Technische Universität Ilmenau
Fakultät für Informatik und Automatisierung
Fachgebiet Neuroinformatik und Kognitive Robotik

3D Person Sensing for interactive industrial process monitoring

Masterarbeit zur Erlangung des akademischen Grades Master of Science

Thomas Schnürer

Betreuer: Dr. Stefan Fuchs (Honda Research Institute Europe)
Dipl.-Inf. Markus Eisenbach (Technische Universität Ilmenau)
Verantwortlicher Hochschullehrer:
Prof. Dr. H.-M. Groß, FG Neuroinformatik und Kognitive Robotik

Die Masterarbeit wurde am 02.01.2018 bei der Fakultät für Informatik und Automatisierung der Technischen Universität Ilmenau eingereicht.

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während der Anfertigung dieser Masterarbeit unterstützt haben.
Zunächst bedanke ich mich bei Prof. Dr. H.-M. Groß sowie Dipl.-Inf. Markus Eisenbach, die diese Arbeit betreut und begutachtet haben.
Des Weiteren möchte ich mich bei den Mitarbeitern des Honda Research Institute Europe und insbesondere meinem Betreuer Dr. Stefan Fuchs bedanken, die mich mit ihrer Expertise tatkräftig unterstützt haben. Neben all den hilfreichen Kollegen gilt ein besonderer Dank den Studenten, die durch ihre Unterstützung die Aufnahme von Trainingsdaten ermöglicht haben.
Ein großes Dankeschön geht auch an Alexander Katzmann und Benjamin Lewandowski für das Korrekturlesen meiner Arbeit.

Erklärung: "'Hiermit versichere ich, dass ich diese Masterarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe. Alle von mir aus anderen Veröffentlichungen übernommenen Passagen sind als solche gekennzeichnet."'

Ilmenau, 02.01.2018 .....

Thomas Schnürer

Contents

Chapter 1

Introduction

1.1 Motivation

In modern production plants a commonly observed objective is to increase the amount of automation. Besides other strategies, this objective can be achieved by the introduction of robots into the work flow. While this approach is not new, the amount of interaction between such robots and human workers is usually either highly limited or prohibited entirely. To enable a scenario in which robots work together with humans, a number of problems need to be solved.

For example, it is essential for the robot to always be aware of the location and pose of any humans within the shared workspace. This knowledge can then be used to plan and perform interactions and avoid accidents.

One possible solution for this task is to represent a humans pose by estimating the location of their body joints. In this master thesis, such an approach for human pose estimation by body joint localization in an industrial workbench scenario will be presented.

1.2 Scenario and Objectives

The problem of Human Pose Estimation for human-machine interaction occurs in a wide variety of scenarios that can lead to different approaches and solutions. The main

focus of this thesis is to find a feasible solution for a specific industrial scenario within certain limitations. The following section will outline the scenario, its limitations and the consequential objectives for finding a feasible solution.

1.2.1 Target Scenario

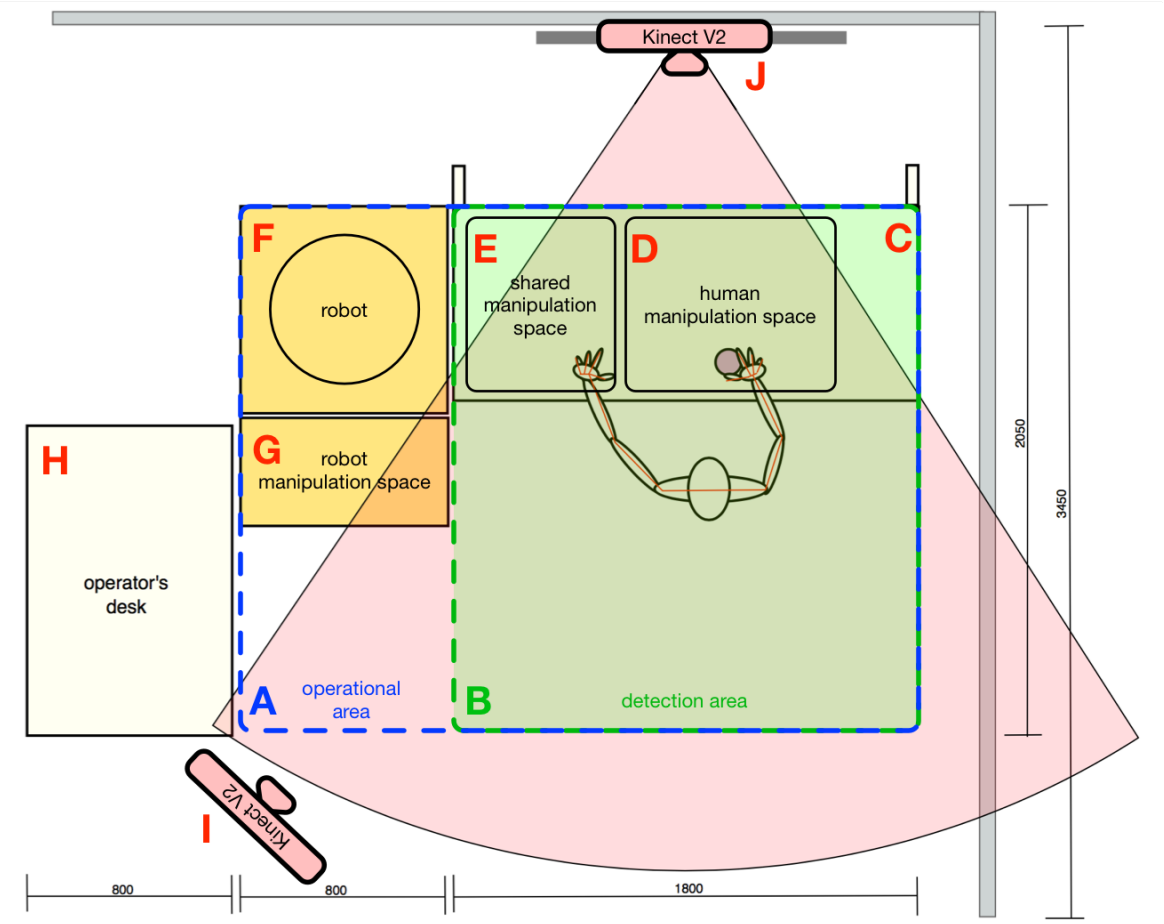


Figure 1.1: Specific industrial workbench scenario for Human Pose Estimation
*A Kinect V2 sensor (**J**) will be used to detect any pose of a human worker within the detection area (**B**) while he is performing tasks on the workbench (**C**).*

Within an industrial environment, there are several possible scenarios for human-machine interaction. Here, a scenario called "industrial workbench scenario" will be considered. It is mainly characterized by a small, fixed operational area and a single

human worker at a defined position. This worker will be performing tasks in front of a workbench or table and can be monitored by a camera at a fixed location.
The specific setup used for this master thesis is displayed in figure 1.1. The overall dimensions for the whole room are roughly $3.4 \times 3.45\,\mathrm{m}$, the operational area is $2.6 \times 2.05\,\mathrm{m}$ (**A**) and the detection area about $1.8 \times 2.05\,\mathrm{m}$ (**B**). Within the latter, one human worker will perform some simple tasks that may include the workbench (**C**) or several small objects (see sec. 1.2.2 for details on the tasks).
Next to the workbench, a stationary robot is situated (**F**). It is equipped with a mechanic arm that can be moved freely within a limited area and will be used to assist the human. For a controlled cooperation between both actors, the available operational area is split into three parts:

- A **robot manipulation space** (**G**) which will be accessed exclusively by the robot
- A **shared manipulation space** (**E**) which can be accessed by both the human worker and the robot
- A **human manipulation space** (**D**) which will be accessed exclusively by the human worker

Lastly, a desk for the operator is located outside of the operational area for supervision. The detection area can be observed by two Kinect V2 cameras (**I,J**), which are mounted at fixed positions in an effort to minimize occlusions.

The industrial partner provided this target scenario as well as the following requirements:

- Only the depth sensor of the front facing camera above the workbench (**I**) must be used for the task of Human Pose Estimation.
- The estimation must be possible at about 30 Hertz (Hz).
- The estimation must only use independent, single images (no tracking)

- The estimation must be independent of any specific background
- If possible, the estimation should build on the existing system presented by [Arenknecht, 2016]

1.2.2 Use Case Constraints and Distinctions

From the characteristics of the workbench scenario and the specifications provided by the industrial partner described above, a number of constraints were derived to narrow down the specific requirements and limitations. Those constraints are divided into five groups and describe the circumstances under which the pose estimation must work properly. The complete list of constraints can be found in appendix A.1. In short, they address the sensor, room setup, detection area specification, a person's appearance and detectable poses. Within the setup described above, all upper body joints of a typical worker must be estimated while performing some simple tasks.
Whether, and to what extend, they actually influence the performance of the resulting system will be examined in section 8.3. Ideally, only a few constraints are mandatory to ensure optimal performance.

1.2.3 Objectives

The aim of this master thesis is to estimate human joint positions within the constraints described above. Therefore, a suitable solution must fulfill the following requirements:

- **Joint Detection:** The location of all visible upper body joints (see figure 1.2) must be estimated to reconstruct the pose (preferable in 3D)
- **Real Time:** To fully exploit the camera frame rate, the detection must be possible at 30Hz
- **Accuracy:** The average detection error must be less than 10 cm per joint to ensure usability for later applications
- **Versatility:** Within the above constraints, every pose in every environment for any person must be estimated as accurate as possible

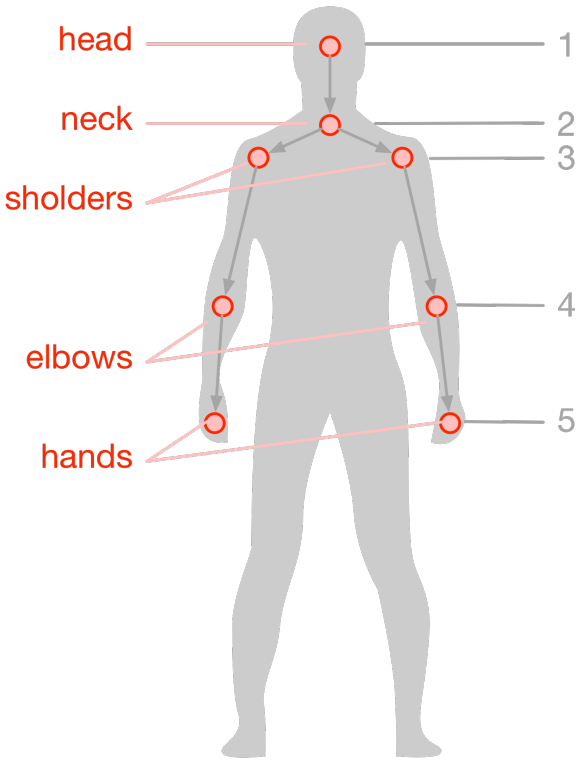


Figure 1.2: The eight upper body joints that are to be estimated
*All upper body joints (**red**) can be arranged into a kinematic chain (**grey arrows**). On the right side (**grey**), the level within this chain is shown. Joints higher up (like the head and neck) can be detected more easily than joints further down.*

1.3 Tasks and Structure

In order to reach the above stated objectives for the scenario described in section 1.2, a number of problems need to be solved. In a most general view, the overall task is to develop a system that produces pose estimates from the data of a singe depth sensor. These pose estimates consist of location estimations for every visible upper body joint (shown in figure 1.2). Since the data of a depth sensor can be represented in different ways, three main tasks can be derived from this:

- **Input Data Preprocessing:** An advantageous representation for the succeeding joint estimator must be found and implemented. This may also include feature extraction, filtering or other types of preprocessing.

- **Pose Estimator:** Based on the preprocessed input data, a system for pose estimation by body joint detection must be developed, implemented and fine-tuned. Certain types of Human Pose Estimation systems also require suitable training data.

- **Benchmark and Validation:** The finished system must be tested under different conditions and the performance must be compared to similar solutions. For this, a suitable measure of performance must be found first.

This thesis presents an approach to solve the aforementioned tasks and is structured as follows:
First, a review of the state of the art in Human Pose Estimation will be provided in chapter 2. By comparing different aspects of relevant approaches, a development strategy will eventually be derived in section 2.4. Thereafter, a brief introduction to the most important theoretical principles will be provided in chapter 3, including a metric for performance measurement in section 3.3.
Since the preceding work of [Arenknecht, 2016] should be utilized if possible, different options for this are explored and compared in chapter 4. As a part of this analysis, multiple input representations are compared in section 4.2. Besides other results, this showed that new training data was necessary. Thus the process of training data creation will described in chapter 5. Guided by the performance on this dataset, the development process of the Human Pose Estimation system will be described in chapter 6. Subsequently, chapter 7 will provide a detailed description of the final design. In chapter 8, the performance of the presented system will be analyzed and compared to state of the art systems. Finally, a brief summary and potential for future work will be given in chapter 9.

Chapter 2

State of the Art

Human Pose Estimation is a topic with a wide range of applications and therefore a wide range of approaches. Publications like [Sarafianos et al., 2016] provide some form of structured overview, but their structure is not completely sufficient for the scope of this thesis. In the following sections, a review of the state of the art in the field of Human Pose Estimation will be provided based on the structure and key characteristics displayed in figure 2.1. The most important aspects within each of the three parts will be discussed in the next three sections.

2.1 System Input

This first section considers important aspects of the input data, from data capture to feature generation and preprocessing. Specifically, benefits of multi-camera systems as well as possible ways to utilize depth data and image sequences will be examined.

2.1.1 Benefits of Multiple Cameras

In this thesis, only one camera will be used. Still, it is worth thinking about what kind of disadvantages this implies - especially for pose estimation in 3D space.
If multiple cameras are used for Human Pose Estimation, they are usually utilized on one of two ways:

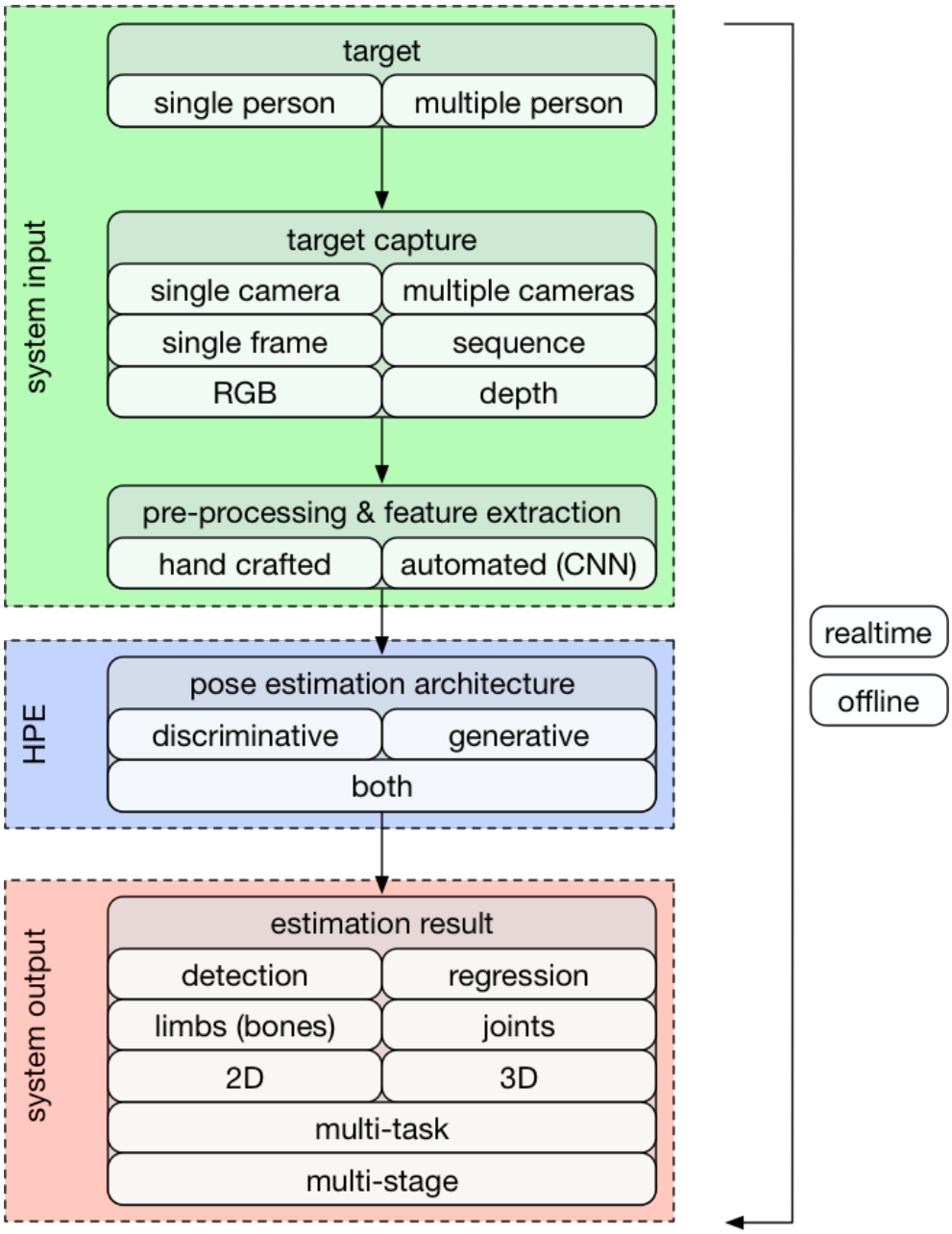


Figure 2.1: Key characteristics used to structure the state of the art for Human Pose Estimation

*The approaches are structured based on the differences along the signal processing chain. The signal processing chain itself is divided into three main sections: input processing (**green**, section 2.1), Human Pose Estimation architecture (**blue**, section 2.2) and system output (**red**, section 2.3). An additional criterion is weather or not one pass of the signal chain can be processed in real time ($t_{pass} \leq 100ms$).*

- Multiple 2D observations are used to create a system of equations that can then be solved by an optimizer to find the best matching 3D pose. [KADKHODAMOHAMMADI et al., 2017] report accuracy improvements by combining multiple RGB-D views in such a way compared to a single view. Similarly, [ELHAYEK et al., 2015] combine multiple 2D views to create 3D estimates, even for multiple persons.

- Alternatively, the Human Pose Estimation can be applied directly on the depth point cloud. In this case, multiple cameras are used to improve the quality of the point cloud. Such an approach is presented in [ZHANG et al., 2012].

Most approaches, however, did not utilize more than one camera and are still able to reach or surpass the performance of the mentioned multi-camera systems. Therefor, it is assumed that multiple cameras are not mandatory for a well performing 3D pose estimation system.

