thesis.

In 2014 the *Inception* network introduced a novel method to find different sized features which appear closer or further away from the observer. In their work, they split the input into parallel paths which they refer as towers. An *Inception* block starts with an input layer, several towers with convolutional and pooling layers and at the end these layers are **concatenated** in the output layer. With this, they apply different filter sizes to the paths to obtain different respective fields and learn different sized features. Similar approaches can be observed in the newer HRNet [25], however the underlying idea remains. The convolutional layers are strided down and then the filters learn features of multiple resolutions. This concept of concatenation to learn multiple features from different resolutions is as well applied in the thesis' presented method.

In the following two sections the computer vision task has to additionally keep track of the location not only the label of the found objects.

3.2 CNN Detection Tasks

3.2.1 Classification

As already mentioned, classification is the task of assigning a label to an image. An image is passed through a convolutional network with the already presented basic building blocks of convolutional layers and pooling layers. Usually the final layers are fully connected layers with a softmax activation function used for the last layer to find the labels which most probably belong to the input image.

3.2.2 Image Segmentation

Image segmentation is the pixel-wise classification of an image. The network detects for each pixel, which class it belongs to, such as person, vehicle, background etc. In the thesis presented method this pixel-wise labeling is applied to the body-part detection module. Each pixel is classified of being one of the body parts or the background class.

3.2.3 Localization of Human Joints

As already introduced in the previous chapters human pose estimation is the localization of human joints or keypoints. In particular, 2d pose estimations finds (x,y) coordinates, whereas 3d pose estimation additionally locates the depth with (x,y,z) coordinates. Nevertheless, this thesis focuses on 2d pose estimation.

DensePose [2] was one of the first proposals to initial research work in 2d pose estimation. They used the AlexNet as backbone and added a (x,y) 2k output dense layer to predict the 2k joint positions in the image.

Later studies however revealed, that the estimation of confidence maps for the joint locations works better. Therefore 2d Gaussians of constant variance are calculated as labels around the ground-truth 2d keypoint locations and the final output layer localizes these keypoints with a similar concept as presented in subsection 3.2.2 [3]. In chapter 6 the ablation study investigates both approaches.

Chapter 4

Dataset

4.1 Figure Skating Dataset

The correct dataset with the according labels, as the key factor of a neural network architecture, decides whether keypoints will be learned successfully from image- 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 ones such as FIS-V or the MIT datasets only include the total technical and performance scores of the complete performances, which are not applicable here 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 amount of videos 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.

4.1.1 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 [21, 31, 45].
When working with markers there are two sensor types: optical or inertial. Optical markers are reflective markers, which are attached to characteristic parts 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 [66].
Inertial motion capture uses IMU sensors that are attached to body parts [38, 73]. 3D positions then are calculated via multiple sensor metrics. The XSens system for example includes several trackers that contain magnetometers, gyroscopes, and accelerometers. Via their MVN Animate and Anlaze systems, the tracked data is post-processed 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 [14], 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.

4.1.2 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 with a lifetime 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 [36].

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 calibration process had to be repeated several times when the calibration failed. These problems probably came from the different gliding movements 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 inertial 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 were transmitted correctly. The post-processing from their software did work really good as well. What was disruptive, however, were the sensors and battery, which the skater had to carry. This made the movements feel constrained. Indeed, in figure ice skating falls are very 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 [38]. These products would very likely better fit for figure ice skating motion captures.

Dataset Creation 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* [33] program to create a figure ice skating avatar. Here we used the figure ice skating dress and ice skates, which are freely available from the MakeHuman community [64]. With the help of the MakeHuman plugin in Blender 2.7 [5] we then imported the created avatar from MakeHuman and retargeted 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 [8, 42, 51]. So this could improve the calculations of 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 mandatory when it comes to figure ice skating, because of the following reasons:
First, with inertial motion tracking is very accurate especially in an environment such as an ice rink when recording can be easily be done during practice sessions.
Second, much fewer 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 retargeted, 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 associate animation software or post-processing program are just more exact than 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 our own dataset was not possible due to the huge time efforts. The synthetic dataset 3DPeople as described in the next section corresponds to our expectations of an appropriate dataset.

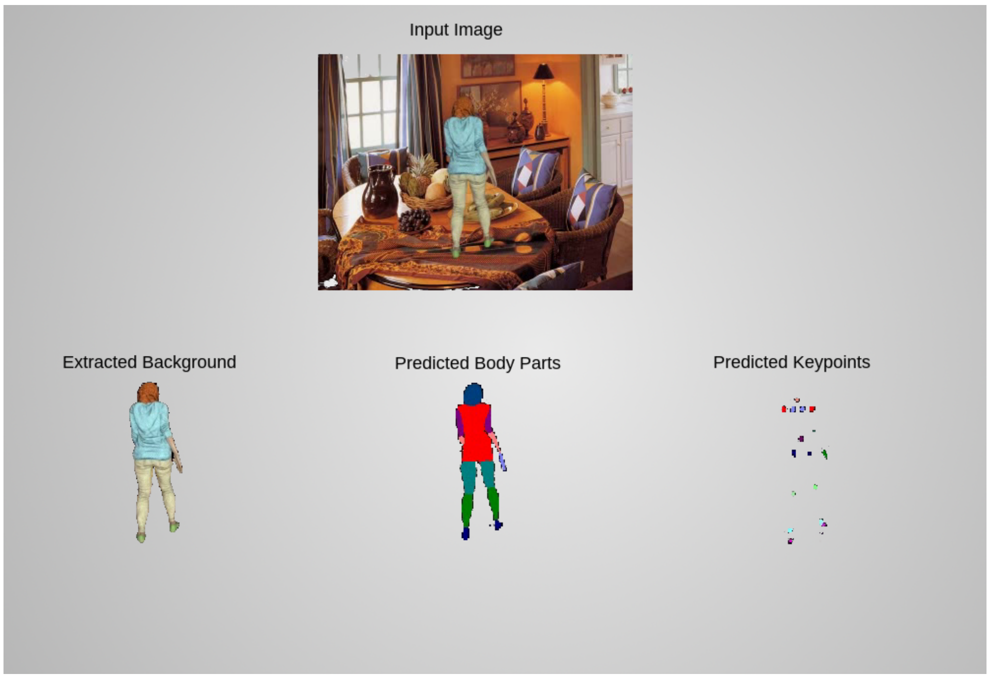


FIGURE 4.1: Learned labels from an image sequence of the 3DPeople dataset: Woman stiff walk

4.2 Synthetic Dataset: 3DPeople

The dataset 3DPeople was created via recorded motion capture files. They took 70 realistic action sequences from Mixamo [35]. The sequences contain on average 110 frames and range from usual activities with little motions such as driving or drinking to breakdance moves or backflips. These actions they then retarget onto 80 armatures which are created with MakeHuman [33] or Adobe Fuse [32]. There are 40 female and 40 male characters with plenty of variation in body shapes, skin tones, outfits, and hair. In detail, 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 varies arbitrarily 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 associate depth maps, optical flow, and semantic information such as body parts and cloth labels [42].

We found this dataset highly sophisticated 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 [52]. 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 [22, 52]. In Figure 4.1 we demonstrate the learned labels of our modules from a random image of the 3DPeople dataset.

4.3 Data Processing

The 3DPeople dataset can be requested through their project homepage and is allowed for use in research environments. 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 deletes the initial downloaded data.
We included several processing steps: In sum, we created four compressed numpy archives. Every archived file included an array with all frames of one movement sequence to allow a 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 were 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-to-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 4.2. The borders of the body-parts often contained a mixture of RGB values, which were of no use to our network.