2.1.2 Using Depth Data as Input

The vast majority of Human Pose Estimation approaches use 2D RGB images as input data. Since a depth sensor will be used in this thesis, approaches using depth data are especially interesting.

[BAAK et al., 2011], for example, first filter the depth data to get a point cloud that only contains the human and no background. In a second step, a shape model is fitted into this point cloud. In a similar approach, [ZHANG et al., 2012] also fit a shape model into a point cloud, but they use multiple depth cameras to improve the accuracy of the point cloud first. These approaches require a sophisticated model of the human body in both shape and articulation.

A different kind of concept is to compute features on the depth data. For example, [SHOTTON et al., 2013] compute simple features for each pixel like shown in figure 2.2. These features describe the differences in depth at specific offsets from the examined pixel and can be computed relatively fast. Random decision forests are then used to classify each pixel based on those features. In a similar approach, [ARENKNECHT,

2016] computed IRON-features on the depth image and then used AdaBoost for classification. These IRON-features describe the 3D shape in a local area around the pixel. Possible benefits of IRON-features for Human Pose Estimation will be examined in section 4.2. Yet another way to process depth information is by treating it as an image

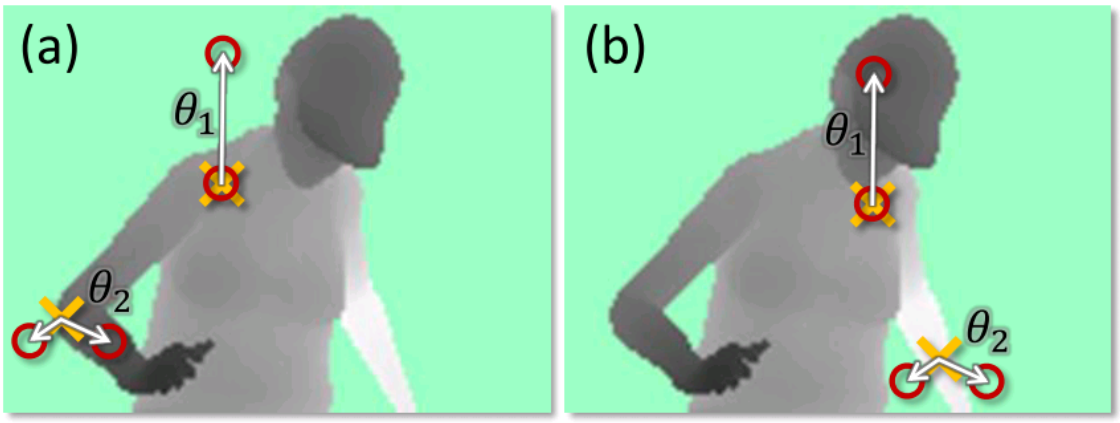


Figure 2.2: Simple but effective depth features
"The yellow crosses indicates the pixel x being classified. The red circles indicate the offset pixels. In (a), the two example features give a large depth difference response. In (b), the same two features at new image locations give a much smaller response."
Source: [Shotton et al., 2013]

channel and feed it into a Convolutional Neural Network (CNN). With only a single depth image as input, [Huang and Altamar] use a CNN to derive joint positions and utilize an original loss function that incorporates constraints from a kinetic model. Multiple possible ways to utilize the depth information of a single sensor with a CNN will be compared in section 4.2. In a more extensive use of input data, [Kadkhodamohammadi et al., 2017] use the RGB-D images of multiple kinect cameras for Human Pose Estimation in operating rooms. Here, the depth data is just used as an additional channel on top of the RGB images and fed into a deep-CNN. After creating a 2D estimation for every view, the estimations are fused into 3D poses. Like in [Huang and Altamar], a kinetic model is used to introduce skeleton constrains in addition to the multi-view constraints.

2.1.3 Using Image Sequences as Input

It is also worth noting, that almost all approaches focus on Human Pose Estimation in single images rather than sequences. However, [Gkioxari et al., 2016] present a system using chained CNNs to detect human body poses in single images as well as image sequences. Some approaches also use filtering and smoothing after the pose estimation like in [Mehta et al., 2017] to improve the results for video sequences. But still, the estimation only uses single images. Conclusively, the detection in sequences rather than single images seems to be unprofitable.

2.1.4 Conclusion

The characteristics of the input data can be altered in many ways. Nevertheless, there seems to be no method that provides a systematical advantage.
Multiple cameras can be used to improve the results for Human Pose Estimation in 3D. However, the results are not significantly better than single image approaches and usually come at the cost of higher processing times. Depth information can either be exploited explicitly by fitting a volumetric shape model into a point cloud or by calculating features specifically on the depth data. Alternatively, a trend in recent approaches is to move away from hand crafted features and towards deep CNNs and image representations. Since this kind of approach has shown to be very effective recently (see benchmark is section 2.4), a CNN based system might be best suited for the task of this thesis. This might even eliminate the need for sophisticated preprocessing and manual feature computation, reducing both complexity and computation time.

2.2 Human Pose Estimation Architecture Designs

The most common way to categorize architectures for Human Pose Estimation is by differentiating between generative and discriminative approaches. Such a comparison will be provided in the next subsection.
On the other hand, a specific type of architecture becomes increasingly prominent in recent approaches. At least since [Toshev and Szegedy, 2013] introduced DeepPose,

deep CNNs gained a lot popularity and have now taken a clear lead in benchmarks (see section 2.4). Additionally evident in these recent benchmarks, the Stacked Hourglass architecture is especially influential. Therefore, it will be described in more depth afterwards. The principle of another CNN design, the Convolutional Pose Machine, is outlined in appendix A.2.

2.2.1 Generative and Discriminative Approaches

Generative methods use some form of a model of the human body as a priori knowledge. At the most basic level, it usually consists of a kinematic model describing the relation between the individual joints (similar to the kinematic chain in figure 1.2). As an example, this kinematic model can then be used to manipulate a shape model in such a way that it fits the observation the closest like described in [Zhang et al., 2012] and [Baak et al., 2011]. Kinematic models can also be used to introduce constraints, e.g. the maximum possible angle between limbs or bone lengths like in [Elhayek et al., 2015]. A subcategory of generative methods are part base approaches. There, a human is modeled by defining body parts and constrains to connect them. The most popular approach for part based Human Pose Estimation is the Pictorial Structure Model like used in [Eichner et al., 2012].
In any case, the performance of such systems is always dependent on the quality of the model.

Discriminative methods, on the other hand, don't exploit this kind of a priori knowledge. Instead, they usually try to learn the relation between input data and pose directly. Often times, those designs are able to perform faster. With the growing success of CNN approaches, discriminative methods become more and more dominant recently. One of them will be described in the next section in more detail.

Finally, hybrid approaches try to combine the benefits of both methods. For example, [Baak et al., 2011] extract features from a filtered depth cloud to find the closest match in a pre-defined pose database. Then, they use a kinematic and volumetric model to

match the found pose more closely to the actual point cloud. [Elhayek et al., 2015] compare a generative, a discriptive and a hybrid approach. They report that the descriminative method outperforms the generative. Further, the hybrid method does not significantly improve the performance but is able to better resolve more difficult situations.

2.2.2 Stacked Hourglass Architecture

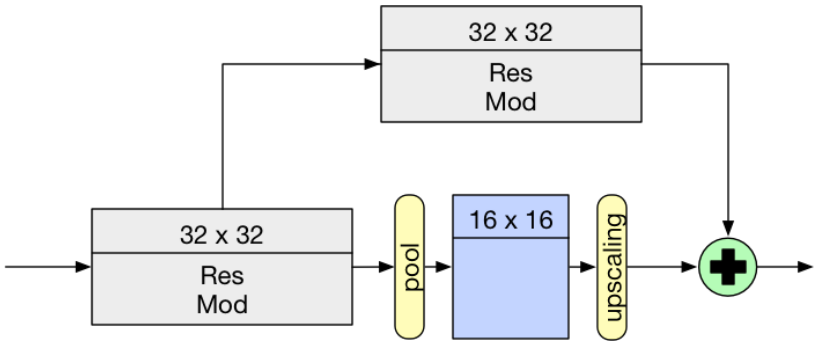


Figure 2.3: Basic building block of the Stacked Hourglass architecture
All rectangular blocks represent some sort of feature computation. The grey blocks resemble Residual Modules like displayed in figure 2.4, the blue block can either be a Residual Module as well or yet another Stacked Hourglass building block to create a nested architecture like shown in figure 2.5.
While the upper branch computes features on the original resolution (here: 32×32 *pixel), features in the lower branch are downscaled to half the original resolution (max-pooling) and successively upscaled (deconvolution with fixed weights). Finally, both branches are combined by adding the results. This resembles a multi-resolution residual module.*

The Stacked Hourglass (SHG) is a discriminative CNN architecture for Human Pose Estimation that is able to preocess images and produce 2D joint estimates. It was introduced in [Newell et al., 2016] and gained a lot of popularity since then. As discussed later in section 2.4, five out of the ten best performing approaches on the MPII Human Pose Dataset benchmark are based on SHG architectures.

The SHG combines some principles of the Pose Machine (see appendix A.2) into a more efficient architecture by the extensive use of residual learning (see section 3.2.2) on multiple levels. One foundation is the assumption that features at different scales are needed for a good estimation result. Instead of multiple identical pipelines that process the image at different scales, the SHG uses a single pipeline with skip layers. The basic building block of this pipeline is a multi-resolution residual module as displayed in figure 2.3. Features are computed on two separate resolutions and then combined by element wise addition like in a regular residual module. By nesting multiple of these blocks in each other (replacing the blue module in figure2.3 by yet another block), the hourglass architecture in figure 2.5 is created. In [Newell et al., 2016], 4 of these blocks are nested within each other so that the network reaches its lowest resolution at 4x4 pixel.

The SHG will be compared to alternative CNN architectures in section 6.1.

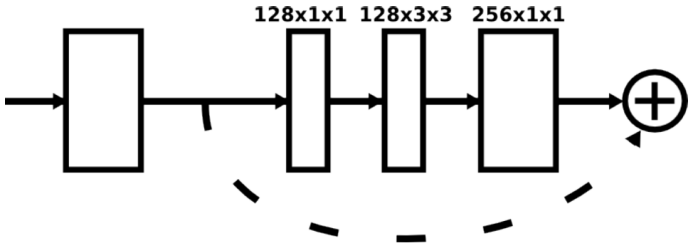


Figure 2.4: Residual Module that is used in the Stacked Hourglass architecture
The design of the basic Residual Module is based on [He et al., 2016]. It consists of three Convolutions which form a bottleneck and a skip connection (dashed line). If the number of filters in the last Convolution is different from the input, the skip connection contains an additional Convolution.
Source: [Newell et al., 2016]

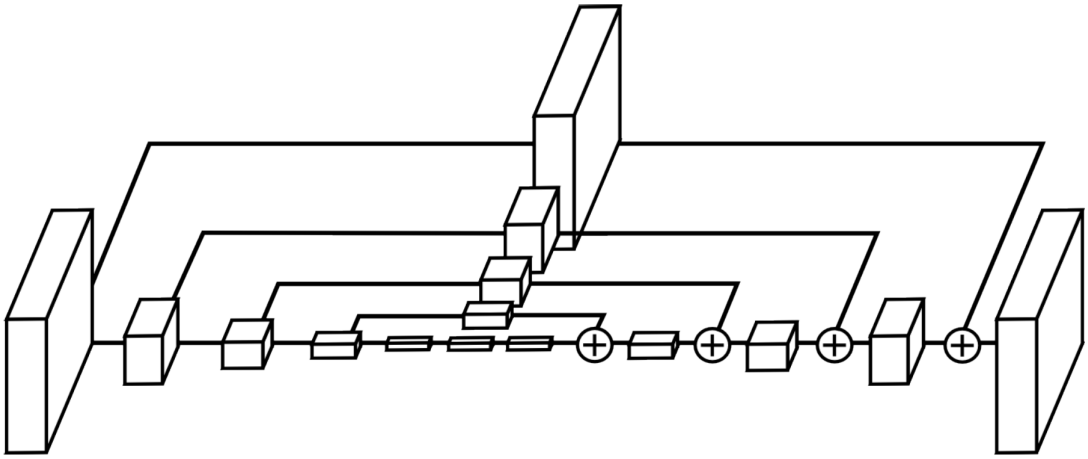


Figure 2.5: Basic Stacked Hourglass architecture
"An illustration of a single "hourglass" module. Each box in the figure corresponds to a residual module as seen in Figure 2.4. The number of features is consistent across the whole hourglass"
*Source: [*Newell *et al., 2016]*

2.2.3 Conclusion

State of the art architectures for Human Pose Estimation differ in the way they try to incorporate a priori knowledge. A lot of early designs utilized a human model, but the definition of such a model is not trivial and increases complexity. Hybrid approaches mostly tried to stabilize a fast discriminative pose estimation by including some sort of model. However, many recent CNN based designs like OpenPose [Cao et al., 2017] and VNect [Mehta et al., 2017] have shown that this is not necessary for a robust real time detection. For these reasons, a discriminative CNN approach will be considered for this thesis.

2.3 System Output

While several common principles can be observed regarding the system input and architecture across a lot of approaches, the concepts for system output are more diverse. This section will examine the differences in targets and results of the methods for Human Pose Estimation. Specifically, the difference between detection and regression tasks, approaches for estimation in 3D and the possibilities to utilize multiple stages or tasks will be discussed.

2.3.1 Detection or Regression

One possible way of categorization is to distinguish between detection and regression tasks. While regression is used to produce numeric estimates in a (more or less) continuous range, the detection task is a form of classification. In Human Pose Estimation, regression is mostly used to directly produce the coordinates for each joint. This can be done in 2D like shown in [Toshev and Szegedy, 2013], but is mostly found in 3D like described in [Li et al., 2014], [Sun et al., 2017] or [Martinez et al.]. Attempts to produce 3D pose estimates with regression are briefly mentioned in section 6.3
Detection, on the other hand, is mostly used for a per-pixel classification with one class per joint. As a result, a matrix is created that describes the probability for every pixel to be in a specific class (see figure 2.6). With one class per joint, the location of a joint can be found by looking for the maximum. Such a matrix is called heatmap or belief map and arguably the most popular choice in recent approaches. Especially the majority 2D approaches utilizes heatmaps, for example [Insafutdinov et al., 2016], [Chu et al., 2017], [Gkioxari et al., 2016], [Bulat and Tzimiropoulos, 2016], [Newell et al., 2016], [Cao et al., 2017], [Ramakrishna et al., 2014], [Rafi and Leibe, 2016], [Tompson et al., 2014] or [Wei et al., 2016]. However, they can also be used for 3D estimation, as successfully demonstrated by [Mehta et al., 2017]. Because joint detection with heatmaps is successfully utilized in a wide range of well performing state of the art approaches, it will be used in this thesis.
Both tasks can also be combined, as discussed later in subsection 2.3.3 and 2.3.4.

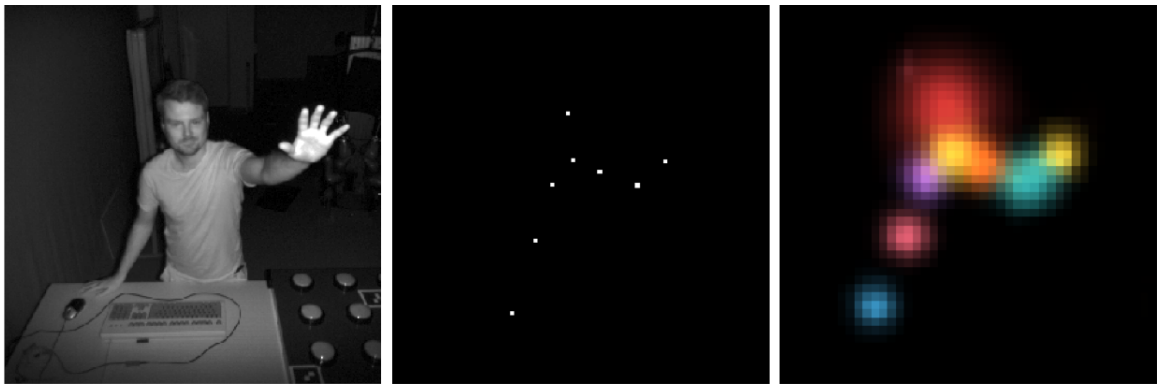


Figure 2.6: Heatmap representation of a pose
*The joint's positions (**center**) that represent the pose of a person (**right**) can be denoted as a heatmap (**left**). Each color intensity represents the probability of a pixel to belong to one of the eight joint classes.*

2.3.2 Estimation Results in 3D

One of the main differences regarding the result is weather the estimation is in 2D or 3D. Although very few approaches use depth information for the input data, several approaches try to reconstruct a 3D pose. For example, [Mehta et al., 2017] use a complex CNN architecture to create 3D poses from single 2D RGB images in real time. Similar to other CNN approaches, heatmaps are used to indicate the 2D-location of joints. Different to other approaches, however, three location maps additionally describe the relative distance from a root joint in all three dimensions (X,Y,Z) for every joint. Also, auxiliary tasks like bone length estimation are incorporated into the CNN to guide the detection.
Following a different strategy, [Kadkhodamohammadi et al., 2017] use a multi-camera setup to first create one 2D skeleton estimate with certainty scores per view. In a second fusion step, these multi-view estimates are merged into a 3D skeleton estimate by energy minimization.
The approach presented in [Tekin et al., 2016] tries to stabilize the 3D pose estimation by fusing it with the results of a heatmap based 2D estimation. Since

the optimal fusion strategy was not obvious, the used CNN architecture learned the fusion parameters itself.
Focusing more on speed than accuracy, [Baak et al., 2011] use a big database of 3D poses and just search for the pose that matches the input the best. Thus the output is a discrete pose out of a set of predefined poses. [Huang and Altamar] adapt this approach for the use of depth images as input, but use absolute coordinates to describe resulting joints rather than relative ones. Their comparison shows, that this can improve the regression accuracy slightly. However, [Sun et al., 2017] suggest that using joint coordinates for the pose estimate has some disadvantage and suggests using bones (vector pointing from one joint to its parents) instead.
Independently of the pose estimation itself, [Martinez et al.] describe a simple CNN architecture that can be used to efficiently transform normalized 2D joint estimations into 3D skelton estimations without additional information.

In conclusion, pose estimation in 3D is still comparably rare and attempted with diverse methods. However, most of them show that a 3D estimation can be done as a subsequent step to a 2D estimation. This would allow to focus on 2D estimation first and benefit from the high number of good state of the art approaches in this area.

2.3.3 Multi-Stage Architectures

A common theme in more recent approaches is the use of multi-stage architectures, where each stage refines the result of the previous stage and therefore increasing precision. One of the early popular approaches exploiting this is the Pose Machine introduced in [Ramakrishna et al., 2014], where multiple similarly constructed stages create pose estimates based on the output of the previous stage and the original input image (see appendix A.2). This allows to exploit intra-joint relations (spatial context) in higher stages by computing features on the output of previous stages like displayed in figure 2.7. For example, if the result of the first stage suggests the head to be in a specific location, suggestions for the right elbow that are very far from this location become very unlikely.

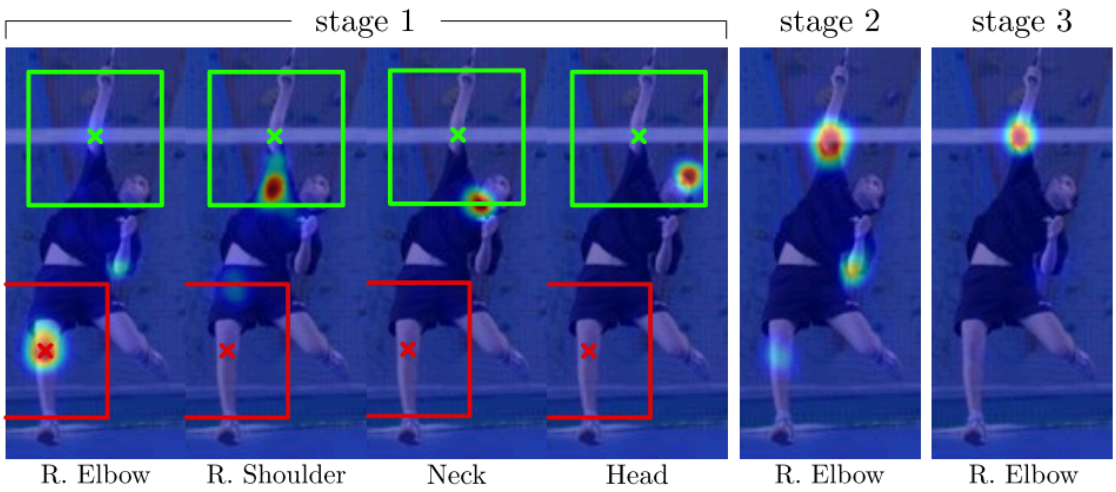


Figure 2.7: Usage of spatial context in multi-stage architectures
"Spatial context from belief maps of easier-to-detect parts can provide strong cues for localizing difficult-to-detect parts. The spatial contexts from shoulder, neck and head can help eliminate wrong (red) and strengthen correct (green) estimations on the belief map of right elbow in the subsequent stages."
Source: [Wei et al., 2016]

Similarly, [Toshev and Szegedy, 2013], [Newell et al., 2016] and [Chu et al., 2017] also use a multi-stage CNN to gradually refine the pose estimation results. [Tompson et al., 2014] also exploits spatial context, but by learning a explicit spatial model.

Another advantage of multi-stage architectures is how easy the trade-off between speed and precision can be influenced. By reducing the number of stages and sacrificing a certain degree of precision, they can be tweaked for faster inference times. However, since speed is one of the main concerns in this thesis, fewer stages will be preferred.

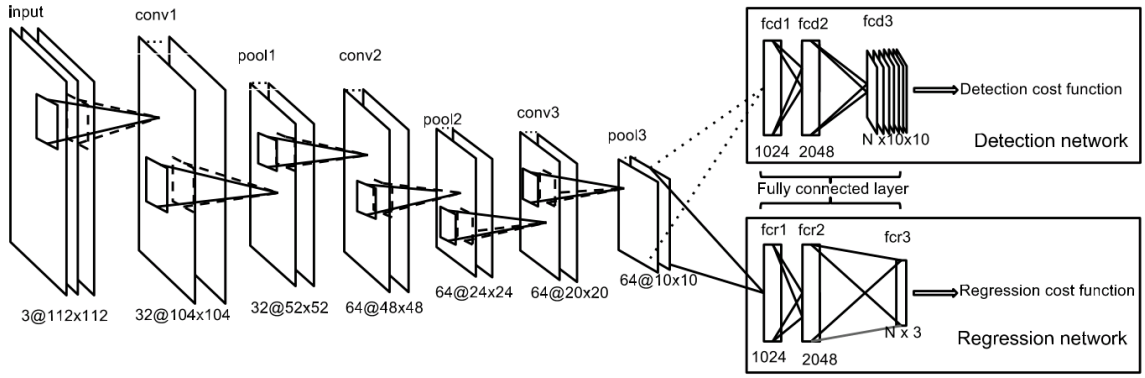


Figure 2.8: Network architecture that is trained for multiple tasks
"For network training using multi-task learning, the pool3 layer is connected to both the fcd1 and fcr1 layers. For pre-training with detection tasks, pool3 is only connected to fcd1 layer. After pre-training, this connection is removed and pool3 is connected to fcr1. N is the number of joints."
Source: [Li et al., 2014]

2.3.4 Multi-Task Architectures

Rather than stacking multiple similar stages on top of each other, there are also approaches that combine different tasks. For example, [Li et al., 2014] use a multi-task approach to generate 3D pose estimates from 2D RGB images like shown in figure 2.8. First, they train a CNN to perform the easier detection task where just the 2D location of joints is detected. Then, the last layers of the network are replaced by a regression head. The full Network is then trained to reconstruct the full 3D pose from the 2D images by using the pre-trained layers from the detection task. The resulting 3D joint positions are relative to their parent joint which decreases the possible error range.

[Bulat and Tzimiropoulos, 2016] use a similar approach where the two tasks (detection and regression) are combined at run time. The regression network takes the result of the detection network as input to produce the estimates. In that respect, the architecture is also a multi-stage-architecture. Other concepts also combine detections for different resolutions (like [Rafi and Leibe, 2016]) or 2D and 3D detection tasks

(like [Tekin et al., 2016]).
This also demonstrates the possibility to first solve the easier 2D estimation task and later extend the system for 3D estimation.

2.3.5 Conclusion

In conclusion, tasks can be formulated either as detection or regression. Both can be performed in 2D or 3D. While detection is easier in general, pose estimation in 3D usually is a regression problem. By utilizing multi-task architectures, the problem of 3D pose estimation can be split in more manageable parts. A system could build upon the wide range of well performing 2D estimation approaches and extend it to 3D estimation with an additional task.
The result of these tasks can be defined as bones or joints, either relative to a root or absolute. Further investigation is needed to determine which definition performs the best (see section 6.3). It is also possible to combine all of the above in several different ways or improve accuracy with multiple stages.

2.4 Summary

In this chapter, several ways of structuring the state of the art were presented. Most notably, the majority of approaches try to solve the problem of heatmap based 2D joint detection in single 2D images. While there are also several approaches for 3D detection, they are more diverse in their methods and results. However, most of these 3D approaches extend a preceding 2D estimation.
Even though early approaches often tried to incorporate different types of a priori models or hand crafted features, a clear trend towards CNN architectures could be observed recently. This also becomes apparent in rankings and benchmarks, for example the MPII Human Pose Dataset (MPII) Benchmark (see 2.9). By closer inspection, the SHG architecture design is especially successful, being used in five of the ten best performing approaches.

Taking all of this into consideration, the task of Human Pose Estimation using depth data will be tackled in multiple steps. First, a state of the art CNN architecture will be utilized for discriminative 2D joint detection. For this, a suitable depth data representation (section 4.2) and CNN architecture (section 6.1) must be found. After different improvements (section 6.2), possibilities for an extension to 3D estimation will be considered (section 6.3).

Method	Head	Shoulder	Elbow	Wrist	Hip	$S_{0.5}$
[Insafutdinov et al., 2016]	96.8	95.2	89.3	84.4	88.4	88.5
[Wei et al., 2016]	97.8	95.0	88.7	84.0	88.4	88.5
[Bulat and Tzimiropoulos, 2016]	97.9	95.1	89.9	85.3	89.4	89.7
[Newell et al., 2016]	**98.2**	**96.3**	**91.2**	**87.1**	**90.1**	**90.9**
Ning et al., 2017	98.1	96.3	92.2	87.8	90.6	91.2
Luvizon et al., 2017	**98.1**	**96.6**	**92.0**	**87.5**	**90.6**	**91.2**
[Chu et al., 2017]	**98.5**	**96.3**	**91.9**	**88.1**	**90.6**	**91.5**
Chou et al., 2017	**98.2**	**96.8**	**92.2**	**88.0**	**91.3**	**91.8**
Chen et al., 2017	98.1	96.5	92.5	88.5	90.2	91.9
Yang et al., 2017	**98.5**	**96.7**	**92.5**	**88.7**	**91.1**	**92.0**

Figure 2.9: Comparison of the ten best performing approaches on the MPII
*The scores represent the accuracy on the MPII using the $S_{0.5}$ metric (see section 3.3). All entries use deep CNNs and more specifically, the bold entries use the **SHG** architecture.*
Excerpt taken from the MPII Benchmark [MPI]

Chapter 3

Theoretical Principles

This chapter briefly introduces the theoretical background of the most important components in three sections. First, the measurement principle and possible error sources of the used depth sensor are described. Next, the basic concept of Convolutional Neural Networks as well as recent advancements in Machine Learning are discussed briefly. Finally, an evaluation metric will be introduced that is used throughout this thesis .

3.1 Depth Sensor Kinect V2

In this thesis depth information will be used as input data. A depth sensor can be used to capture such depth information by measuring its distance to objects in the scene. This can then be used to create a depth image (displaying the measured depth for every pixel). Different depth sensor types vary in the measurement method that is used to acquire the depth information. Here, the depth sensor of a Microsoft Kinect V2 shown in figure 3.1 will be used which utilizes the Time of Flight measurement method. It's measurement method and limitations will be described in the following section.

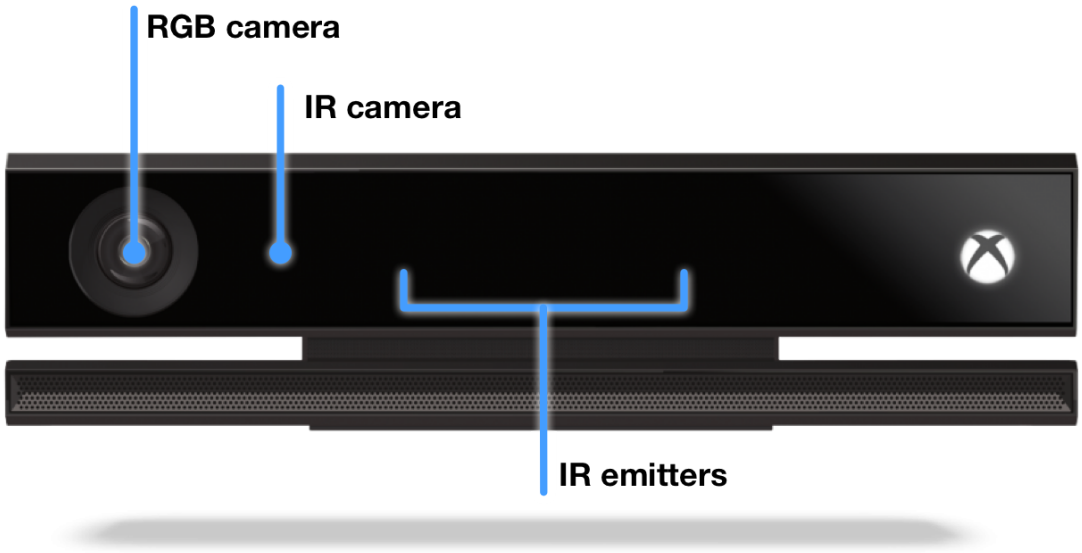


Figure 3.1: Features of the Kinect V2 sensor

The Kinect V2 can measure depth by utilizing the IR emitters and IR camera. According to the requirements for this thesis, the RGB camera will not be used.

3.1.1 Time of Flight Measurement

The goal of Time of Flight measurement is determining the time it took a wave to travel a distance through a medium. For this, the scene is first illuminated by emitting near infrared intensity modulated periodic light (Continuous Wave Intensity Modulation). After being reflected by objects in the scene, the light eventually returns and is received by the sensor of the infrared (IR) camera. Due to the traveled distance, the received light will be phase shifted by the time it reaches the sensor. This shift is detected by each sensor pixel in a so called "mixing proces" and used to calculate the corresponding object distance. In [Sarbolandi et al., 2015], this measurement process is described in more detail and compared to structured light measurement.

The Kinect V2 performs such a measurement at 30 Hz to create a 512×424 pixel (px) depth image. As shown in figure 3.2), the raw IR image used for depth calculation is available as well.

Figure 3.2: Example data provided by the Kinect V2
*The raw infrared image (**right**) is available along with the depth image (**left**) created with Time of Flight measurement.*

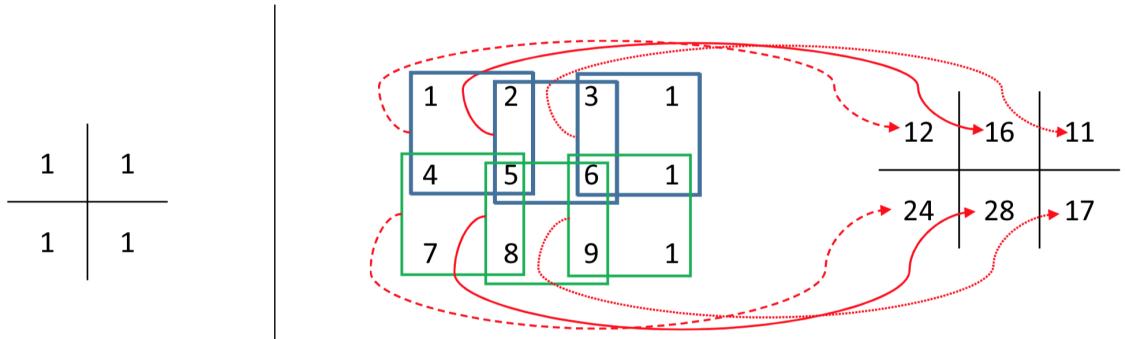
3.1.2 Error Sources and Limitations

According to the official specifications [kin, 2017], the depth sensor works best at ranges from 0.5 to 4.5 meters. In [Sarbolandi et al., 2015], the limitations are examined more closely and compared to the previous Kinect generation. There, a decrease in accuracy at the edges of the image and with increasing distance to the sensor could additionally be observed. Since the measurement principle of the Kinect V2 depth sensor relies on IR light measurement, it is sensitive to other light sources with the same modulation frequency. Also, strong light sources like sunlight will cause saturation and decrease the signal-to-noise ratio. For consistency, those influences should be reduced as much as possible.
Furthermore, multiple Kinect V2 in close range can interfere with each other. It is reported, that the phases of two Kinect V2Ls IR sensors in close proximity will overlap periodically which results in distorted measurements [Sarbolandi et al., 2015]. However, this could not be reproduced in the used setup.
Another issue are reflections and multi-path effects which result in conflicting measurements regarding the distance. To minimize those effects, mirroring surfaces should be avoided.

3.2 Neural Networks and Machine Learning

The review of the state of the art in chapter 2 has show that Convolutional Neural Networks (CNNs) are the most popular and successful result. In this section, a brief introduction to CNNs will be given, followed by an overview of recent advances in this field.

3.2.1 Convolutional Neural Networks



(a) A 2 × 2 kernel

(b) The convolution input and output

Figure 3.3: Basic principle of the convolution operation
*The kernel (**a**) is multiplied with the input to create the output (**b**). In this example, only a single input channel is used. However, this operation is also possible with multiple input channels (for example three channels for red, green and blue).*
Source: [Wu, 2016]

Convolutional Neural Network (CNN)s are a specific form of Neural Networks that is able to process data in a grid-like topology. This makes them especially useful for tasks where the data can be represented as images.
One of it's main building block is the convolutional layer. Like shown in figure 3.3 such a layer performs a convolution for an input using a kernel. The result of such an operation is called feature map, heatmap or belief map. By applying multiple different kernels to the same input, multiple output channels can be created. In example shown in figure 3.3, the output resolution was reduced by the convolution. In order to keep the

output resolution the same, padding can be added around the input image. Besides the amount of padding, the kernel size as well as the distances between kernel applications (stride) are the main parameters to configure the behavior of a convolutional layer.
A more detailed explanation of CNNs can be found in [Ian Goodfellow, Yoshua Bengio, 2015] and [Wu, 2016].
The aforementioned kernels are stored as weights and need to be learned in a training process. Stochastic gradient descent can be used to adapt the weights during this training process. A prerequisite for this is a sufficiently large set of training examples. While the amount of time necessary for this training might be in the order of days, it is not relevant in this thesis. Because, however, the system is expected to run in real time, the duration for processing a single frame is important. This duration is called inference time.

3.2.2 Recent Advancements in Machine Learning

A number of recent advancements enabled deeper networks, faster training, less memory consumption and a reduction in computation time. In this subsection, two of these advancements which are relevant for this thesis will be introduced.

Batch Normalization

When training very deep Neural Networks, a commonly observed problem is the distribution change of each layer's inputs during training (internal covariance shift). In an effort to provide means for compensating this internal covariance shift, [Ioffe and Szegedy, 2015] introduced the method of Batch Normalization. It can easily be implemented by adding BatchNorm layers at the desired locations. These layers calculate the output $\mathbf{y}$ to an input $\mathbf{x}$ by applying the following formula:

$$\mathbf{y} = \gamma \frac{\mathbf{x} - \mu_B}{\sigma_B^2 + \epsilon} + \beta \tag{3.1}$$

where $\mathbf{x} \in \mathbf{X_B} = (\mathbf{x_1}, \ldots, \mathbf{x_n}, \ldots, \mathbf{x_k})$ is the batch of training samples used to calculate the mean $\mu_B = \frac{1}{k}\sum_{n=1}^{k} x_n$ and variance $\sigma_B^2 = \frac{1}{k}\sum_{n=1}^{k} x_n - \mu_B$ while $0 < \epsilon \ll 1$ avoids division by zero

During training, a normalization is estimated within each layer by normalizing over each batch's mean and variance. These estimations for each batch are accumulated during training. At run time, the normalization is done regarding these accumulated values of the whole training set.

This kind of normalization helps with vanishing or exploding gradients, enables the use of higher learning rates and up to 14 times fewer training steps [Ioffe and Szegedy, 2015]. Besides others, it was used for Human Pose Estimation with a deep CNN by [Newell et al., 2016], [Rafi and Leibe, 2016], [He et al., 2016], [Mehta et al., 2017] and [Bulat and Tzimiropoulos, 2016]. Furthermore, it is an essential component to build deep residual networks.