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]
}
```

FIGURE 4.2: RGB-Classes Mappings

Chapter 5

Method

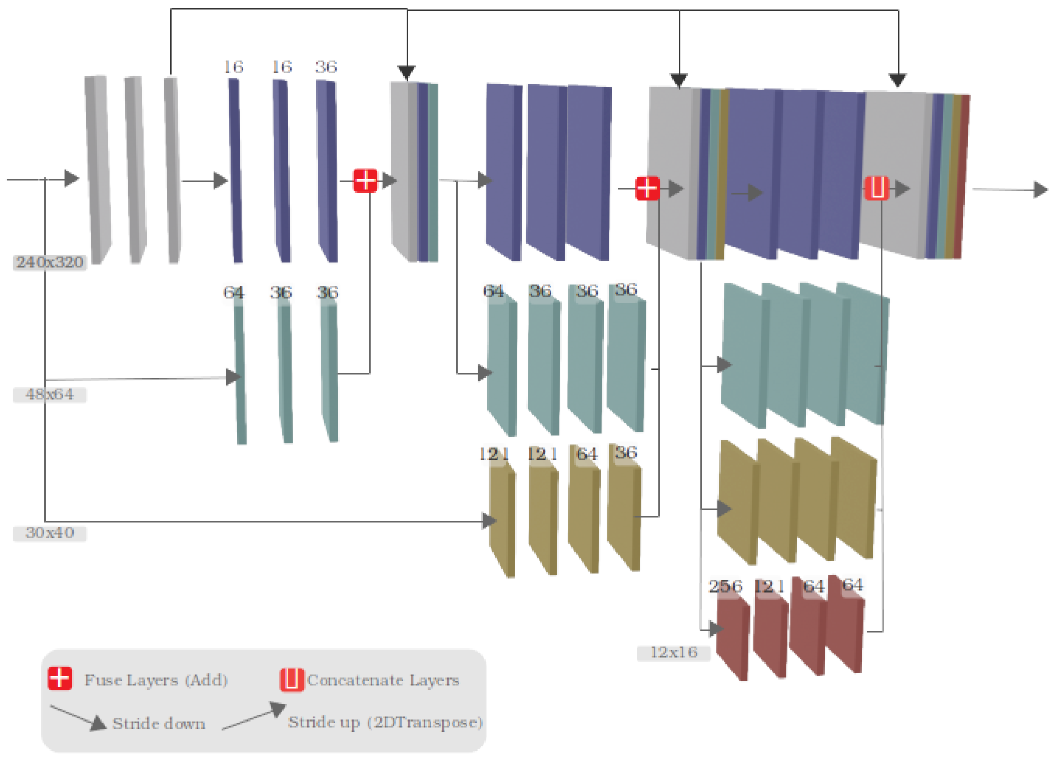


FIGURE 5.1: HRNetV3: Our created High-to-low resolution network architecture

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 background was not important for other keypoint recognition architectures such as *OpenPose* [6] or *VideoPose3D* [41], 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

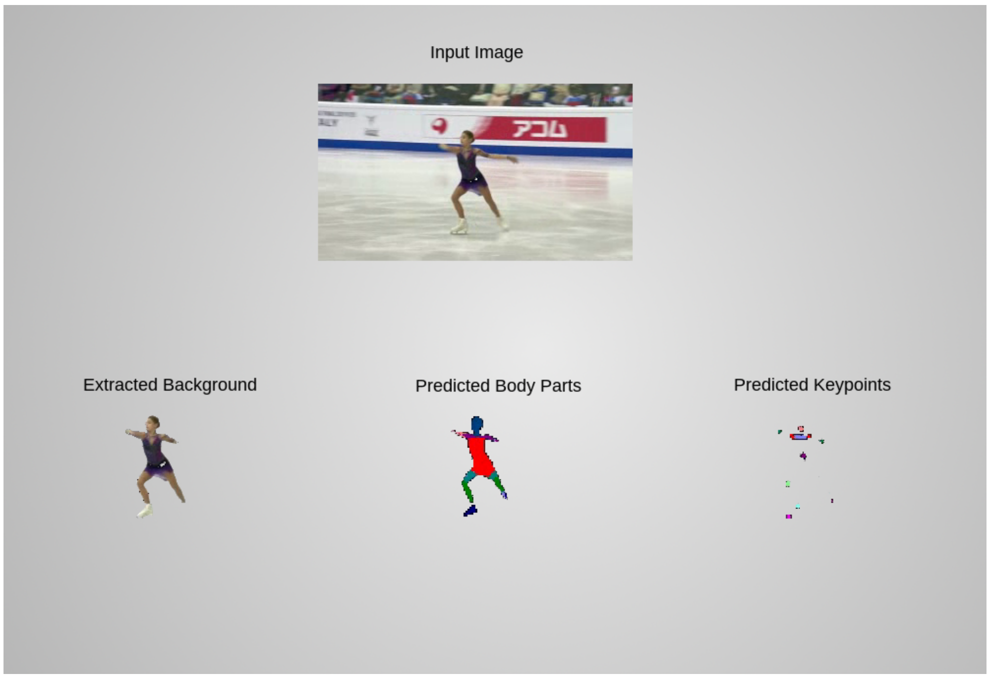


FIGURE 5.2: Learned labels by the three modules: extracted background, human part detection and keypoint detection *skater: Alena Kostornaia, 2020 European champion[62]*

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 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 pixels and a standard deviation of three. In Figure 5.2 we demonstrate an example frame labeled by our three modules showing Alena Kostornaia, the 2020 European champion during her program.

All in all, we have built 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 [25, 68].

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 associate strides $\mathtt{s}$ and kernel sizes $\mathtt{k}$, with $\mathtt{s} = \mathtt{k}$. To increase the feature maps we use transposed convolutions with again `s` = `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 [25].

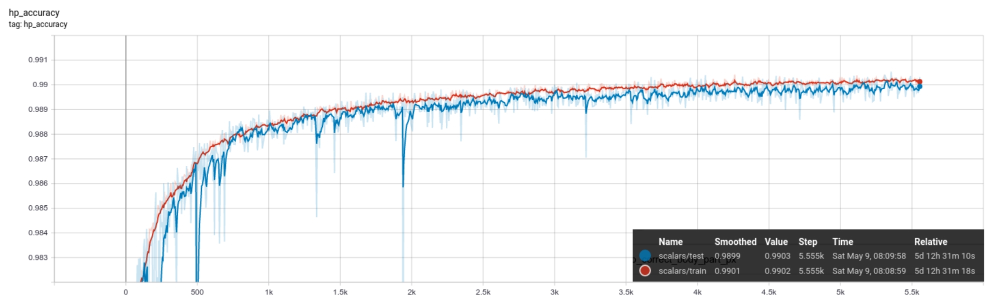
As visualized in 5.1 we adjusted the number of feature maps for the different blocks. Moreover, in the first stage, we use only three convolutional layers for one block, in the other stages we use four layers for the lower levels and only in $\mathcal{N}_L$ we use three convolutional layers for the blocks 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, which 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 function. 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 function and in the human part detection network we use a custom loss function **??** 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.

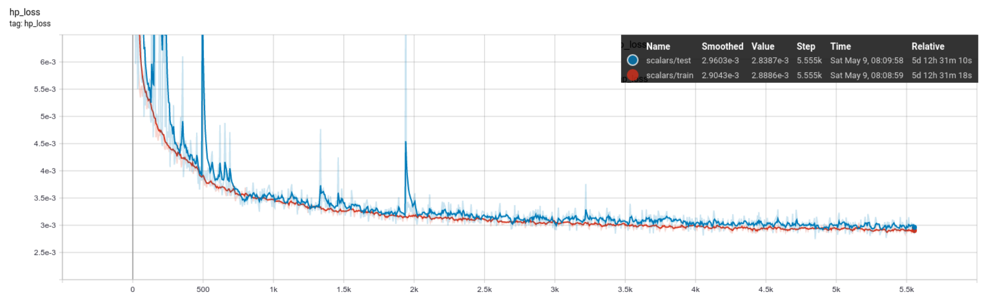
5.1 Training Performance (human parts)

We trained the human parts module with the Adam optimizer for 5556 episodes, a batch size of 3, and 64 steps per epoch. One training epoch took on average 85.43 seconds. The whole training lasted about 5.5 days. For optimization, we have built a custom loss function as explained in **??** to better deal with the class invariance of



	Name	Smoothed	Value	Step	Time	Relative
●	scalars/test	0.9899	0.9903	5.555k	Sat May 9, 08:09:58	5d 12h 31m 10s
●	scalars/train	0.9901	0.9902	5.555k	Sat May 9, 08:08:59	5d 12h 31m 18s

FIGURE 5.3: Improving Accuracy of Training HRNetV3



	Name	Smoothed	Value	Step	Time	Relative
●	scalars/test	2.9603e-3	2.8387e-3	5.555k	Sat May 9, 08:09:58	5d 12h 31m 10s
●	scalars/train	2.9043e-3	2.8886e-3	5.555k	Sat May 9, 08:08:59	5d 12h 31m 18s

FIGURE 5.4: Decreasing Loss of Training HRNetV3

the occurring pixel labels. The Figure 5.3 shows, that the accuracy of our network Figure 5.3 rises very steep until the 500th episode and then flattens more and more until the last episode, where the network reaches an excellent accuracy level of 0.99. In detail, we divided our data into 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 5.4 shows equal opposite trends to the accuracy graph. It steeply decreases until the 500th episode and then starts to flatten. Again train and test datasets do not cross each other but show similar courses.

5.2 Inference Runtime Analysis

The time to predict one frame of size 640x480 pixel takes about 0.3 seconds to 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 supported 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, it is to be expected that the inference of our network does work as well on commonly sold mobile phones today [37]. This sets apart our architecture from *OpenPose*, which we could not use for inference on the mentioned laptop.

5.3 Implementation Details

We build our Network upon the high-level API TensorFlow 2 which is based on the Keras API [57]. 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* [56] 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 6

Experiments

We conducted several experiments on our human parts and keypoint detection module. In fact, we applied the resulting network architecture from the human parts module later to the background extraction module. All in all, 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 of which one we have come up with ourselves. In the subsequent sections we will present our experimental setup, results, and constructive thoughts behind the studies.

6.1 Ablation Study

6.1.1 Body Parts Detection Module

In 6.1 we demonstrate our network architectures, which range from 88 until 177 layers and are all fully convolutional networks. For the HRNet *(traditional)* 6.1.1 we slightly deviate the HRNetV2 presented in [25] by replacing the pooling layers with strided-down and transposed convolutional layers. The amount of levels and sizes of the feature maps correspond to 5.1. 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)* 6.1.2, we adjust the filters in a way, so that higher levels use

TABLE 6.1: Ablation Human Parts Module: Network Architecture Comparison

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

fewer 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.
Subsequent architectures all make use of the adjusted filter amount.
For the HRNet *(3 stages)* 6.1.3, we only use the three first stages from 6.1.2 and omit $\mathcal{N}_{XS}$. In HRNet *(stide-down-up)* 6.1.4, 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* 6.1.5 uses the initially strided down Input layer as input for every stage instead of the concatenated results for lower levels.
As presented in the MobileNet paper [53] in 6.1.7 we experiment with depth-wise convolution and replace the Concatenation layers after each stage with Add layers.
In the UNet 6.1.8 we use all levels and combine them via concatenation according to the traditional UNet [48].
Finally, we present our resulting high-to-low resolution network HRNet *(v3)* 6.1.9, 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 5.

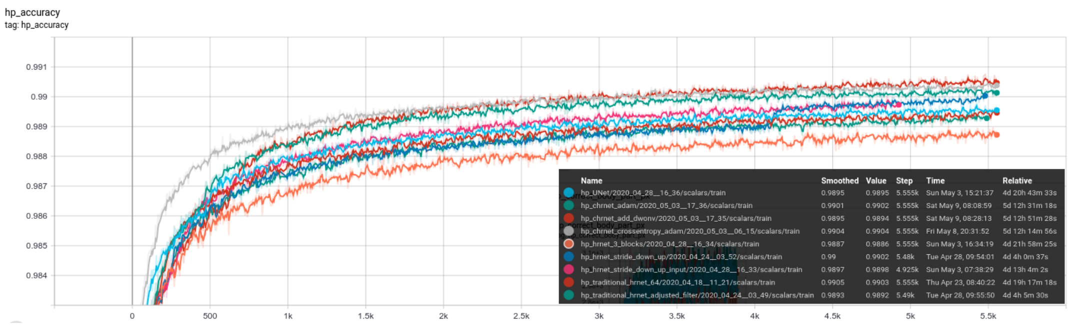
The HRNet stride-down-up 6.1.4 seems to train the fastest, when looking at 6.2 the 4.705kth step, it is only at three days and 13h closely followed by the traditional

HRNet with the adjusted filters 6.1.7 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 both trainings where conducted at the same dates, and the other dates differ. So the server could have had other workloads, 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 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. Indeed, to improve comparability, we measured the third epoch to omit starting warm up phases.

Comparison of network architectures

The U-Net 6.1.8 has the smallest amount of layers with 88 and 654,261 trainable parameters. The HRNet *stride-down-up* 6.1.4 composes most trainable parameters with 4,949,157 and 177 layers, since the input layer is first strided down and at the end transposed up again. On the other hand, due to the reduced 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.

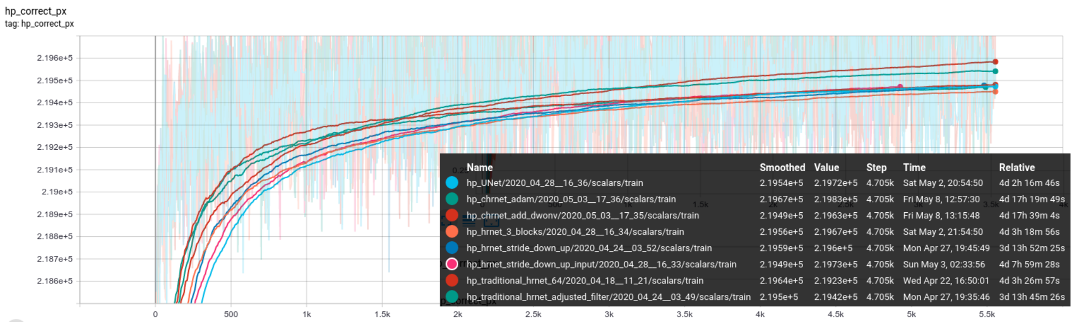
When looking at 6.4 and 6.5 this steep initial increase is visually reflected. When after the first episode the circumferences of the body shapes are predicted vaguely, after the 20th episode the networks have already learned to predict the locations of body parts relatively close. Of course, there has to be regarded, that the different predictions were made on different poses, which might differ in their complexity. Nevertheless, the UNet 6.1.8 and the HRNetV3 5.3 seem to accomplish superior performance. After the first episode, the body shape is already estimated precisely close to the input image. HRNetV3 even shows correct class estimations for different body parts such as the torso, head, arms, and legs already. In the 20th episode then the labels seem to be estimated very close to the true labels.



	Name	Smoothed	Value	Step	Time	Relative
●	hp_UNet/2020_04_28__16_36/scalars/train	0.9895	0.9895	5.555k	Sun May 3, 15:21:37	4d 20h 43m 33s
●	hp_chrnet_adam/2020_05_03__17_36/scalars/train	0.9901	0.9902	5.555k	Sat May 9, 08:08:59	5d 12h 31m 18s
●	hp_chrnet_add_dwonv/2020_05_03__17_35/scalars/train	0.9895	0.9894	5.555k	Sat May 9, 08:28:13	5d 12h 51m 28s
●	hp_chrnet_crossentropy_adam/2020_05_03__06_15/scalars/train	0.9904	0.9904	5.555k	Fri May 8, 20:31:52	5d 12h 14m 56s
●	hp_hrnet_3_blocks/2020_04_28__16_34/scalars/train	0.9887	0.9886	5.555k	Sun May 3, 16:34:19	4d 21h 58m 25s
●	hp_hrnet_stride_down_up/2020_04_24__03_52/scalars/train	0.99	0.9902	5.48k	Tue Apr 28, 09:54:01	4d 4h 0m 37s
●	hp_hrnet_stride_down_up_input/2020_04_28__16_33/scalars/train	0.9897	0.9898	4.925k	Sun May 3, 07:38:29	4d 13h 4m 2s
●	hp_traditional_hrnet_64/2020_04_18__11_21/scalars/train	0.9905	0.9903	5.555k	Thu Apr 23, 08:40:22	4d 19h 17m 18s
●	hp_traditional_hrnet_adjusted_filter/2020_04_24__03_49/scalars/train	0.9893	0.9892	5.49k	Tue Apr 28, 09:55:50	4d 4h 5m 30s

FIGURE 6.1: Ablation Human Parts Module: Accuracy Comparison

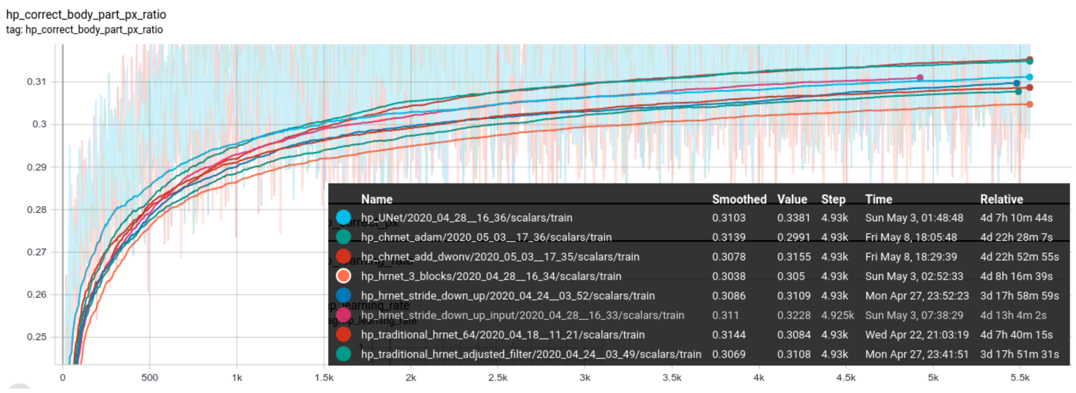
The accuracy graph shows that the HRNet *traditional* and HRNetV3 get the best results from epoch 500 onwards. At step 55.55k they reach 99 percent. However, the HRNet *stride-down-up* has a lower curve until the 4.3kth epoch, it then makes a jump in Accuracy and at the end reaches 99 percent as well.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