Deep Residual Learning

When designing deep architectures, adding more layers often times increases the training process and sometimes even decreases the overall performance. Introduced by [He et al., 2016], Deep Residual Learning is an architectural design pattern to counter these problems. At its core, the basic idea is to only learn what was not already learned by lower layers. By adding the output from lower layers to the own output like shown in figure 3.4, only the difference is learned. As a result, additional layers are used more effectively. By preventing additional layers from decreasing the performance, Deep Residual Learning enables the effective training of even deeper networks and helps with vanishing gradients. To achieve network depths of 50 layers and more, [He et al., 2016] utilized Batch Noramilzation layers. These principles were used in deep CNN for Human Pose Estimation by [Chu et al., 2017], [Bulat and Tzimiropoulos, 2016], [Kadkhodamohammadi et al., 2017], [Newell et al., 2016], [Sun et al., 2017], [Insafutdinov et al., 2016], [Mehta et al., 2017] and others.

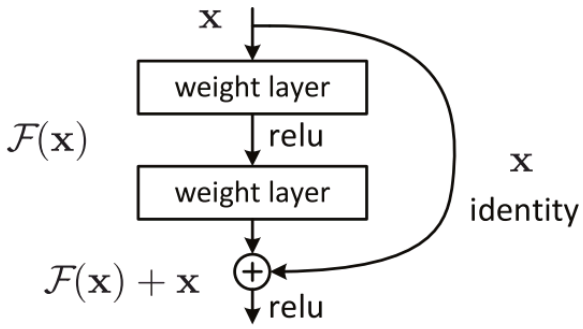


Figure 3.4: Residual learning building block

The basic building block for residual learning contains multiple weight layers (here: two) and a shortcut connection. The result of this block is calculated by adding the shortcut (and therefor the original input) to the result of the weight layers.

Source: [He et al., 2016]

3.3 Evaluation Metric

A metric for objective comparison is very important for both the development process and the evaluation of the final results. Such a metric depends on the specific goals and properties the evaluated system should achieve. In this thesis, the goal is to design a joint detection system that is precise, robust and fast. The amount of time needed to train such a system is not considered relevant.
Precision can be measured as the distance between the systems output and the ground truth (detection error). Robustness can be perceived as the amount of precision variance and the existence of a (soft) upper limit for the detection error. For speed comparison, the number of processed frames per second can be used. A proper metric should enable a fair comparison of those properties between different systems.
Several metrics for this purpose are used in the literature, three of which will be briefly compared here. All of them are based on measuring the percentage of correctly detected joints or limbs. They differ in the way the maximum allowed distance to the ground truth is determined in order to be considered correct. Some of the make an effort to normalize this distance so that it is independent to the objects distance to the camera.

- **Percentage of Correct Parts (PCP)** [Eichner et al., 2012]: a limb is considered detected, if the distance between its detected endpoints and the ground-truth endpoints is within a fraction p of the limb length. Usually, p is chosen to be 0.5. It comes with the disadvantage of penalizing shorter limbs.

- **Percentage of Detected Joints (PDJ)** [Toshev and Szegedy, 2013]: a joint is considered detected, if the distance between detection and ground-truth is within a fraction p of the torso length. Usually, p is chosen to be 0.2. It requires a torso to be labeled and detected.

- **Probability of Correct Keypoint ($\mathbf{PCK_h}$)** [Andriluka et al., 2014]: it is similar to Percentage of Detected Joints, but a joint is considered detected if the distance between detection and ground-truth is within a fraction p of the head length. Usually, p is chosen to be 0.5.

The PCK_h is best suited here because it introduces normalization regarding the camera distance without requiring a torso joint. Additionally, it is popular in recent papers and therefore allows for an easier comparison of the results. Conclusively, the PCK_h will be used in this thesis. Specifically, a graphical evaluation will be provided by plotting the detection rates for a varying $p \in [0, p_{max}]$ like shown in figure 3.5 (performance curve).
For a detailed evaluation, several aspects of such a graph are of interest. First of all, the **median detection error** (marked with a dot or short line) is an indicator for the precision. The slope of a curve (top graphic) and spread of points (bottom graphic) correlates to robustness. In figure 3.5, for example, the neck detection is more robust than the head detection, even though it has a higher median error. This is apparent in the steeper curve (top graphic) and smaller error variance (bottom graphic). Finally, the **maximum detection rate** marks the highest possible detection rate for the maximum allowed error p_{max}. Depending on the type of graphic p_{max} might be equal to or grater than 0.5.

In this thesis, the head length used for scale is the distance between the head and neck joints and can be expected to be roughly about 20 cm at $p_{max} = 1.0$. In all chapters regarding design and development, $p \in [0, 1.0]$ will be used. This allows for a more detailed analysis and the exposure of potential improvement possibilities. In chapter 8, $p \in [0, 0.5]$ will be used for evaluation because it is common in the state of the art and therefore allows for a better comparison of the results.
In the following chapters, the $PCK_h(p)$ will be denoted as S_p.

Unless noted otherwise, all experiments in this thesis were done on a single GeForce GTX TITAN X with 12 GB G5X RAM using Caffe 1.0 [caf, 2017] and cuda 7.5 [cud, 2015].

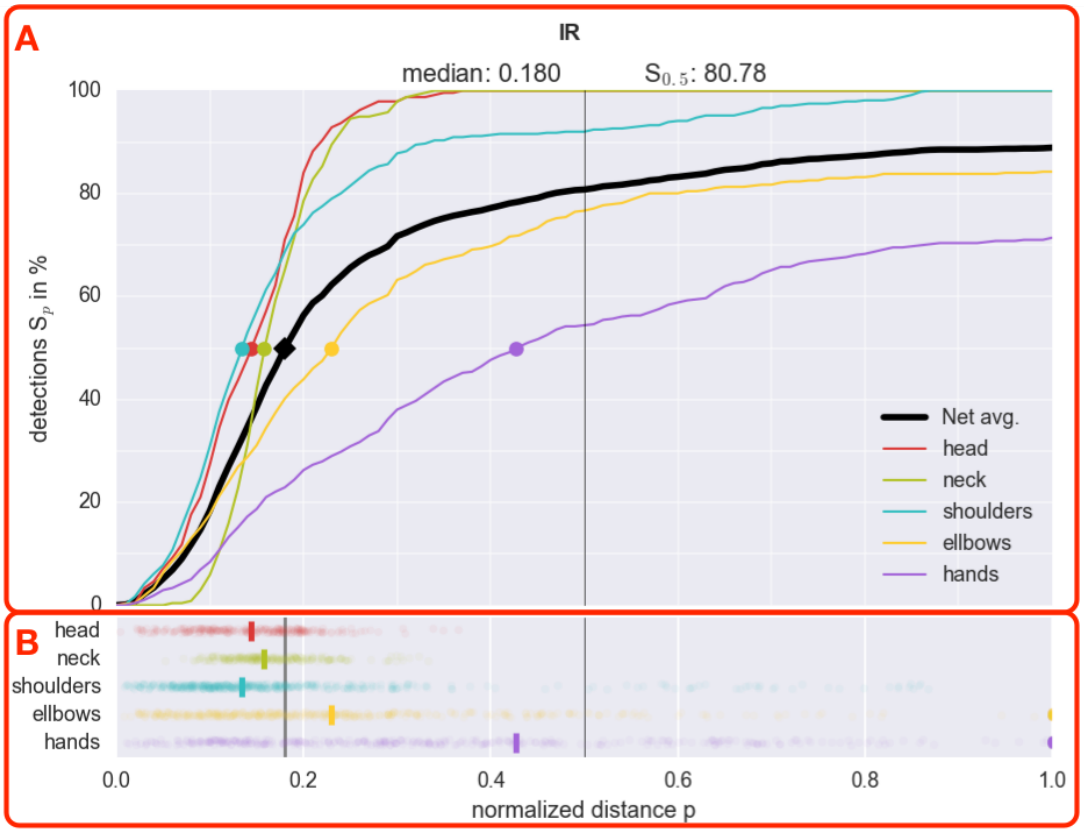


Figure 3.5: Example performance curve used for Evaluation

***B**: For each joint, the error of every single detection (normalized to head size) is marked as a dot on the distance axis. With this visualization, the variance and median (denoted by a small line) of the detection error becomes easily visible.*

***A**: For each joint, the percentage of correct detections $S_p \equiv PCK_h(p)$ is plotted for varying $p \in [0, p_{max}]$. A joint is correctly detected if it's error is below p. The median error distance (resulting in 50% detection rate) is marked with a dot.*

In both graphics, the average performance across all joints is marked with a bold black line. A small vertical line indicates the commonly used $PCK_h(0.5)$ metric where $p = 0.5$. To enable for easy comparison, the median detection error as well as the detection rate at $p = 0.5$ across all joints are additionally shown at the top of the graphic.

Chapter 4

Analysis of Existing Components

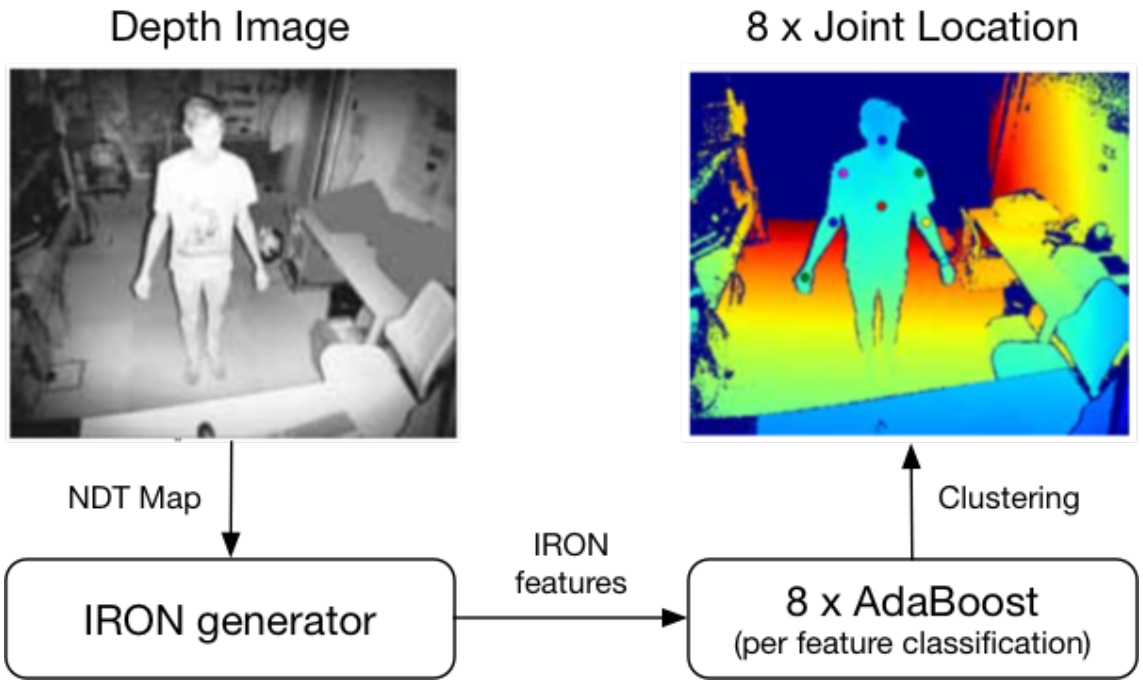


Figure 4.1: Main components of the Human Pose Estimation system presented by [Arenknecht, 2016]

*An **IRON generator** is used to calculate IRON features based on a single depth image. All of those features are then fed into eight **AdaBoost classifiers**,one for each joint. The output of each of those classifiers is then clustered to find the 2D center for each joint.*

This thesis succeeds the work of [Arenknecht, 2016], where a Human Pose Estimation system was developed as well. The basic principle of that system is shown in figure 4.1. As explained in section 2.4, a CNN architecture will be used in this thesis, but it might still be possible to build upon some of the already existing components. More precisely, the **IRON generator**, the **AdaBoost classifier** and a set of annotated images used for training (**training data**) can be used.
In this chapter, the possibilities to utilize those components will be explored.

4.1 Utilizing the IRON Generator and AdaBoost Classifier

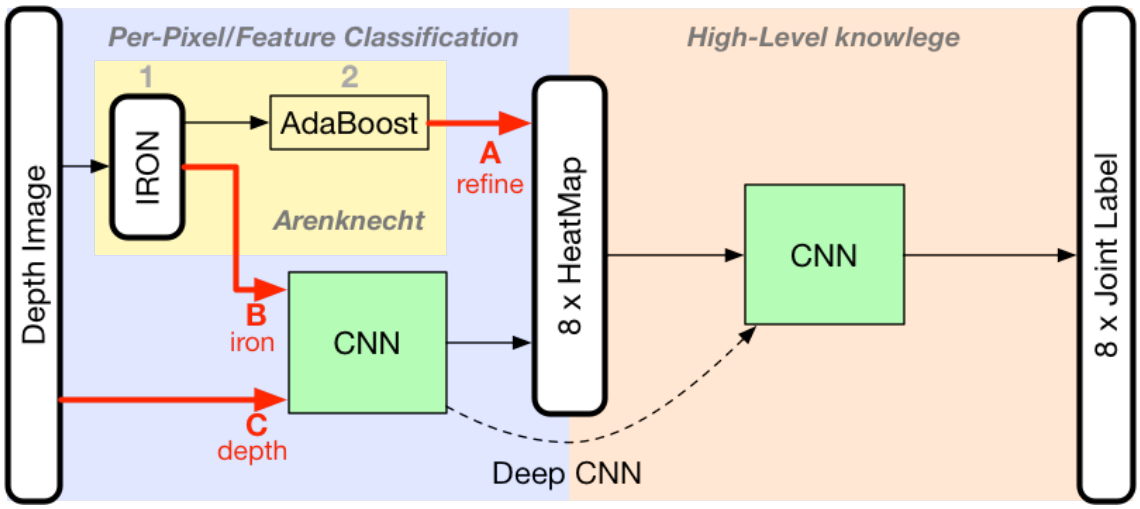


Figure 4.2: Possible architecture options incorporating existing modules
*The figure shows three possible options (**A (refine)**, **B (iron)** and **C (depth)**) using the preexisting modules (yellow box) for IRON descriptor generation (**1**) and joint detection (**2**) from [Arenknecht, 2016] to different degrees.*

The system from [Arenknecht, 2016] provides IRON-feature computation followed by a classification of these with one classifier per body joint. However, no high-level knowledge like the spacial relation between joints is considered, which in many cases was shown to be highly beneficial for Human Pose Estimation (see section A.2). Similar

to the multi-stage and multi-task approaches presented in section 2.3.3 and 2.3.4, high level knowledge may be introduced by a succeeding CNN that refines the initial results.

To combine the existing modules of [Arenknecht, 2016] with a CNN-based state of the art architecture, this work proposes three options as illustrated in figure 4.2:

Option A (refine)

As described in section 2.3.3, the state of the art implies that refining initial estimations can be done using multiple successive stages. Following this idea, the entire existing system could be used for an initial estimation. A subsequent CNN can then be trained to refine this estimation by using high level knowledge like spacial relations between joints. However, this option may not be practical as the existing system needs at least 30 ms per frame to compute, which is already equal to the target frame rate. Since the computation time was additionally found to be significantly higher in reality, option **A** and therefore the AdaBoost Joint classifier by [Arenknecht, 2016] will not be used.

Option B (iron)

As a second option it might be possible to replace the AdaBoost classifier by a CNN that produces the initial estimates directly. This would allow to build upon the feature generator and use IRON features as input for the CNN. If the IRON features contain valuable information this might simplify the problem and allow for a better pose estimation.

Option C (depth)

In case that the IRON features prove to be of no advantage, a deep CNN should also be able to find meaningful features on its own. This third option results in the simplest architecture but does not use any of the existing components.
To determine if IRON features can provide an advantage, they are compared to multiple alternative input representations in the next section.

4.2 Usability Evaluation of IRON Features

To examine the usability of IRON features for a CNN, a comparison with four alternatives will be presented in this section. This corresponds to a comparison of option **B (iron)** and option **C (depth)** in figure 4.2 by testing several specific implementations.
In order to find the best solution, all available options were evaluated using a simple CNN architecture. It is similar to the first stage of the Convolutional Pose Machine (see A.2) and consists of five convolutional layers and one pooling layer after the first convolution. The network was trained for 5,000 iterations with a batch size of 32 on all five input data representations, resulting in a total of five individual CNNs.

The tests in this section were originally performed on the data by [Arenknecht, 2016] described in section 4.3, but were repeated later on the training data that is used throughout this thesis (described in chapter 5) in order to verify the initial results. These experiments on the new dataset did indeed support the original conclusions by matching the initial results very closely. To allow for better comparison with other experiments in this thesis, they will be described here instead.
In order to simplify the problem and allow a faster and easier comparison, only a small portion of the available data was used. However, this small portion should be comparably homogeneous so that it is possible for the tested network to converge quickly. For this reason, only the data of the G4 movement category (dataset CG4w, see section 5.5) was used. More precisely, it was found to be suitable by fulfilling the following criteria:

- The range of positions within the detection area is comparably small, since the person is always in front of the desk and facing the camera.
- Additionally, the persons position is the closest to the camera. Therefor, the size of the person is as big as possible and roughly constant.
- The range of motions and poses is limited to moving objects on the table and therefore relatively small and uniform.

In total, the CG4w data set used for training contained roughly 5,000 images. For evaluation, the CEvalW dataset with about 500 frames was used, because it belongs to the same G4 movement category (see section 5.5).

Since a CNN will be used, the input must be presented as $I \in \mathbb{R}^{X \times Y \times C}$. This can be interpreted as an image with C channels of size $X \times Y$ px in width and height. $x \in [1, X]$ and $y \in [1, Y]$ describe a point within such an image. In this way, depth information can be represented as a value $z \in \mathbb{R}$ for each point (x, y) of the image, denoting the distance from the recording camera.
The five different input representations use this image-like input format by filling the C channels with different kinds of data, following either option **B (iron)** or **C (depth)** of figure 4.2. A visual comparison of the used representations is shown in figure 4.3. In more detail, they use the channels in the following ways:

- **IRON** (option **B**) uses 39 channes in total, one for the depth (z) and the remainder for the 38 numeric values of the IRON features.

- **IRONZ** (option **B**) uses one channel containing only the depth information (z) of the IRON features

- **Depth** (option **C**) uses one channel containing the depth image of the Kinect V2 (after normalization without clipping)

- **IR** (option **C**) uses one channel containing the IR image of the Kinect V2 (after normalization with clipping)

- **IrDepth** (option **C**) uses both the depth and IR image of the Kinect V2 after a custom normalization on two separate channels

A final comparison of the performance using these input representations is provided with figure 4.8 in section 4.2.3.