	Name	Smoothed	Value	Step	Time	Relative
●	hp_UNet/2020_04_28__16_36/scalars/train	2.1954e+5	2.1972e+5	4.705k	Sat May 2, 20:54:50	4d 2h 16m 46s
●	hp_chrnet_adam/2020_05_03__17_36/scalars/train	2.1967e+5	2.1983e+5	4.705k	Fri May 8, 12:57:30	4d 17h 19m 49s
●	hp_chrnet_add_dwonv/2020_05_03__17_35/scalars/train	2.1949e+5	2.1963e+5	4.705k	Fri May 8, 13:15:48	4d 17h 39m 4s
●	hp_hrnet_3_blocks/2020_04_28__16_34/scalars/train	2.1956e+5	2.1967e+5	4.705k	Sat May 2, 21:54:50	4d 3h 18m 56s
●	hp_hrnet_stride_down_up/2020_04_24__03_52/scalars/train	2.1959e+5	2.196e+5	4.705k	Mon Apr 27, 19:45:49	3d 13h 52m 25s
●	hp_hrnet_stride_down_up_input/2020_04_28__16_33/scalars/train	2.1949e+5	2.1973e+5	4.705k	Sun May 3, 02:33:56	4d 7h 59m 28s
●	hp_traditional_hrnet_64/2020_04_18__11_21/scalars/train	2.1964e+5	2.1923e+5	4.705k	Wed Apr 22, 16:50:01	4d 3h 26m 57s
●	hp_traditional_hrnet_adjusted_filter/2020_04_24__03_49/scalars/train	2.195e+5	2.1942e+5	4.705k	Mon Apr 27, 19:35:46	3d 13h 45m 26s

FIGURE 6.2: Ablation Human Parts Module: Correct Pixel

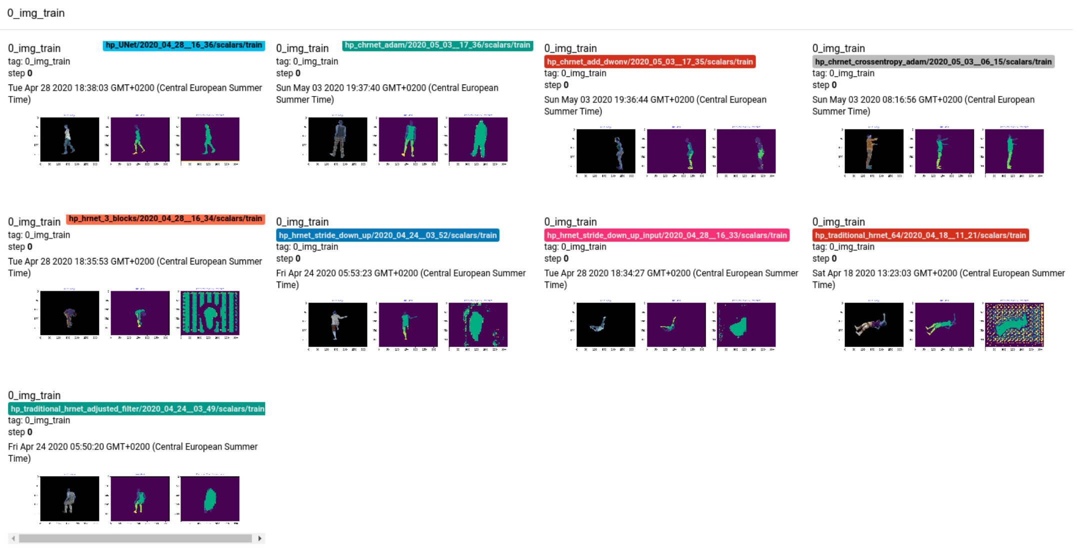
The correct pixel graph 6.2 measures how many pixels of the 240x320 image where predicted correctly in one batch. The maximum which could be reached here is $240 \times 320 \times 3 = 230\,400$. Since the single epochs vary a lot, it makes more sense to look at the smoothed results. The curves again are very close to each other, with the best results coming from the HRNetV3 with 219 670 and the HRNet *traditional* with 219 640. The lowest curve again shows the HRNet 3 *stages*.



	Name	Smoothed	Value	Step	Time	Relative
●	hp_UNet/2020_04_28__16_36/scalars/train	0.3103	0.3381	4.93k	Sun May 3, 01:48:48	4d 7h 10m 44s
●	hp_chrnet_adam/2020_05_03__17_36/scalars/train	0.3139	0.2991	4.93k	Fri May 8, 18:05:48	4d 22h 28m 7s
●	hp_chrnet_add_dwonv/2020_05_03__17_35/scalars/train	0.3078	0.3155	4.93k	Fri May 8, 18:29:39	4d 22h 52m 55s
●	hp_hrnet_3_blocks/2020_04_28__16_34/scalars/train	0.3038	0.305	4.93k	Sun May 3, 02:52:33	4d 8h 16m 39s
●	hp_hrnet_stride_down_up/2020_04_24__03_52/scalars/train	0.3086	0.3109	4.93k	Mon Apr 27, 23:52:23	3d 17h 58m 59s
●	hp_hrnet_stride_down_up_input/2020_04_28__16_33/scalars/train	0.311	0.3228	4.925k	Sun May 3, 07:38:29	4d 13h 4m 2s
●	hp_traditional_hrnet_64/2020_04_18__11_21/scalars/train	0.3144	0.3084	4.93k	Wed Apr 22, 21:03:19	4d 7h 40m 15s
●	hp_traditional_hrnet_adjusted_filter/2020_04_24__03_49/scalars/train	0.3069	0.3108	4.93k	Mon Apr 27, 23:41:51	3d 17h 51m 31s

FIGURE 6.3: Ablation Human Parts Module: Correct Pixel Ratio

For 6.3 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 variances 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 6.2. The HR-Net *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 the best results, whereas HRNet *3 blocks* performs worst with 30.38 percent at step 4.93k.


0_img_train

0_img_train hp_UNet/2020_04_28__16_36/scalars/train
tag: 0_img_train
step 0
Tue Apr 28 2020 18:38:03 GMT+0200 (Central European Summer Time)

0_img_train hp_chrnet_adam/2020_05_03__17_36/scalars/train
tag: 0_img_train
step 0
Sun May 03 2020 19:37:40 GMT+0200 (Central European Summer Time)

0_img_train
hp_chrnet_add_dwonv/2020_05_03__17_35/scalars/train
tag: 0_img_train
step 0
Sun May 03 2020 19:36:44 GMT+0200 (Central European Summer Time)

0_img_train
hp_chrnet_crossentropy_adam/2020_05_03__06_15/scalars/train
tag: 0_img_train
step 0
Sun May 03 2020 08:16:56 GMT+0200 (Central European Summer Time)

0_img_train hp_hrnet_3_blocks/2020_04_28__16_34/scalars/train
tag: 0_img_train
step 0
Tue Apr 28 2020 18:35:53 GMT+0200 (Central European Summer Time)

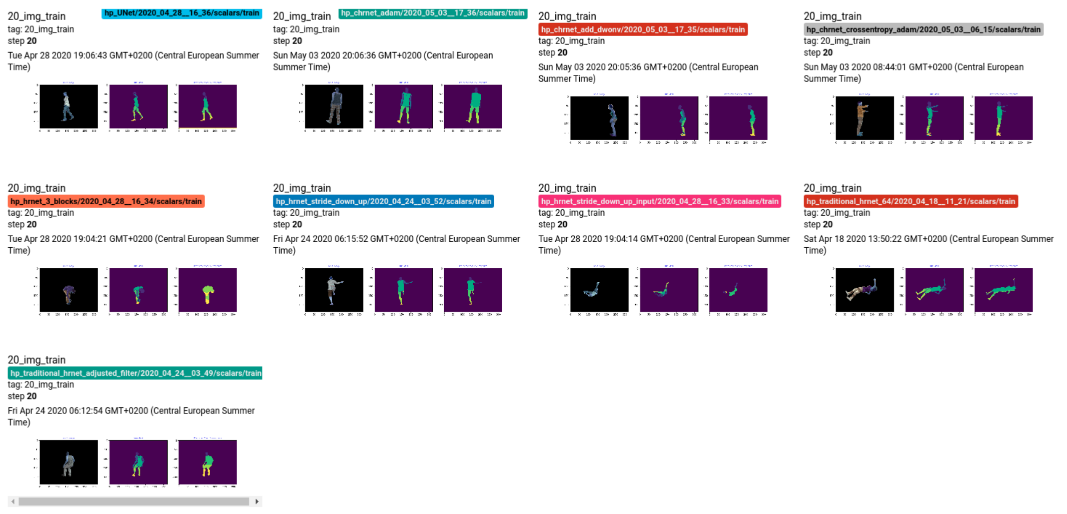
0_img_train
hp_hrnet_stride_down_up/2020_04_24__03_52/scalars/train
tag: 0_img_train
step 0
Fri Apr 24 2020 05:53:23 GMT+0200 (Central European Summer Time)

0_img_train
hp_hrnet_stride_down_up_input/2020_04_28__16_33/scalars/train
tag: 0_img_train
step 0
Tue Apr 28 2020 18:34:27 GMT+0200 (Central European Summer Time)

0_img_train
hp_traditional_hrnet_64/2020_04_18__11_21/scalars/train
tag: 0_img_train
step 0
Sat Apr 18 2020 13:23:03 GMT+0200 (Central European Summer Time)

0_img_train
hp_traditional_hrnet_adjusted_filter/2020_04_24__03_49/scalars/train
tag: 0_img_train
step 0
Fri Apr 24 2020 05:50:20 GMT+0200 (Central European Summer Time)

FIGURE 6.4: Predicted images of network architectures after first episode



20_img_train
hp_UNet/2020_04_28__16_36/scalars/train
tag: 20_img_train
step **20**
Tue Apr 28 2020 19:06:43 GMT+0200 (Central European Summer Time)

20_img_train
hp_chrnet_adam/2020_05_03__17_36/scalars/train
tag: 20_img_train
step **20**
Sun May 03 2020 20:06:36 GMT+0200 (Central European Summer Time)

20_img_train
hp_chrnet_add_dwonv/2020_05_03__17_35/scalars/train
tag: 20_img_train
step **20**
Sun May 03 2020 20:05:36 GMT+0200 (Central European Summer Time)

20_img_train
hp_chrnet_crossentropy_adam/2020_05_03__06_15/scalars/train
tag: 20_img_train
step **20**
Sun May 03 2020 08:44:01 GMT+0200 (Central European Summer Time)

20_img_train
hp_hrnet_3_blocks/2020_04_28__16_34/scalars/train
tag: 20_img_train
step **20**
Tue Apr 28 2020 19:04:21 GMT+0200 (Central European Summer Time)

20_img_train
hp_hrnet_stride_down_up/2020_04_24__03_52/scalars/train
tag: 20_img_train
step **20**
Fri Apr 24 2020 06:15:52 GMT+0200 (Central European Summer Time)

20_img_train
hp_hrnet_stride_down_up_input/2020_04_28__16_33/scalars/train
tag: 20_img_train
step **20**
Tue Apr 28 2020 19:04:14 GMT+0200 (Central European Summer Time)

20_img_train
hp_traditional_hrnet_64/2020_04_18__11_21/scalars/train
tag: 20_img_train
step **20**
Sat Apr 18 2020 13:50:22 GMT+0200 (Central European Summer Time)

20_img_train
hp_traditional_hrnet_adjusted_filter/2020_04_24__03_49/scalars/train
tag: 20_img_train
step **20**
Fri Apr 24 2020 06:12:54 GMT+0200 (Central European Summer Time)

FIGURE 6.5: 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 instance 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 is 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.

6.1.2 Joint Detection Module

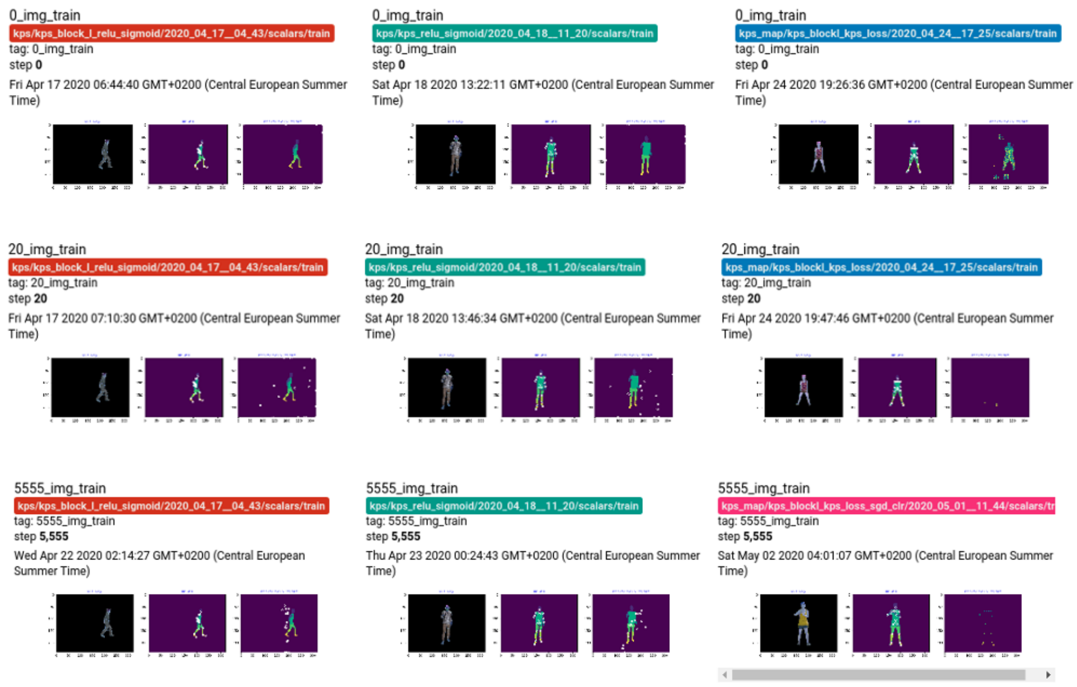
TABLE 6.2: Ablation Keypoint Detection Module: Network Architecture Comparison

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

Network Architectures

We created three variations of keypoint detection network architectures of which two**??** utilize Dense layers to estimate the keypoint locations and KPS-FCN 6.2.3 implements the third stage of our build HRNetV3.
KPS-Dense 6.2.1, 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 connected 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-squared-error.
In fact, the KPS-FCN 6.2.3 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-squared-error.

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


0_img_train
kps/kps_block_l_relu_sigmoid/2020_04_17__04_43/scalars/train
tag: 0_img_train
step **0**
Fri Apr 17 2020 06:44:40 GMT+0200 (Central European Summer Time)

0_img_train
kps/kps_relu_sigmoid/2020_04_18__11_20/scalars/train
tag: 0_img_train
step **0**
Sat Apr 18 2020 13:22:11 GMT+0200 (Central European Summer Time)

0_img_train
kps_map/kps_blockl_kps_loss/2020_04_24__17_25/scalars/train
tag: 0_img_train
step **0**
Fri Apr 24 2020 19:26:36 GMT+0200 (Central European Summer Time)

20_img_train
kps/kps_block_l_relu_sigmoid/2020_04_17__04_43/scalars/train
tag: 20_img_train
step **20**
Fri Apr 17 2020 07:10:30 GMT+0200 (Central European Summer Time)

20_img_train
kps/kps_relu_sigmoid/2020_04_18__11_20/scalars/train
tag: 20_img_train
step **20**
Sat Apr 18 2020 13:46:34 GMT+0200 (Central European Summer Time)

20_img_train
kps_map/kps_blockl_kps_loss/2020_04_24__17_25/scalars/train
tag: 20_img_train
step **20**
Fri Apr 24 2020 19:47:46 GMT+0200 (Central European Summer Time)

5555_img_train
kps/kps_block_l_relu_sigmoid/2020_04_17__04_43/scalars/train
tag: 5555_img_train
step **5,555**
Wed Apr 22 2020 02:14:27 GMT+0200 (Central European Summer Time)

5555_img_train
kps/kps_relu_sigmoid/2020_04_18__11_20/scalars/train
tag: 5555_img_train
step **5,555**
Thu Apr 23 2020 00:24:43 GMT+0200 (Central European Summer Time)

5555_img_train
kps_map/kps_blockl_kps_loss_sgd_clr/2020_05_01__11_44/scalars/tr
tag: 5555_img_train
step **5,555**
Sat May 02 2020 04:01:07 GMT+0200 (Central European Summer Time)

FIGURE 6.6: Predicted images of keypoint detection networks after first, 20th and last episode

Both figures 6.7, 6.8 show similar curves. Until the 500th episode they decrease steeply and then they start to flatten. KPS-Dense 6.2.2 has the fewest layers with 13, however KPS-FCN 6.2.3 contains only 1,109,991 trainable parameters, the dense networks train $\tilde{2}0$/ $\tilde{2}18$ 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 6.6. 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.

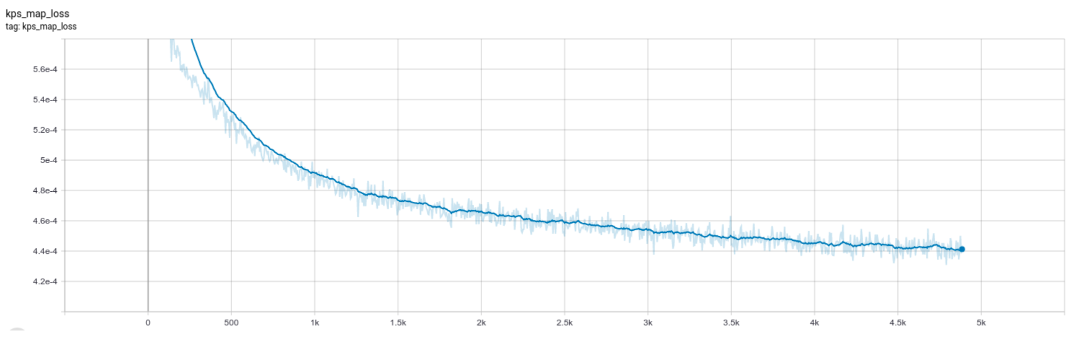


FIGURE 6.7: Ablation Keypoints Detection Module: Loss FCN

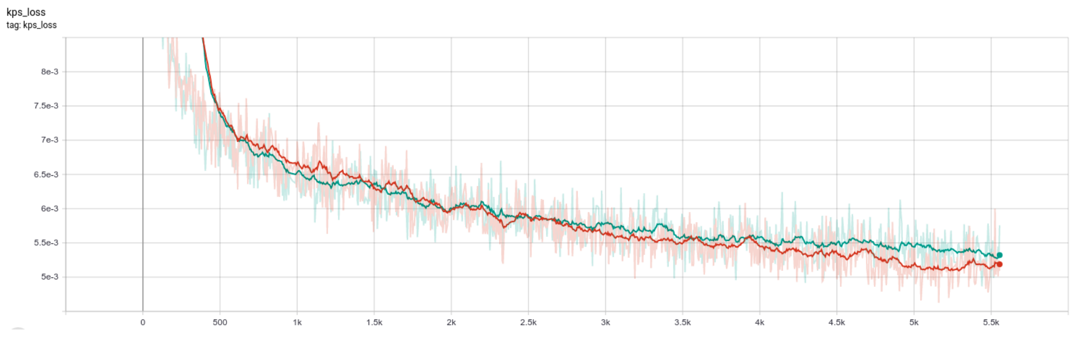


FIGURE 6.8: Ablation Keypoints Detection Module: Loss Dense Layers

Experiments with optimizers

We conducted our training with Mini-Batch Gradient Descent with a batch size of three. In addition, we will present some mathematical backgrounds of optimizer algorithms and explain further aspects of SGD in more detail.

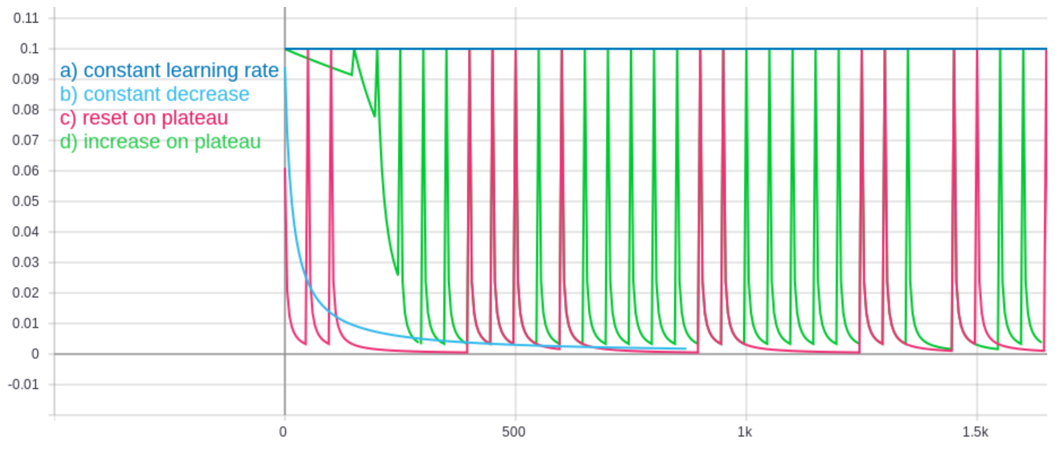


FIGURE 6.9: Experiments with Learning Rate SGD.