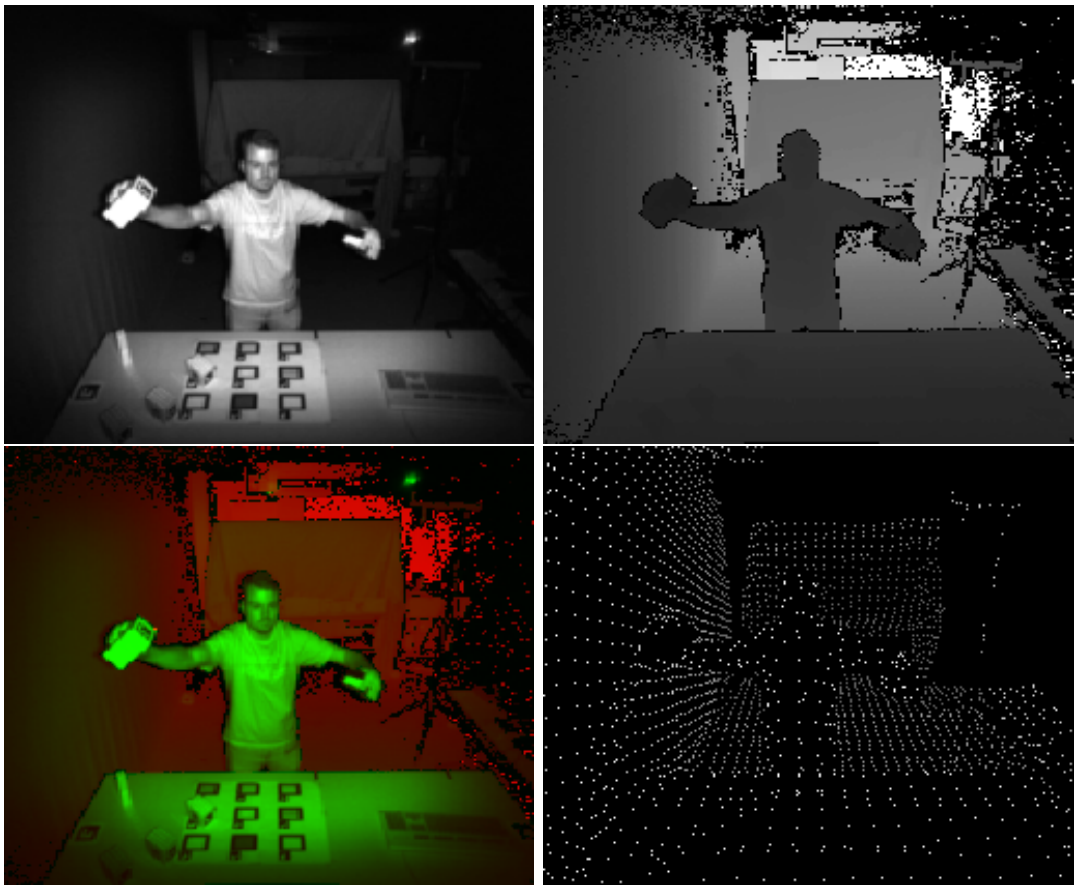


Figure 4.3: Visual comparison of input representations
top left: *only IR data, capped at intensity 70000 and normalized to be within* $[0, 1]$
top right: *only depth data, normalized to be within* $[0, 1]$
bottom left: *IR(green) and depth(red) data together, normalized to be within* $[0, 1]$
bottom right: *IRON (or IRONZ) values projected into the image*

4.2.1 IRON Features as Input

As shown in figure 4.1, [Arenknecht, 2016] used IRON features as proposed by [Schmiedel et al., 2015] for a joint classification. They contain information about the surface structure in a small region around a specific point. With the existing IRON generator, about 2,000 features are created for every depth image at interesting points in a grid-like fashion. Every feature has three coordinates for its (x, y, z) position and a vector of 38 numerical values that describe its characteristics. To make them

usable for a CNN, the feature's x and y coordinates were used to project them into images with 39 channels (see fig. 4.3 bottom right). That is one channel for its depth value z and one for every numerical value in the feature vector. Because features were only computed on surfaces with a high amount of linearity, their location alone might contain information about the unterlying object.
For this evaluation, the features were created offline using the tools that resulted from the work of [Arenknecht, 2016]. On average it took 2.07 seconds to create the features for one image.
As figure 4.4 (top) shows, the detection performance is acceptable for the head, neck and shoulders but gets worse further down the kinematic chain. By this results alone it is difficult to say whether the IRON-Features can provide advantages for this task. By closely inspecting the estimation results, it was assumed that only the z values of the IRON-Features are used while the other values provide no additional advantage.

To verify this, a CNN that uses only the z value of the IRON features (IRONZ) is trained and compared to the one using full IRON features. As can be seen in figure 4.4, both networks seem to perform identically. This suggests, that the features themself are not used at all and only their location is used. This was then additionally verified by inspecting the first layer of the IRON-network with Layer inspection Tool (LiT) (see section 6.2.2). The inspection results are shown in figure A.8 and confirm that only the depth information (first channel) of the IRON features was used.

Conclusively, IRON features do not seem to be suitable for this application though depth values seem to contain meaningful information. However, since the features are only generated for surfaces with specific properties, their location alone might contain some useful information. To investigate this assumption, different possibilities for using the image data directly will be examined in the next section.

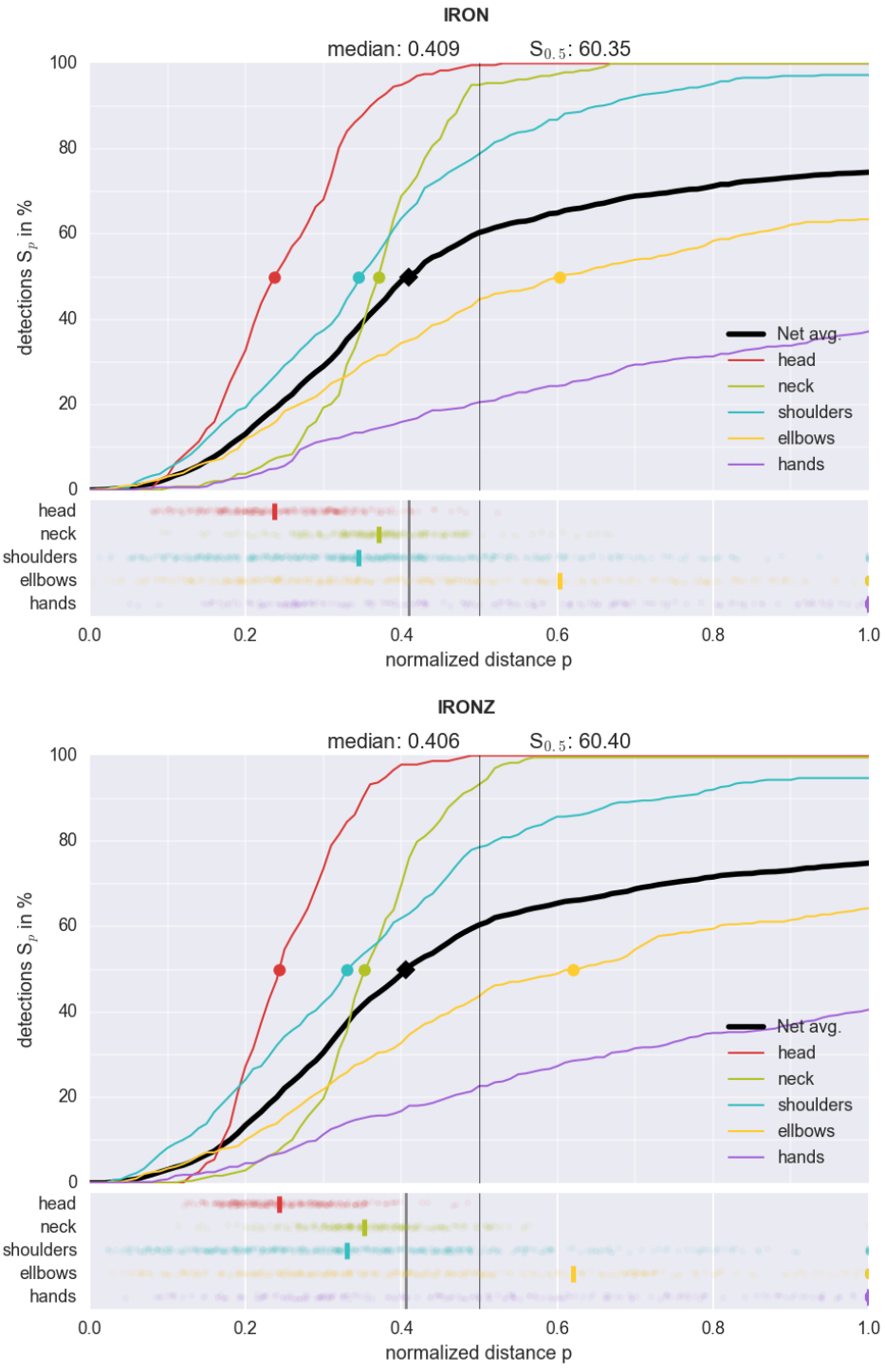


Figure 4.4: Performance for using IRON-features as input

*The detection rate of the CNN trained on IRON features (**top**) is not noticeable different form the one that only used their depth values (**bottom**). In both cases, head, neck and even shoulders could be detected well. However, the detection for elbows and especially hands is not sufficient.*

4.2.2 Depth and Infrared Images as Input

As a baseline for comparing the IRON features, a network was trained using only normalized depth images as input. The performance of this network is illustrated in figure 4.5. With even higher detection rates than the previous results, the detection for head and neck is very robust. Even though the detection of the hands and elbows is still very noisy, it is noticeably better than using IRON features. Nevertheless, the detection rate $S_{0.5}$ for hands and elbows is still only about 50%. To further improve these results the depth images were filtered before normalization by removing any information further away than 3.5 m. This was however found to be ineffective and did not improve the performance, possibly because of the large amount of noise still remaining.

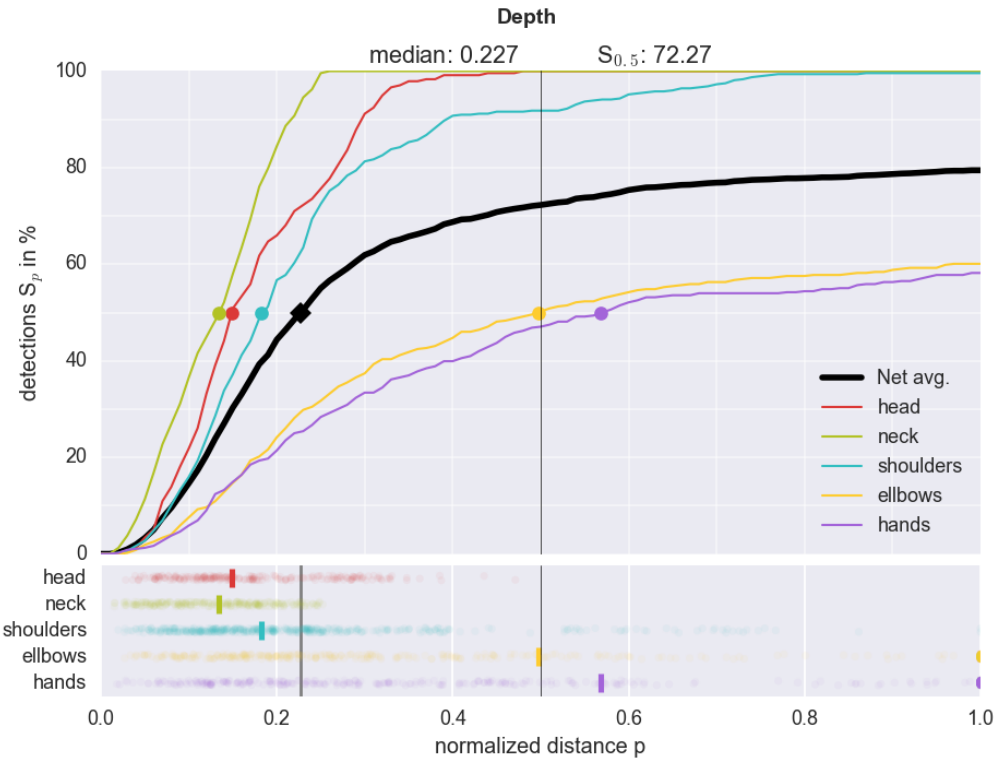


Figure 4.5: Performance for using depth images as input

The detection rate of the CNN trained on depth images is significantly better compared to the performance using IRON features (figure 4.4).

Apparently, one of the main error sources are situations where the hands are close to the background, for example lying on the table or in front of the body. The usage of IR images may lead to better decomposition in those situations because they still contain easily visible edges in those cases.

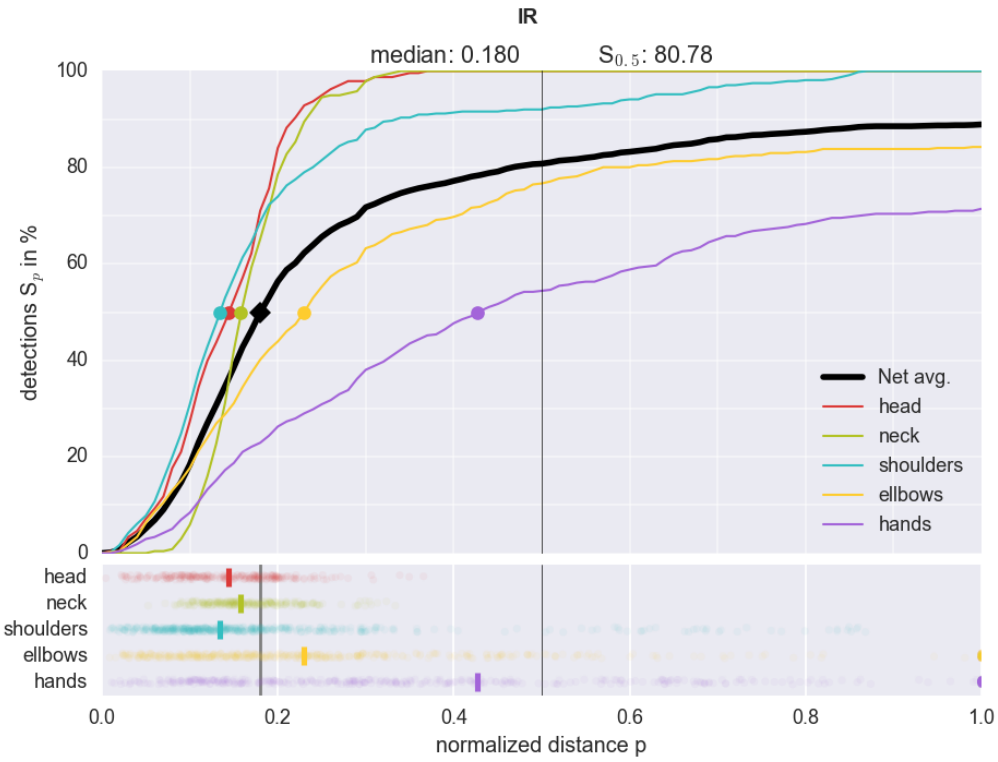


Figure 4.6: Performance for using IR images as input

The detection rate of the CNN trained on IR images is better for all joints compared to the performance using depth images (figure 4.5). Especially the mean detection error for elbows improved by more than 100%.

To investigate this assumption, a CNN was trained that uses only normalized IR images as input. An illustration of the resulting performance is shown in figure 4.6. Compared to the previous results, the detection for head and neck is even more robust and the hands and elbows are getting detected better as well. Even though the hand detection with $S_{0.5} > 50\%$ may seem reasonable, it is still the biggest source of errors.

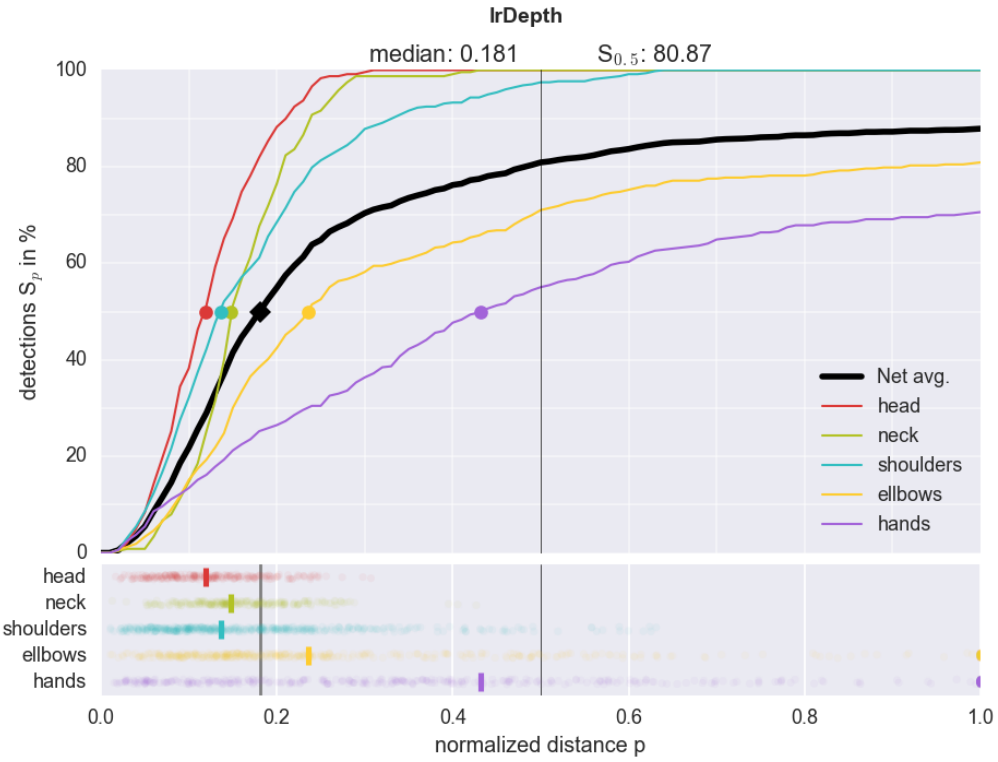


Figure 4.7: Performance for using both depth and IR images as input
The combination of depth and IR images does not seem to noticeably improve the detection rates compared to using only IR images (figure 4.6).

To examine if a combination of both IR and depth images can provide more expressive information than each of them alone, a network was trained that uses both as input. Even though minor improvements could be observed, the difference in performance was very small (see figure 4.7).