In our experiments we tested SGD, Adam, and Nadam all with initial 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 its influence on the training process. One SGD configuration is with a constant learning rate η, one decays with $\eta = 0.01$, and one learning rate decays 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 6.9.

Comparison of Adam, Nadam and SGD

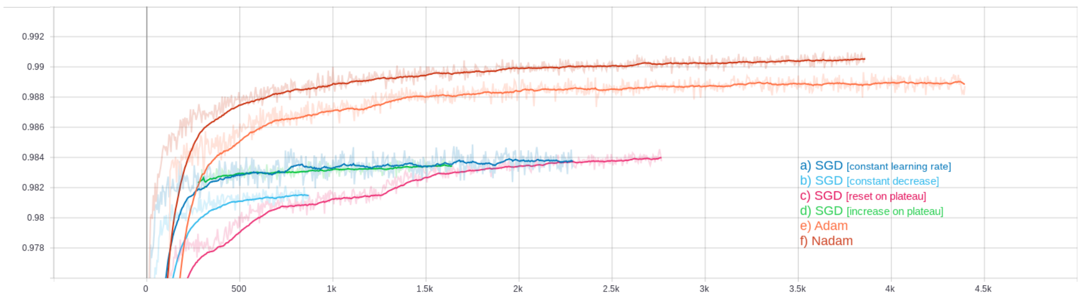


FIGURE 6.10: Accuracy of different optimizer settings

The Figure on accuracy 6.10 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 its 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.

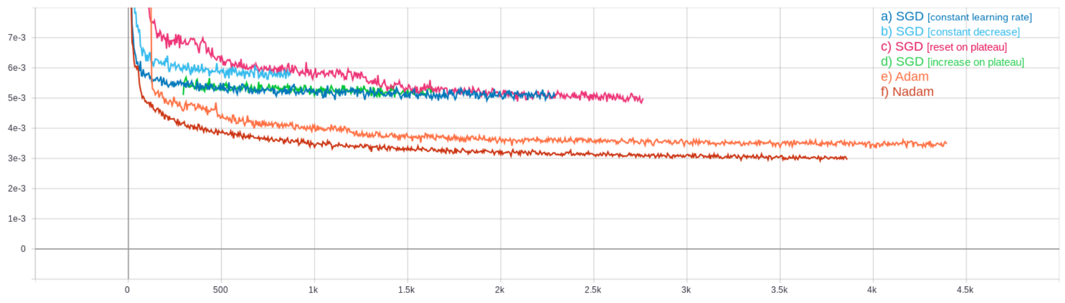


FIGURE 6.11: Experiments: Loss curves

Similar process curve trends can be observed in 6.11, while the curves are mirrored at the x-axis. This correlation of loss and accuracy trend proves the associate learning process for the different optimizers.

Conclusions

Our results are reflected by the research from Choi et Al [10]. They conducted 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.

6.2 Performance of loss functions

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

However, with our Body Part Detection Module we initially had problems with class imbalances. First, we conducted our training with Sparse Categorical Cross Entropy (SCCE)

Nevertheless, our first results 6.13 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 likeliness 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.

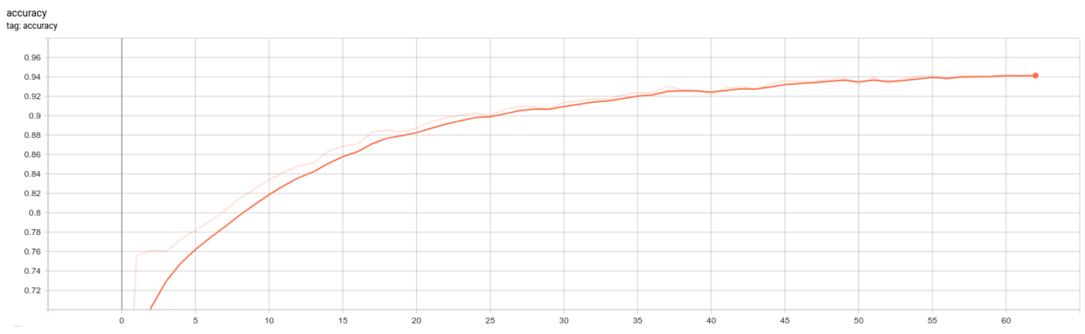


FIGURE 6.12: Accuracy for training with Sparse Categorical Cross Enntropy loss function

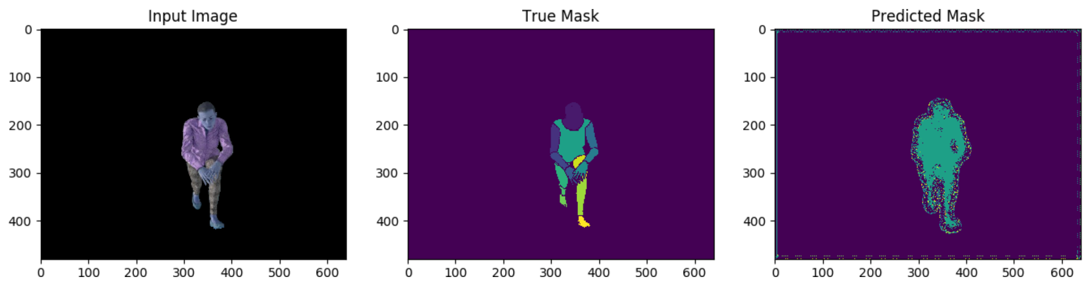


FIGURE 6.13: Predicted images for training with Sparse Categorical Cross Entropy loss function

We then created our own loss function for this class imbalance problem as explained in subsection 6.2.1. Moreover, we reduced the number of classes from 14 to nine accumulating left and right body parts such as foot and arm. A later experiment shows, on the contrary, that our build network HRNetv3 even learns slightly better than with our own created loss function CILoss 6.1. Even though, this of course could be by chance due to current training circumstances.

6.2.1 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 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.

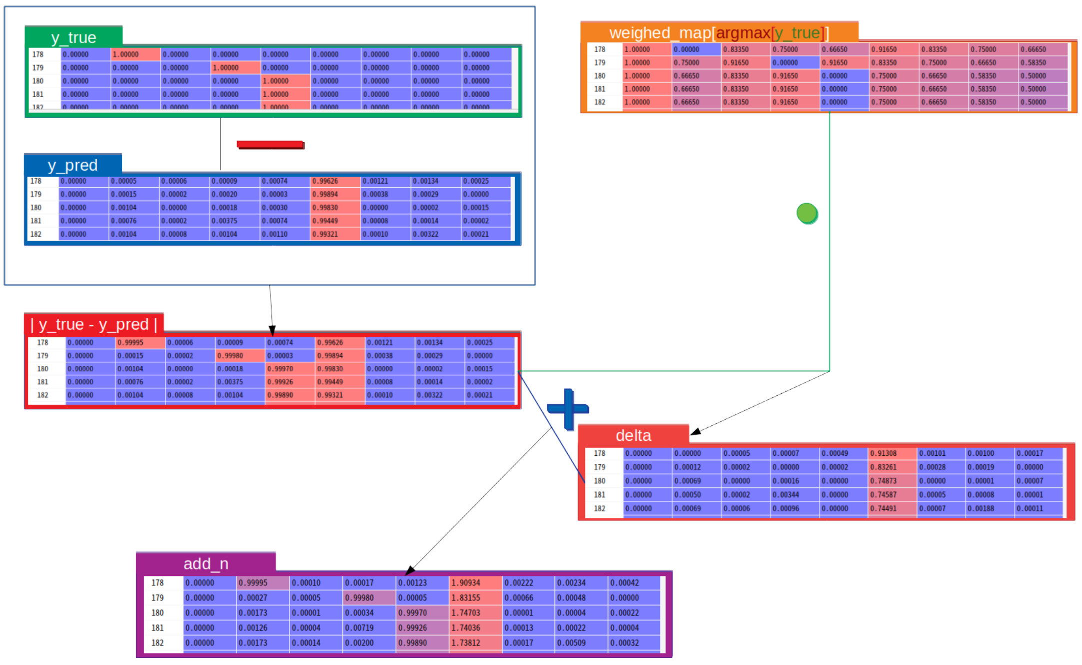
Additionally, this weight map is evened out with a multiplier to reduce the distances and facilitate the learning process for the network.

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 $\mathcal{E}$ and delta δ pixel-wise:

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

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

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



y_true

178	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
179	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000
180	0.00000	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000
181	0.00000	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000
182	0.00000	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000

y_pred

178	0.00000	0.00005	0.00006	0.00009	0.00074	0.99626	0.00121	0.00134	0.00025
179	0.00000	0.00015	0.00002	0.00020	0.00003	0.99894	0.00038	0.00029	0.00000
180	0.00000	0.00104	0.00000	0.00018	0.00030	0.99830	0.00000	0.00002	0.00015
181	0.00000	0.00076	0.00002	0.00375	0.00074	0.99449	0.00008	0.00014	0.00002
182	0.00000	0.00104	0.00008	0.00104	0.00110	0.99321	0.00010	0.00322	0.00021

weighed_map[argmax[y_true]]

178	1.00000	0.00000	0.83350	0.75000	0.66650	0.91650	0.83350	0.75000	0.66650
179	1.00000	0.75000	0.91650	0.00000	0.91650	0.83350	0.75000	0.66650	0.58350
180	1.00000	0.66650	0.83350	0.91650	0.00000	0.75000	0.66650	0.58350	0.50000
181	1.00000	0.66650	0.83350	0.91650	0.00000	0.75000	0.66650	0.58350	0.50000
182	1.00000	0.66650	0.83350	0.91650	0.00000	0.75000	0.66650	0.58350	0.50000

| y_true - y_pred |

178	0.00000	0.99995	0.00006	0.00009	0.00074	0.99626	0.00121	0.00134	0.00025
179	0.00000	0.00015	0.00002	0.99980	0.00003	0.99894	0.00038	0.00029	0.00000
180	0.00000	0.00104	0.00000	0.00018	0.99970	0.99830	0.00000	0.00002	0.00015
181	0.00000	0.00076	0.00002	0.00375	0.99926	0.99449	0.00008	0.00014	0.00002
182	0.00000	0.00104	0.00008	0.00104	0.99890	0.99321	0.00010	0.00322	0.00021

delta

178	0.00000	0.00000	0.00005	0.00007	0.00049	0.91308	0.00101	0.00100	0.00017
179	0.00000	0.00012	0.00002	0.00000	0.00002	0.83261	0.00028	0.00019	0.00000
180	0.00000	0.00069	0.00000	0.00016	0.00000	0.74873	0.00000	0.00001	0.00007
181	0.00000	0.00050	0.00002	0.00344	0.00000	0.74587	0.00005	0.00008	0.00001
182	0.00000	0.00069	0.00006	0.00096	0.00000	0.74491	0.00007	0.00188	0.00011

add_n

178	0.00000	0.99995	0.00010	0.00017	0.00123	1.90934	0.00222	0.00234	0.00042