4.2.3 Conclusion

A performance comparison for all five input data representations evaluated in this section is shown in figure 4.8. As discussed in section 4.2.1, the performance for IRON and IRONZ features is virtually identical and the lowest of all tested representations. This suggests that only their depth information is used. Hence, this sort of represen-

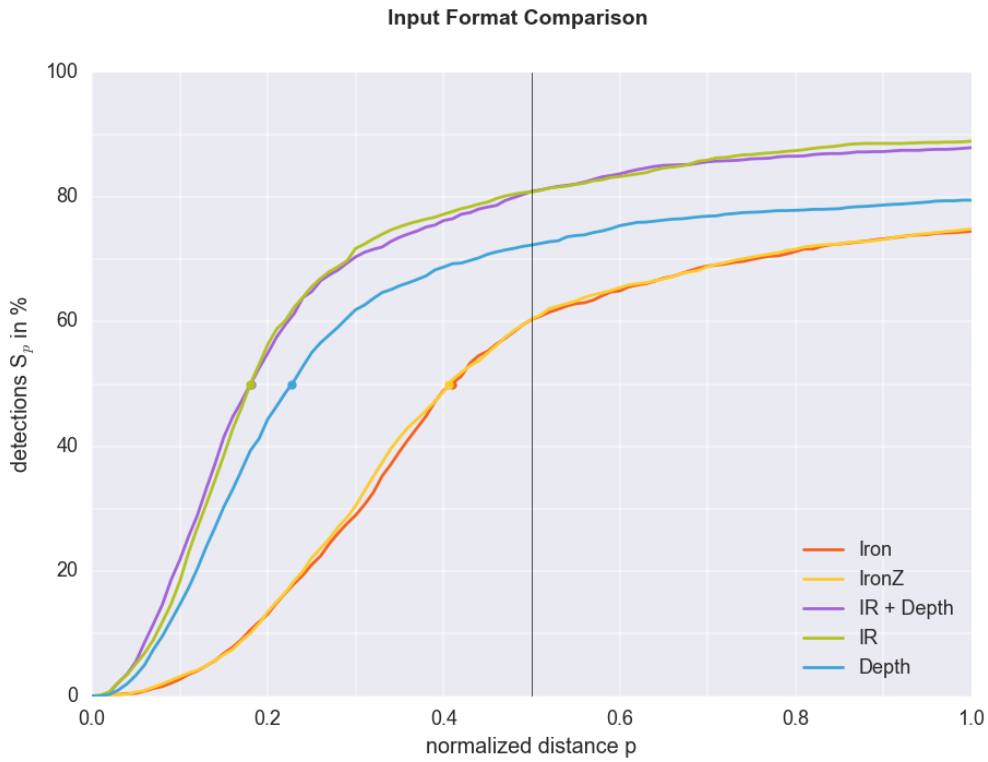


Figure 4.8: Comparison of input data representations for CNN pose estimation
The usage of depth information in the form of IR or depth images as input data seems more suitable than using IRON features.

tation is not beneficial. In addition to those disadvantages, the computation of these features is with up to two seconds quite time intensive. Therefore, IRON features will not be used in this thesis.

In contrast, the performance using the Kinect V2 camera data directly as input is significantly better. While the IR images performed slightly better than depth images, a combination of both could only improve the results marginally. However, the latter did not increase the computation time either and might prove to be beneficial in situations where the distance between person and camera is higher.

For these reasons stated above IRON features and therefor option **B (iron)** of figure 4.2 will not be used. Following option **C (depth)** instead, a combination of IR and depth images will be used as input.

4.3 Training Data

Besides the components for IRON feature detection and joint classification described previously, the work of [Arenknecht, 2016] also includes training data that was recorded for a very similar scenario to the one described in section 1.2. This data is still available and consists of two sequences, walking around the detection area and simulated work, each performed by two different persons. The data was recorded at 30 fps, but only a fraction of it was annotated, resulting in a total of four sequences and 1,400 labeled frames. Each labeled frame was annotated by hand and contains the 2D position of eight upper body joints.
The Kinect V2 camera used for recording was installed above head height and is angled down, which results in a noticeable perspective distortion of the human body. Because of that, the head appears very big if the person is close to the camera.
The recorded data contains IR, depth and RGB images. However, the depth images are preprocessed so that they only contain information that is closer than 3.5 meters.

4.3.1 Experiments

In order to determine if the existing training data can be used for the use case scenario described in section 1.2, a simple CNN architecture was implemented and trained on this data. Similar to the experiments in section 4.2, a simplified version of the Convolutional Pose Machine was used.
About 75% of the available data was utilized for training and 25% for evaluation. With a process similar to the one described in section A.3.4, heatmaps were generated from the available annotations as labels for training. Multiple different networks were trained to perform increasingly difficult tasks, ranging from detecting solely the head to full upper body detection.
The networks trained in this fashion were able to detect the head robustly and the shoulders most of the times. The detection for elbows and hands was more problematic though. Poses with the hands above shoulder height were never detected correctly. Instead, the network falsely assumed them to be in a neutral position next to the body. This also happened a lot for simulated work movements. Those

observations indicate, that the neutral position is overrepresented in the training data and therefore the network always defaults to it when unsure. Also, more uncommon poses where the hands are above elbow height seem to be underrepresented, since they were never detected correctly. A closer inspection of the pose distribution in the training data verified this assumption.

To further investigate whether the available data is applicable for the use case scenario, a simple motion sequence was recorded with the setup detailed in section 1.2.1 and preprocessed to fit the training data. The networks described above performed quite poorly on this sequence, struggling to detect the head. This may have been due to the different camera position which depicts the person (and especially the head) smaller and less distorted.

4.3.2 Conclusion

In conclusion, the available training data is not suitable for the task of Human Pose Estimation like outlined in section 1.2 because of three reasons:

- The perspective across all recordings is quite similar which limits the possibilities for generalization. In addition, this consistent perspective differs too much from the use case and introduces a high degree of distortion.
- The number of contained poses is too low and their distribution is disadvantageous. The neutral pose is highly over represented and a lot of more uncommon poses (for example hands above the head) are missing entirely.
- The number of 1,400 training samples is comparably low which likely results in overfitting, especially since many samples are quite similar. The state of the art approaches presented in section 2 usually were trained on at least 5,000 data samples, but often the number of training samples is much higher.

Therefore, better suitable training data has to be found. The process of creating such data will be described in the next chapter.

Chapter 5

Training Data

As explained before, a deep CNN architecture will be used for Human Pose Estimation in this thesis. Such a CNN usually contains a large number of free parameters (weights) that needs to be adjusted in order to perform as needed. This adjustment will be done in a training process called supervised learning, where the network is presented with pairs of input data and desired output. In order to enable the CNN to generalize and avoid overfitting, a sufficiently large number of such training samples is needed.
As of now, there is no broadly accepted heuristic available to determine the minimum amount of required training samples, since it depends on many different factors. Well performing approaches in the literature are usually trained on at least 5,000 data samples, but often times the number is much higher.

Besides quantity, sufficient training data also needs to fit the intended task qualitatively. In the scope of this thesis, suitable training data must therefore fulfill at least the following criteria:

- **Input**: Data from a depth camera (depth image, IR image or point cloud) must be available as input.
- **Labels**: The 3D location of the 8 upper body joints (see figure 1.2) must be available as ground truth.

- **Poses**: The training data must include generic poses of a single person as well as working poses where the lower body is often occluded. It must not contain multiple persons.
- **Count**: Especially for training deep CNNs, a large amount of training samples is needed. About 10,000 unique samples is expected to be sufficient.

A more detailed requirement profile will be given in section 5.1.

As examined in the previous section, a dataset by [Arenknecht, 2016] is available, but not suitable for this thesis. In addition, several different data sets for the task of Human Pose Estimation are publicly available. These are reviewed in appendix A.3.1. In summary, none of the described datasets is suitable for the task because the input data format and the available poses are not sufficient. Synthetic dataset creation is an alternative but would exceed the scope of this thesis. Therefore, the best solution is to manually record a new dataset specifically for the presented task. This also allows ensuring the data is best suitable as well as balanced for the intended use case.
The details regarding the scope, capture and post processing of this training data will be described in this chapter.

5.1 Requirements

To ensure good performance under the use case constraints described in section 1.2.2, several requirements for the training data were derived. These requirements can be split in two groups:

- **Match use case characteristics:**
  - Sensor constraints: The data is recorded with the Kinect V2 ToF camera.
  - Room and detection area constraints:
    - The room and detection area for data recording are set up like described in section 1.2.2.

    * The Kinect V2 camera is positioned like described in section 1.2.2 (behind the workbench, front facing the worker).

- **Achieve generalization for:**
  - Person: There must be various images from multiple people with different optical characteristics across the dataset.
  - Pose: The data must include as many different relevant poses (see section 1.2.2) as possible in roughly equal quantity.
  - Person position: The poses have to be performed all over the detection area.
  - Background and foreground: The data must include at least minor changes in back- and foreground configuration
  - Hand usage: The data must include different tasks in which the hands are used to manipulate objects

Since the detection is based on depth information, the network might learn features based on the shapes of things in 3D space. This means that not only the visual appearance (like seen in a 2D image) needs to be considered when thinking about what training data to generate, but also basic shapes in 3D space. For example, characteristics of the basic shape of a hand might be learned. Such characteristics would change drastically, if an object was held in the hand. To allow for robustness in this aspect, different kinds of tasks in which the hands are used to manipulate objects need to be introduced.
Also, when using the depth information as input, big depth differences between foreground and background (for example a hand 5 m in front of a wall) are ideal for a good detection. Similarly, very small differences (for example a hand flat against the chest) proved to be especially difficult. So additional attention is needed for poses where hands are touching other body parts or objects.
Generalization regarding the specific camera position would be desirable, but was not possible within the used setup. Data augmentation (see section 6.2.1) was later used to compensate for this.

Finally, different datasets for training and validation are needed. This is necessary to monitor the training process and objectively evaluate and compare the results.

5.2 Room Setup, Calibration and Label Creation

The data was acquired in the setup of the use case scenario described in section 1.2.1. Additionally, a second Kinect V2 camera was installed in the opposite direction of the first one. It was used to better calibrate the setup and capture the scene more holistically, but not for the actual training data.
Beyond that, the setup was extended by installing hardware for label generation. Specifically, the multi-camera markerless motion capturing system "Captury Live" [cap, 2017] was used. It provides a robust realtime 3D skeleton with 28 bones (29 joints) based on at least four cameras.

The whole setup was carefully calibrated. First, the instrinsics and extrinsics for Captury Live were adjusted to match the four used cameras and to align the skeleton with a reference point in the real world. Then, a laser based precision measurement system was used to determine the spatial relation between this Captury Live reference point and the lenses of the two Kinect V2 depth sensors. Nevertheless, this calibration cannot completely compensate for all measurement errors. Especially the ToF-based Kincet V2 introduces some errors that are not covered by the sensor model, like multipath and distance related errors due to imperfect sinusoidal IR illumination. For this reason, a tool for refinement based on manual inspection and online adjustment was developed and used to decrease the remaining error between both kinects to about one voxel. The tool was also used to align the Captury Live skeleton equally in the center of the point cloud produced by the two Kinects.

The resulting setup allowed to easily record high quality depth and IR images at 30 Hz and 512×424 px along with matching skeletons at about 50
glsHz. The accuracy and residual errors of the recorded data are discussed in section 5.4 in more detail.

5.3 Variations for Generalization

To ensure that the desired generalization described in section 5.1 is possible, three categories of variations were introduced. This section will describe the specific variations for each category.

5.3.1 Person Variations

To allow generalization regarding the person to be detected, five different people with these varying characteristics were recorded:

- The participants differ in **height**, ranging between 1.65 m and 1.90 m, as well as in their **physique**.
- The **hair color** is ranging from blond to black across different **hair styles** (no longer than shoulder-length).
- The **clothing** differs in color and appearance, but all participants wear T-shirts and long pants.
- Additionally, one participant is wearing **glasses**.

5.3.2 Background and Foreground Variations

For a flexible background configuration, a big black board with a curtain was placed in the background, roughly 4 m away from the front facing Kinect. It was moved randomly to occlude varying parts of the background. Considering depth features, this acts like moving the back wall closer and farther (or just parts of it) and might influence the features that can later be used to separate a person from the background Furthermore, a chair and a box were randomly positioned in the background or on the sides to introduce some noise around the detection area. Sometimes, a box was positioned randomly on the workbench. This occludes parts of the workbench and the detection area and introduces noise to the shape of the work bench.

5.3.3 Position, Pose and Hand Usage variations

To ensure equally good performance for all poses, a set of motion sequences was chosen to cover the whole range of relevant poses as holistic as possible. The sequences and poses are structured into two categories: **Basic Poses** and **Work Poses**. Additionally, a third category exclusively for **evaluation** was created, enabling independent and objective assessment. Within each category, several subcategories were introduced in an effort to cover all their important characteristics explicitly. This segmentation further allows to manually balance the training data later on, if it was found that a certain type of pose is overrepresented or underrepresented. Possible consequences of disadvantageous pose distribution were already discussed in section 4.3.

In short, the sequences were designed around three main goals:

- Contain the complete range of relevant poses
- Crate location independence
- Do not underrepresent unusual non-work poses

To achieve those goals, motion sequences were designed in a two step process. First, key poses are defined which are extremes of a certain type of body part positioning. For example, holding the hands straight above head as one extreme and letting the hands hanging low next to body as the other. Next, a specific pattern to move between them is determined. By performing this movement pattern, all poses between those extremes on the specific path will be covered.
Following this process, the key ingredients for good pose variations are finding meaningful key poses and movement paths between them. A detailed overview of the complete list of motion sequences can be found in the appendix A.3.2.

Basic Poses

The Basis Poses category covers all poses that are not common during work. This includes idle poses like standing and walking, unusual poses like pointing in the air

and particularly difficult poses like covering the head with both hands. The focus is especially on the unusual and difficult poses, since they would be underrepresented and therefore hard to detect otherwise. Because the person may not be working, the location can be anywhere in detection area and the body (hips) and face can be facing any direction.

Four subcategories are introduced to ensure all important aspects are covered: **straight arms (A)**, **bent arms (B)**, **body touching (C)** and **random movement (D)**.
The CNN that will be trained on this data may learn features based on the relation between joints. Because of that, the subcategories **(A)** and **(B)** are meant to cover the range of possible constellations between the 6 joints for left and right hands, ellbows and shoulders.
As explained before, the detection based on 3D features will be difficult if the hands touch something. Therefore, the subcategory **(C)** is included to provide training data specifically for those poses.
Finally, motion sequences following the previous categories might be a bit unnatural sometimes and may exclude some important poses. For this reason, the subcategory **(D)** was introduced in which the participants can freely move in any way they feel like.

Work Poses

This category aims to cover typical and common poses during work, even though the requirements do not specify the kind of work that will be done in great detail. Nevertheless, it will include pressing some buttons on the workbench, manipulating smaller objects and using a keyboard. During work, the person will stand in front of the workbench with their hips roughly facing the camera. In this category, the focus is especially on manipulation of and interaction with objects, as well as performing the tasks specified as work.

Three subcategories are introduced to ensure all important aspects are covered: **idle poses (E)**, **button interaction (F)** and **box/LEGO interaction (G)**.
The focus of subcategory **(E)** are neutral poses where the desk or keyboard is touched. This might be particularly difficult regarding depth features. Subcategory **(F)** is similar in this respect, but also one of the actual work tasks intended for this scenario and therefor more dynamic. Complementary, subcategory **(G)** is the other intended work task and focuses on moving objects with the hands. This influences the 3D appearance of the hands and might also cause occlusions.

Evaluation

The evaluation category is intended for objectively measuring the performance of classifiers trained on the data described above. For a fair and critical analysis, all aspects a classifiers should be able to generalize must differ from the training data. Therefor, it must feature unseen poses (both from the front and back), a different background configuration and a new person that is not present in the training data at all.
To allow for a more detailed and independent analysis, thee subcategories were created: **basic poses (V1)**, **work poses(V2)** and **body touching (V3)**. The first two subcategories are analog to the previously discussed categories and contain a mix of common and especially difficult positions. For example, bending the upper body in category **(V1)** (which is not included in the training data) or crossing both hands while manipulation objects in category **(V2)**. Additionally, the subcategory **(V3)** focuses on particularly difficult poses where the hands are touching the body.

In summary, about half of the validation data is intended to push the classifier to it's limits and to allow for a detailed analysis of performance.

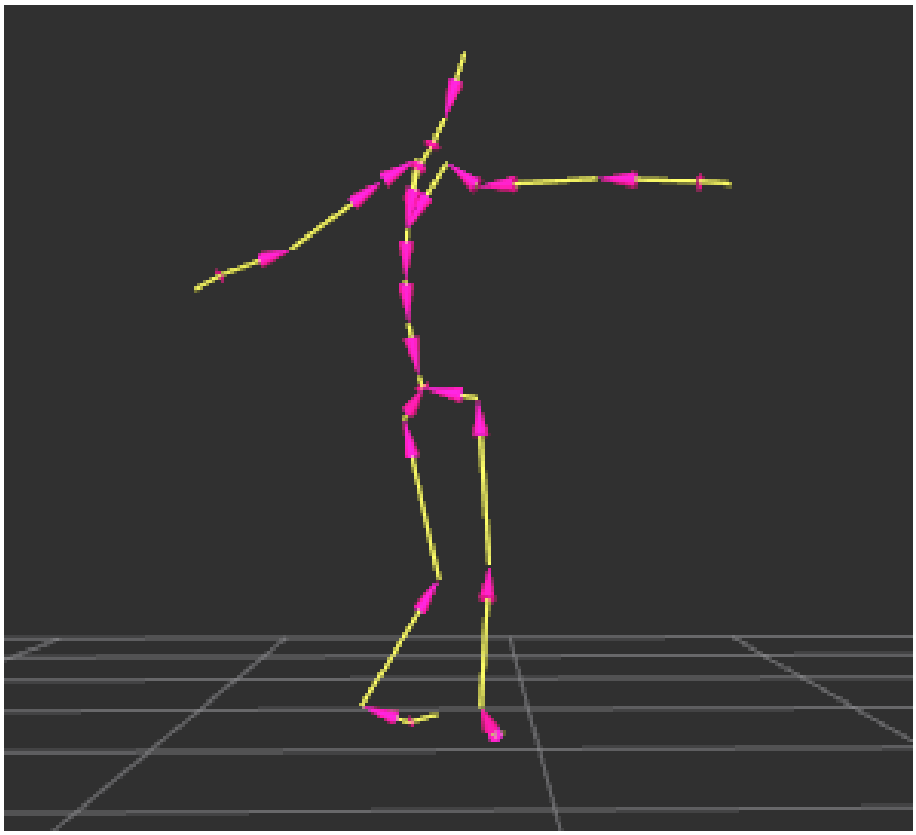


Figure 5.1: Captury skeleton used for labels
Skeletons with 29 joints like this one above are generated by CapturyLive and later used for labels.