179	0.00000	0.00027	0.00005	0.99980	0.00005	1.83155	0.00066	0.00048	0.00000
180	0.00000	0.00173	0.00001	0.00034	0.99970	1.74703	0.00001	0.00004	0.00022
181	0.00000	0.00126	0.00004	0.00719	0.99926	1.74036	0.00013	0.00022	0.00004
182	0.00000	0.00173	0.00014	0.00200	0.99890	1.73812	0.00017	0.00509	0.00032

FIGURE 6.14: Visualization of our custom loss function CILoss

Chapter 7

Conclusion and future thoughts

Inspired by popular research from *OpenPose, VidePose3d* or *Simulation of figure skating* [6, 41, 75] 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, HRNetV2, and MobileNet [25, 53, 68].

Our architecture accomplished to predict keypoints by learning from the synthetic dataset 3DPeople [42]. 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 [57] as core library. We conducted our experiments in multiple Docker containers with the Tensorflow:*latest-devel-gpu* as base image [56] In fact, we have spent 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 whether a neural network learns the desired predictions, we looked into motion capturing as well, since we could not find an appropriate figure ice skating dataset for keypoint recognition. We took some captures with the 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 calibrations had to be done, probably because the system was not prepared for sliding movements. Nevertheless, 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 championships such as Evgenia Medvedeva or Alina Zagitova [11], 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 [66].

Our network architecture could become more compact and better in performance by applying a grid search with multiple different parameters. Moreover, the temporal information from a video could add additional information, allowing to faster and more efficiently predict videos as already argued in [44].

With this here presented study we try to further spur development and research in figure ice skating to improve fairness in this sport. For example, in[75], they simulated some figure ice skating figures and translated these to 3d animation. It would be very interesting to see, how skaters were rated, if the technical specialists and judges would see the animation, without knowing, who the skater was, or what the skater looked like. This could be a huge step in fairness. In addition, the rating could be conducted remotely, saving the jury from the cold ice rink and driving efforts.
An even more advanced step to replace the necessity of jury, who are sometimes hard to get for a competition, would be another highly interesting research topic regarding action recognition. Another application would be the support during practice, with recommending actions to improve some jumps for example.

All in all, the here presented research is meant to serve as a foundation for further investigations towards action recognition in sports, especially artistic ones such as figure ice skating. Indeed, 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 physiotherapy or feedback during fitness and dance routines.

Appendix A

Bibliography

Books and Articles

[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).

[6] 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.

[7] Daniel Castro et al. “Let’s Dance: Learning From Online Dance Videos”. In: (2018).

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

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

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

[12] Timothy Dozat. “Incorporating Nesterov momentum into Adam”. In: *ICLR Workshops* (2016).

[13] 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.

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

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

[19] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: (June 2016), pp. 770–778. DOI: `10.1109/CVPR.2016.90`.

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

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

[27] Günter Klambauer et al. "Self-Normalizing Neural Networks". In: (June 2017).

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

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

[31] 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.

[39] 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).

[40] 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.

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

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

[43] 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.

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

[47] 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.

[48] 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.

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

[51] 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.

[52] 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.

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

[54] 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.

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

[63] 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.

[65] Ivan Vasilev. “Advanced Deep Learning with Python”. In: *Packt Publishing* (Dec. 2019).

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

[68] 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.

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

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

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

[74] 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.

[75] 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.

[76] M. D. Zeiler et al. “Deconvolutional networks”. In: (2010), pp. 2528–2535.

[77] 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).

Online

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

[3] Sudharshan Chandra Babu. *A 2019 guide to Human Pose Estimation with Deep Learning*. 2019. URL: https://nanonets.com/blog/human-pose-estimation-2d-guide/ (visited on 06/26/2020).

[4] Sabrina Barr. *CORONAVIRUS: FROM YOGA TO BARRY'S BOOTCAMP — BEST EXERCISE CLASSES ON ZOOM, INSTAGRAM AND YOUTUBE*. 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).

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

[11] Liz Clarke. *Russia's Alina Zagitova captures gold in women's free skate*. 2018. URL: https://www.denverpost.com/2018/02/22/alina-zagitova-captures-gold-womens-free-skate-russia/ (visited on 06/07/2020).

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

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

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

[20] 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).

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

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

[23] 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).

[24] *ISU Judging System*. URL: https://www.isu.org/isu-statutes-constitution-regulations-technical-rules-2/isu-judging-system (visited on 06/02/2020).

[29] Stanford Vision Lab. *ImageNet Large Scale Visual Recognition Challenge (ILSVRC*. 2015. URL: http://www.image-net.org/challenges/LSVRC/#:~:text=The%20ImageNet%20Large%20Scale%20Visual,image%20classification%20at%20large%20scale.&text=Another%20motivation%20is%20to%20measure,indexing%20for%20retrieval%20and%20annotation. (visited on 06/26/2020).

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

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

[34] Borzilleri Meri-Jo. *2014 Winter Olympics - Olympic Figure Skating Controversy: Judging System Is Most to Blame for Uproar*. Feb. 2014. URL: https://bleacherreport.com/articles/1969257-olympic-figure-skating-controversy-judging-system-is-most-to-blame-for-uproar (visited on 06/02/2020).

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

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

[37] 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).

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

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

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

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

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

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

[58] CFI Tom Meehan. *Using Artificial Intelligence to Catch Shoplifters in the Act*. 2019. URL: https://losspreventionmedia.com/using-artificial-intelligence-to-catch-shoplifters-in-the-act/ (visited on 06/02/2020).

[59] Olessia Tom Meehan. *Expertin bei Eislaufweltmeisterschaften: Drei entscheidende Minuten*. Dec. 2014. URL: https://www.stuttgarter-zeitung.de/inhalt.expertin-bei-eislaufmeisterschaften-drei-entscheidende-minuten.3b370419-3df9-4ea7-81ba-97d64480e78f.html (visited on 06/02/2020).

[60] Technical University Munich (TUM). *Layers of a Convolutional Neural Network*. 2017. URL: https://wiki.tum.de/display/lfdv/Layers+of+a+Convolutional+Neural+Network/ (visited on 06/25/2020).

[62] 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).

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

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

[69] Jerri Wei. *AlexNet: The Architecture that Challenged CNNs*. 2019. URL: https://towardsdatascience.com/alexnet-the-architecture-that-challenged-cnns-e406d5297951 (visited on 06/26/2020).

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

Further Bibliography

[61] Claudia Unger. *Jury work from point of view of german figure ice skating coach and technical specialist Claudia Unger*. private interview. May 2020.