5.4 Post-processing and Error Correction

After recording the raw data, errors need to be detected and the recordings need to be transformed into a more useful representation before it can be used for training. The raw data consists of time-stamped depth- and IR images and labels in the form of time-stamped Captury skeletons. A Captury skeleton consits of 29 joints in 3D space (x,y and z coordinates) which resemble a human skeleton like shown in figure 5.1.
The used methods of transformation and normalization are described in appendix A.3.3 in more detail. Likewise, the found errors are investigated more closely in that section.
Eventually, the algorithm shown in figure A.6 is used for a joint correctness classifi-

cation. It detects occlusions and erroneous poses based on the difference between the supposed joint locations and the actual depth information captured by the Kinect V2 camera. An example result of this algorithm can be seen in figure 5.2.

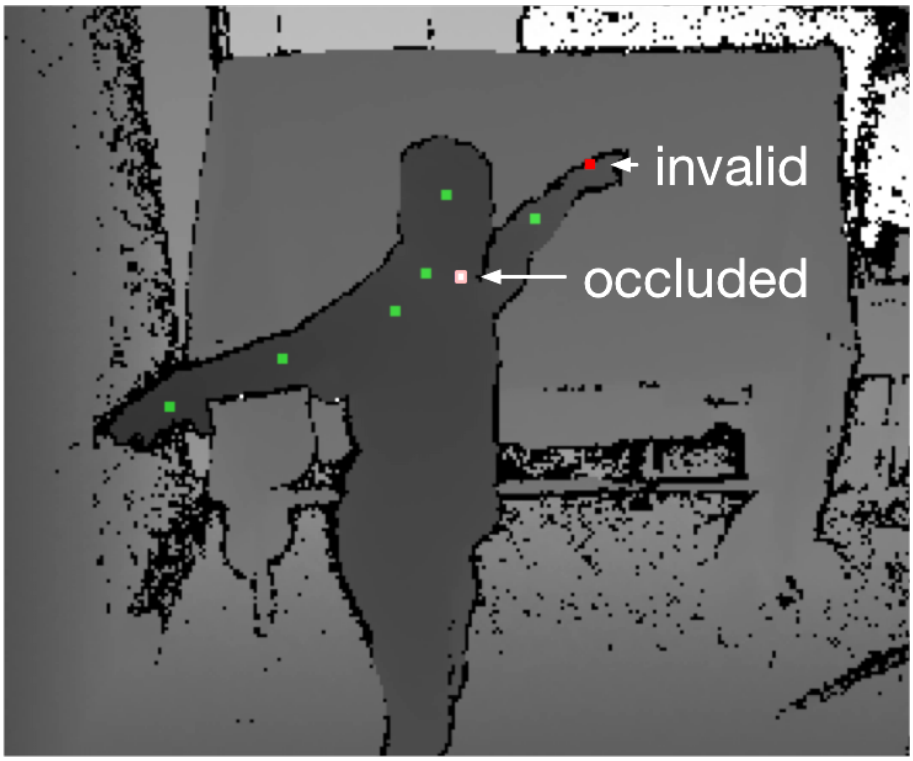


Figure 5.2: Example result of the joint correctness classification
*All three possible correctness values for joints are shown in this image of a person standing sideways. The **white** joint (left shoulder) was labeled as occluded because it is hidden behind the head and chest. The **red** joint (left hand) was labeled as incorrect because it is not in the hands center, but rather on its edge. All other joints (**green**) were labeled as correct.*

The correctness values are then used to synchronize the input images to the training labels with an optimization algorithm. For each recording, a timing offset between both is found that would minimize the number of incorrect joints. Visual inspection additionally ensured, that this offset is indeed correct and there is no delay between movements in the depth image and movements of the skeletons.
The correctness values are also used to filter the training data and exclude wrongly

labeled data. Only those frames are used, where all labels are correct or occluded. As soon as there is one incorrect joint, the complete frame is rejected. Otherwise, there would be visible joints which are not labeled. Training on this data would decrease the performance because detecting such a joint would be wrong according to the label.

After excluding all frames with incorrect joints, about 25.000 frames remained. The dataset with these frames is called C25K dataset.
In order to use this data to train CNNs, the labels need to be transformed from skeletons into heatmaps, which represent the desired output of the CNN. With the algorithm shown in code A.7, one such heatmap was created for every joint in every frame.

5.5 Results

In total, 44 labeled motion sequences across all subcategories with a length of $\approx 30-60$ seconds each were recorded, which results in roughly 100,000 frames. The complete list of included motion sequences can be found in the appendix A.3.2. After post processing, about 25,000 usable frames remained. By using data augmentation like described in section 6.2.1, a dataset of 150,000 frames was created. Additionally, three labeled motion sequences of about 30 seconds with a total of ≈ 700 frames were recorded for validation and augmented to 1,400 frames by flipping them along the vertical axis. A list of all datasets that were recorded with the methods described in this chapter is shown in tab. 5.3. The workflow that was used to create them is displayed in figure 5.4.
For testing the limitations of the trained system, additional datasets were needed later on. For comparison with the state of the art, RGB images are also necessary. But since the Captury Live system was only available for a short period of time, it could not be used for label creation anymore. A new set containing motion sequence V2 including RGB images was recorded and 127 frames were labeled manually (see section 8.2). Further, sets with difficult poses, multiple subjects and difficult clothing were recorded without labels. Even though they are not usable for a quantitative

Name	Frames	Description
C25K	25,000	initial dataset after post processing
C150K	150,000	C25K with 5 additional augmentations for each frame
CG4w	5,000	all frames of C25K that contain the motion sequence G4
CEval	1,400	all frames in the validation motion sequences (700) with augmentations
CEvalW	500	set with only the motion sequence V2 with augmentations

Figure 5.3: All datasets that were created for this thesis using CaturyLive
All datasets contain IR images, depth images and labels in form of 3D skeletons with 8 joints and corresponding heatmaps.

evaluation, they are used to evaluate the limitations qualitatively in section 8.3.

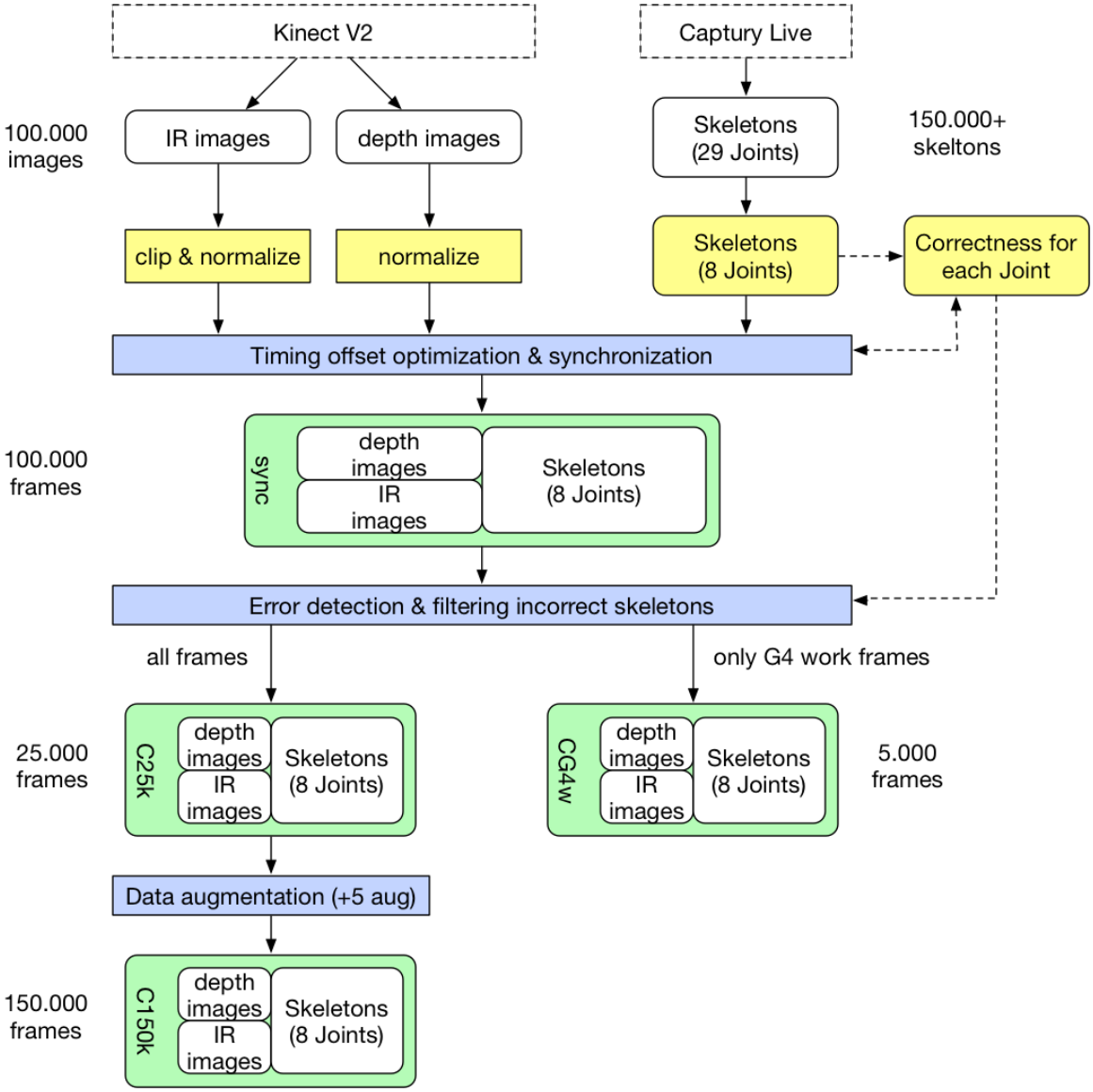


Figure 5.4: Workflow used to create the datasets

This graphic displays the methods (blue) that were used to create the final datasets (green) from the raw data (white). The yellow colored elements are used to assist this process by some means of normalization. The validation datasets are processed accordingly.

Chapter 6

System Design and Development

As shown in chapter 2 almost all state of the art approaches are creating pose estimates by using CNNs to process image data quite successfully, especially most recent ones. Because the output of a depth sensor can be represented as images, it is possible to build upon those promising state of the art results by implementing a similar CNN architecture. More specifically, section 4.2 has shown that a combination of IR and depth images is the most beneficial representation for this purpose an will therefore be used as input.

Despite this decision, a very broad scope of development possibilities for a concrete architecture still remains. Especially because the resulting system is expected to produce pose estimates in 3D, which is still comparably rare in the state of the art.

In order to guide the development process and allow for informed decisions, the training data recorded in chapter 5 was used to perform systematic experiments. The resulting development process can be divided in three parts and will be presented in this chapter. First, a well performing state of the art architecture was determined to serve as a basis for further development. This basic design was then improved regarding speed and detection rates by applying different methods. Finally, multiple possibilities for pose estimation in 3D using this improved design were examined.

6.1 Base Architecture

While there are many different approaches for Human Pose Estimation with a CNN, a lot of them rely on similar base architectures. As a first step to narrow down the possibilities, three such popular designs were identified and compared. Based on this comparison, a base architecture design was chosen as a foundation for further development.

As a first option, the Convolutional Pose Machine (CPM) was chosen because it's basic concepts are well established and often used, most recently in what is arguably the best Human Pose Estimation system to date, OpenPose [Cao et al., 2017].
In contrast, the Stacked Hourglass (SHG) is comparably new but happens to be increasingly popular in recent approaches, featured in many of the best performing systems to date (see table 2.9). Therefore, it was chosen to be the second option
Finally, the ResNet was chosen to be the third option because it is often used for feature computation by approaches for image processing and Human Pose Estimation. Even though is is usually extended by a specialized detection or regression head, it will make for a good baseline.

All three architectures were implemented in Caffe 1.0 [caf, 2017], either by directly using publicly available code or by manually implementing them as described in the corresponding papers. Additionally, small modifications were made to adjust them for the scenario of this thesis and allow for the fastest possible speed. If, for example, the original architecture uses multiple stages to refine the estimation result, only one stage was used.
In order to compare the performances, all architectures were trained on the C25K dataset which uses two input channels, one for the depth image and one for the infrared image. The performance was then evaluated using the CEval dataset (datasets described in section 5.5).

The following subsections will first introduce the three candidates, describing their

implementation and performance in more detail. Finally, a comparison of all three architectures will be given in subsection 6.1.4.

6.1.1 Convolutional Pose Machine

The principle of the Convolutional Pose Machine (CPM) is described in appendix A.2. Its specific implementation was taken from [gitb], featuring the following characteristics:

As input, the original CPM takes images at a resolution of 368×368 px with four channels. That is three channels for the RGB image and one additional channel containing a Gaussian peak that denotes the center of the primary subject (person). The images were also preprocessed by cropping them around this center peak.
This input was then processed by six consecutive stages, each refining the result of the previous one while also considering low-level features computed from the input image directly. Finally, the last layer with 15 channels produces one heatmap per joint at a resolution of 46×46 px.

In order to allow training on the C25K dataset, the input layer was adapted to take two-channel images at 368×368 px. Aiming at high framerates, any preprocessing like center peak computation or person centric cropping was dropped. For the same reason, only a single stage of the CPM was used. Finally, the output was reduced to eight channels at 46×46 px to only predict the upper body joints.
The resulting architecture will be called **CPM vanilla** in this thesis.

This CNN with 30,571,504 free parameters was trained for 60,000 iterations with the hyperparameters described in [gitb] on the C25K dataset. The resulting performance (last epoch) on the evaluation dataset CEval is shown in figure 6.1.

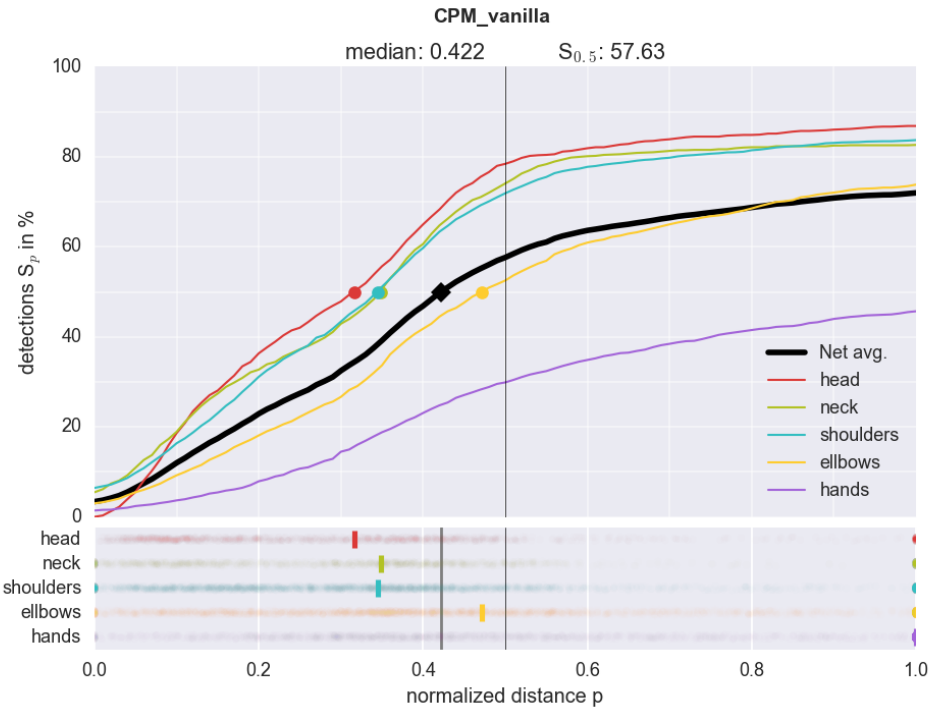


Figure 6.1: Performance of the Convolutional Pose Machine

The reference implementation of the Convolutional Pose Machine is able to detect the head, neck an shoulders equally good with a fairly equal distribution of error distances for $p < 0.5$. *While the detection rate for shoulders is just about acceptable, it is not sufficient for the hands.*

6.1.2 Stacked Hourglass

As described in more detail in section 2.2.2, the Stacked Hourglass (SHG) exploits residual learning on multiple scales and is used in some of the best performing approaches at the time. Its specific implementation was taken from [Alejandro Newell, Kaiyu Yang and Deng, 2017] and manually translated to the Caffe framework.

Originally, the CNN takes images at at a resolution of 256×256 px with three channels (RGB). Then, two successive hourglasses and finally two fully connected layers are used to produce 16 heatmaps at 64×64 px, one per joint estimate.

This architecture was modified slightly to accept a two-channel image as input and employed only a single hourglass to produce eight heatmaps for the upper body joints. The resulting architecture will be called **SHG vanilla** in this thesis.

This CNN with 17,368,904 free parameters was trained for 60.000 iterations on the C25K dataset. The resulting performance (last epoch) on the evaluation dataset CEval is shown in figure 6.2.

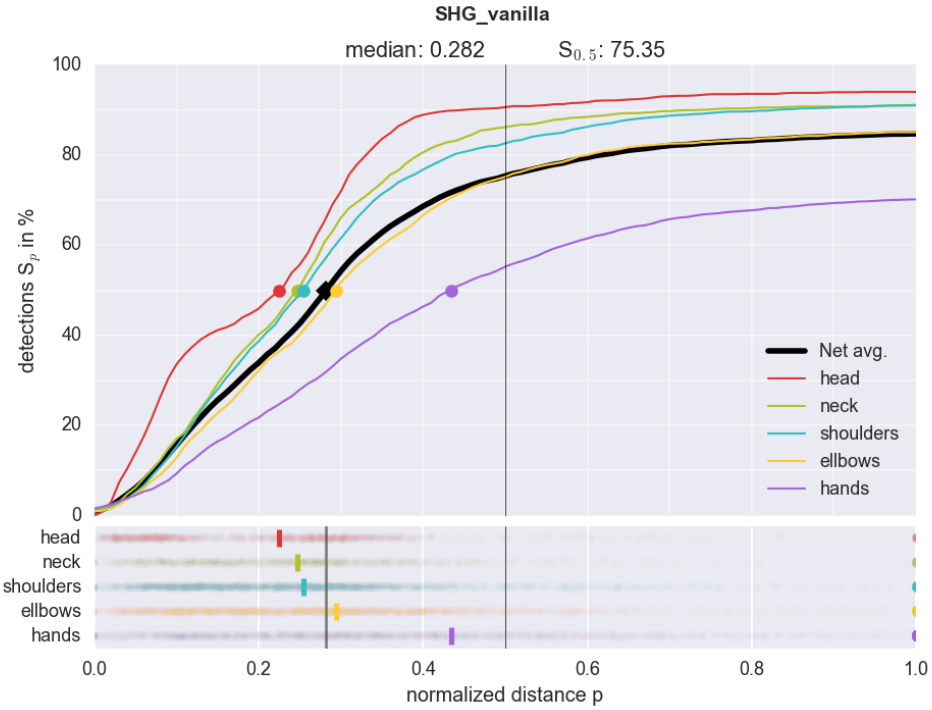


Figure 6.2: Performance of the Stacked Hourglass

The reference implementation of the Stacked Hourglass is able to detect the head, neck, shoulders and even elbows equally good with a fairly equal distribution of error distances for $p < 0.4$. *However, the distribution for the head's detection errors is not as equal and shows two distinct peaks around* $p \approx 0.05$ *and* $p \approx 0.4$. *Although the hands are the joints with the weakest detection, their detection rates are acceptable.*

6.1.3 ResNet 50

Similar to the SHG, the ResNet50 (RSN) also exploits the technique of residual learning as described in section 3.2.2. Multiple different ResNet architectures were introduced by [He et al., 2016] and are used since then in several approaches as a base to build upon, like for example in [Insafutdinov et al., 2016], [Mehta et al., 2017] and [Bulat and Tzimiropoulos, 2016]. These approaches use the ResNet mainly for feature computation and implement specialized modules for pose estimation on top. Although these approaches don't use the pure ResNet without any additions, it is a good baseline for comparison.
In [He et al., 2016], multiple ResNet architectures of varying sizes are proposed. For a fair comparison, the smallest architecture ResNet50 was chosen, because it matches the other architectures the closest regarding wight count and inference time.

The original implementation available at [gita] uses images at 224×224 px with three channels (RGB) as input. Multiple successive convolutions with stride two are then employed, downsizing the image resolution to eventually 7×7 px. Finally, pooling with kernel size seven, a fully connected layer and succeeding softmax produce a vector of length 1,000 as output. This vector is usually used for classification or as input features for succeeding specialized modules.
The described original architecture was modified to take a two-channel image at 256×256 px as input. To keep the output resolution reasonable for heatmap generations, the stride of all convolutions was reduced to one. This resulted in a constant resolution of 64×64 px after an initial convolution with stride two and pooling. Finally, the last layers for prediction were replaced by a single convolution to produce eight heatmaps at 64×64 px, one for each upper body joint.
The resulting architecture will be called **RSN vanilla** in this thesis.

This CNN with 23,578,504 free parameters was trained for 60.000 iterations on the C25K dataset. The resulting performance (last epoch) on the evaluation dataset CEval is shown in figure 6.3.

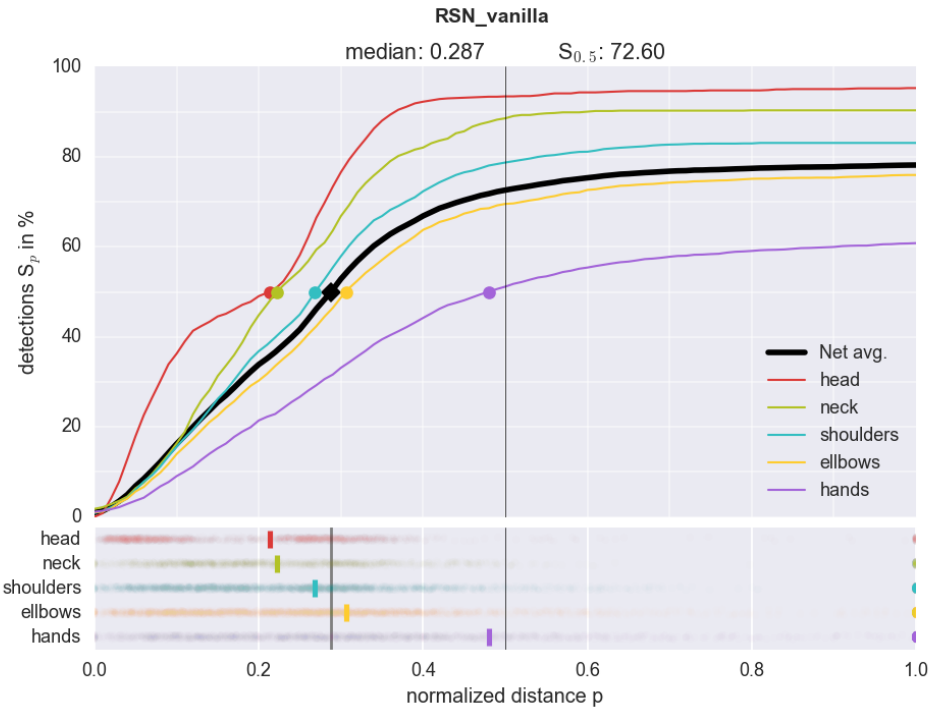


Figure 6.3: Performance of the ResNet50

The detection rates of the ResNet50 reference implementation gradually decrease along the kinematic chain. While the error distribution for all joints is fairly equal for $p < 0.4$, the head's error distribution shows two distinct peaks around $p \approx 0.05$ and $p \approx 0.4$. Although the hands are the joints with the weakest detection, their detection rates are just about acceptable.

6.1.4 Conclusion

A comparison of the three evaluated approaches (Stacked Hourglass (SHG) , Convolutional Pose Machine (CPM) and ResNet50 (RSN)) is shown in figure 6.4, illustrating the difference in average performance and inference time (speed). First, only the average performance (left) will be examined.

As an initial observation, the RSN and SHG performed quite similar while the performance of the CPM is noticeably lower. In fact, the median detection error of ≈ 0.28 is virtually the same for RSN and SHG, but roughly 50% higher for the SHG with ≈ 0.42. Based on those observations, the CPM is assumed to be the least eligible

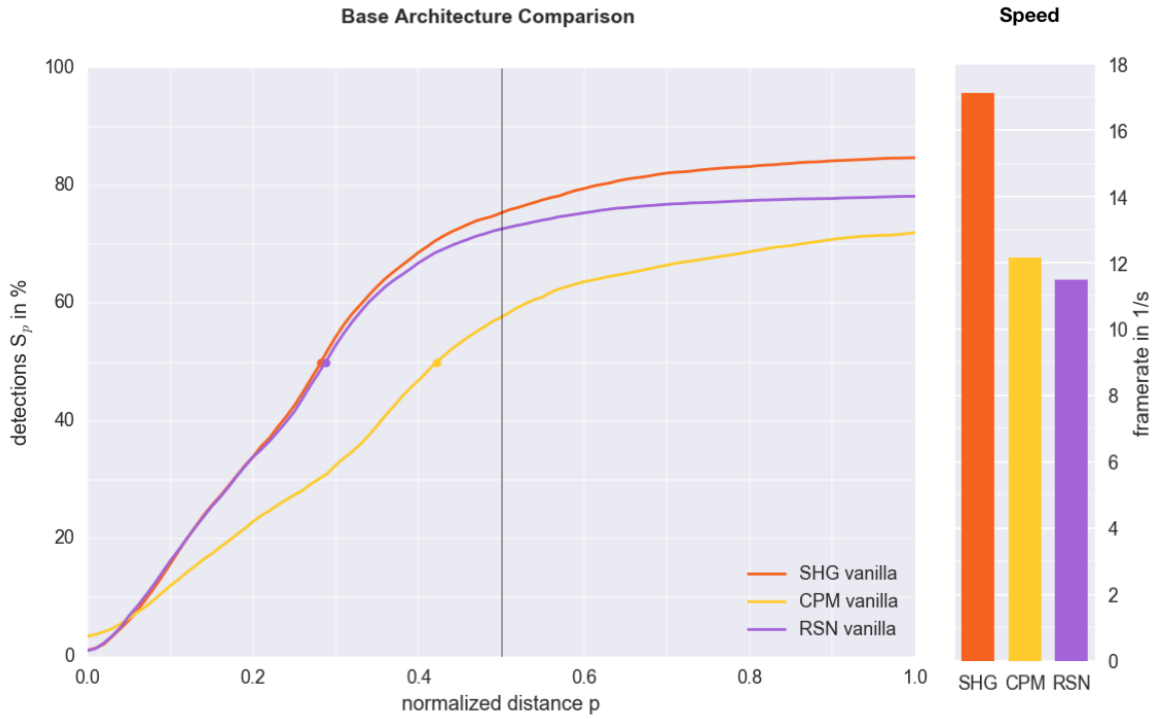


Figure 6.4: Precision comparison of base architecture candidates
*A reference implementation of the Stacked Hourglass(SHG vanilla), Convolutional Pose Machine(CPM vanilla) and ResNet50(RSN vanilla) were trained for 60.000 iterations on the C25K dataset. The **left** graph shows the average detection rates of these networks on the CEval evaluation dataset. The **right** graph depicts the average inference speed in frames per seconds (higher is better).*

architecture and excluded from further discussion.

By taking also the detailed results (shown in figure 6.2 and 6.3) into consideration, more differences between the remaining two approaches become apparent. The RSN's detection rates are more diverse and spread out across the individual joints compared to the SHG. More precisely, its detection rate decreases more drastically for joints further down in the kinematic chain. This is especially apparent when comparing the hand's detection rate, which is noticeably lower for the RSN.
Additionally, the SHG's detection rate continues to increase for distances $p > 0.5$

where the RSN's detection rate shows little to none gains (see figure 6.4). Because of that, the SHG has a higher maximum detection rate.

A likely explanation for this was found by visual inspection of the detection results. The differences were most prominent in difficult situations: while the RSN tended to fail the detection entirely, the SHG was more often able to at least "guess" a joints location approximately, like shown in figure 6.5. Even though this "guess" was not very exact, such behavior might indicate potential for further improvements. For example, more training data could help to refine those "guesses". This will be examined in subsection 6.2.1.

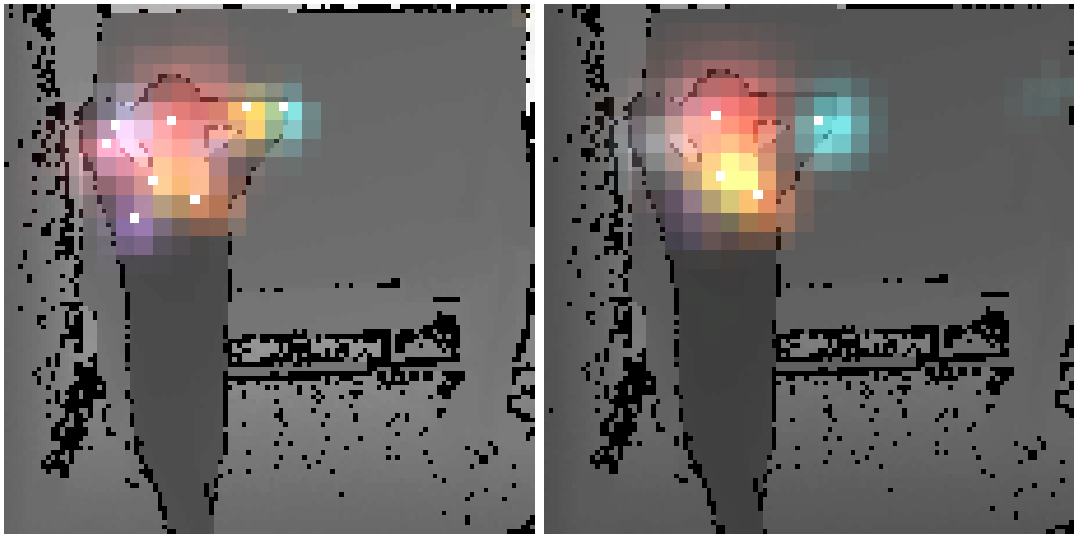


Figure 6.5: Comparison of Stacked Hourglass(**left**) and ResNet50(**right**) performance

*The person in this image (zoomed for better visibility) performs an especially difficult task by standing in great distance to the camera and placing both hands behind the head. Although the SHG's estimation (**left**) is not very precise, still all joint locations (white dots) are estimated and even quite close to the actual location. The RSN (**right**), on the other hand, fails the detect or even guess half of the joints.*

Altogether, the SHG seems to perform slightly better than the RSN, reaching higher maximum detection rates while struggling less on difficult joints further down the kinematic chain.

Since the resulting system is expected to perform in realtime, a speed comparison like in figure 6.4 (right) is also highly relevant. Regarding speed, the SHG performs clearly the best with $\approx 40\%$ higher frame rates.
Combining the results of the detection rate and speed comparisons, the SHG is therefore the best suitable architecture of all three. In the next sections, it will be used as a basis for further refinements and improvements.

6.2 Performance Improvements

Starting from the base architecture established in section 6.1, several approaches to increase the performance are possible. They either focus on increasing the accuracy and subsequently the detection rate or on decreasing the inference time and allowing for higher frame rates. In this section, three approaches for performance improvements will be examined.

6.2.1 Training data augmentation

First, it might be possible to improve the accuracy of the detection result. When training a CNN, often times the quality or quantity of data can be a big factor for accuracy. More precisely, for training a deep CNN the 25.000 frames of the C25K dataset might not be enough. Especially since the camera's position is constant across all frames, there might not be enough variance to allow for good generalization. As previously mentioned in section 6.1.4, there are already indications that more (and more diverse) training data might be beneficial.
A common technique to overcome such shortages in training data is data augmentation. Depending on the type of data, new training samples are generated by introducing a random amount of specific modifications to the available data. For image data, the most common modifications are rotation, translation and scaling, as employed by many state of the art approaches for Human Pose Estimation. More specifically, the following modifications were employed:

For every frame of the training datset **C25K**, five additional frames were created by randomly applying one of the following augmentations, resulting in a total size of 150.000 frames (**C150K** dataset):

- **mirroring:** flip image across the vertical axis
- **rotation:** randomly rotate up to $\pm 30°$, border filled with nearest values
- **translation:** randomly move up to $\pm 20\%$ of the total length for each axis, borders not filled (black)
- **scaling:** randomly scale up to $\pm 25\%$ of the original size around a randomly chosen center point (resulting in scaling and translation). The borders are not filled (black). Also, the depth values are scaled accordingly to keep dimensional relationships constant. An upscaled image with unchanged depth values would represent a unrealistically giant human otherwise

Furthermore, the validation datasets proved to be a little bit to small for an objective and expressive evaluation. Also, they slightly favored the left joints over the right because the poses were not evenly distributed in respect to the vertical axis. Therefore, the validation datasets were also augmented by adding the mirrored version of every frame.

Conclusion

To evaluate the improvements by training data augmentation, two networks were trained on the C25k and the C150k dataset respectively. Building upon the results of section 6.1, the **SHG vanilla** architecture was used for both. As shown in figure 6.6, data augmentation could drastically improve the detection rates. The median detection error decreased about 50% to 0.15 while the maximum detection rate increased about 25% to $S_{0.5} = 93\%$.

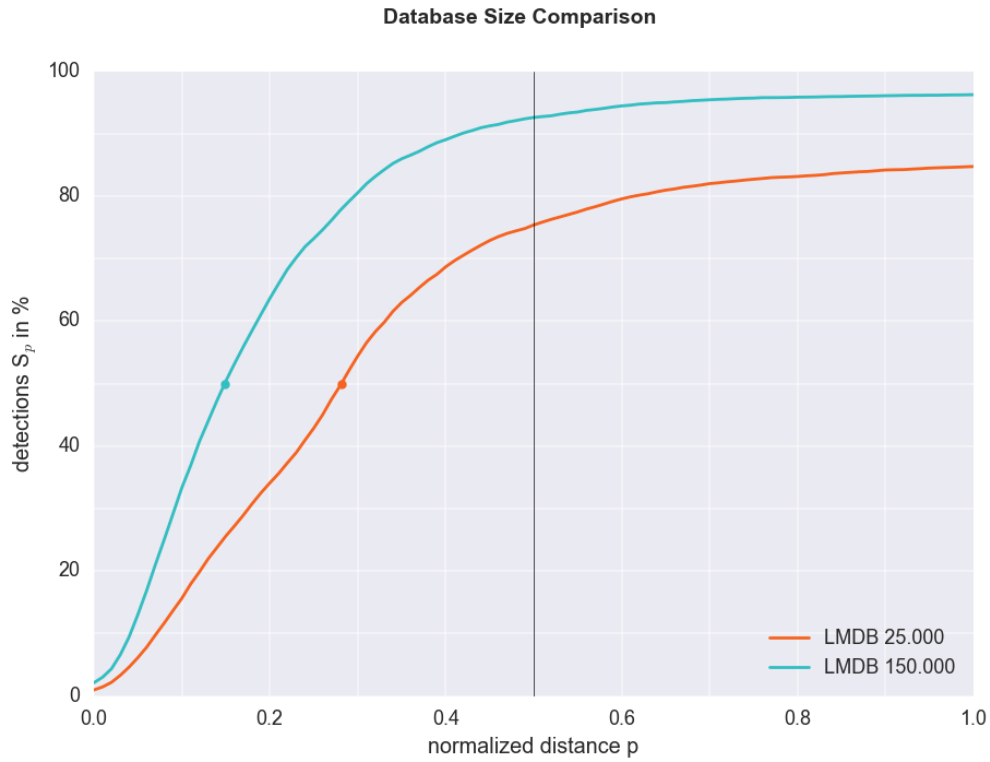


Figure 6.6: Performance increase by data augmentation

Training on the augmented dataset with 150.000 frames significantly improved the performance compared to the initial dataset with 25.000 frames.

6.2.2 Weight Optimization Using the Layer Inspection Tool

A lot of architectures are created generously to avoid bottlenecks by too few or too small layers in advance, even though not all weights and layers might be needed for a specific task. However, the number of weights and layers has influence on training time, inference time and overall performance. A higher number of weights usually increases training time while a higher number of layers increases inference time.
If it would be possible to identify and cut such dispensable weights and layers, the networks inference time, memory usage and training time could be improved while preserving the networks accuracy. With deep CNNs, however, it is often hard to understand what happens internally, which makes such an identification rather difficult. Therefore a tool that allows to inspect a network after training will be very helpful basis for meaningful architecture optimization.
The only available tool for this purpose that works with networks built in Caffe is the Deep Visualization Toolbox [Yosinski et al., 2015]. However, it only works with networks that take a three channel RGB image as an input. Modifying the source code to allow it to work with two channel networks proved to be quite time consuming and not easily possible. Therefor, a new tool called Layer inspection Tool (LiT) was developed for deep neural network layer inspection. Its purpose is to visualize weight usage and indicate possibilities for architecture improvements by providing statistics for every convolution layer.

In this section the LIT's visual interface will be introduced and the results of the realized weight reduction are presented. The mathematical background for the visualization is described in appendix A.4. In the subsequent sections, a different approach for reducing the number of weights and a final comparison will be presented.

Visual Interface of the Layer Inspection Tool

The visual interface of LIT for a typical, healthy layer is displayed in figure 6.7. It is always displaying information for one specific layer. The most prominent element is the **weights matrix (B)**. It displays the raw data of the layers weights, color

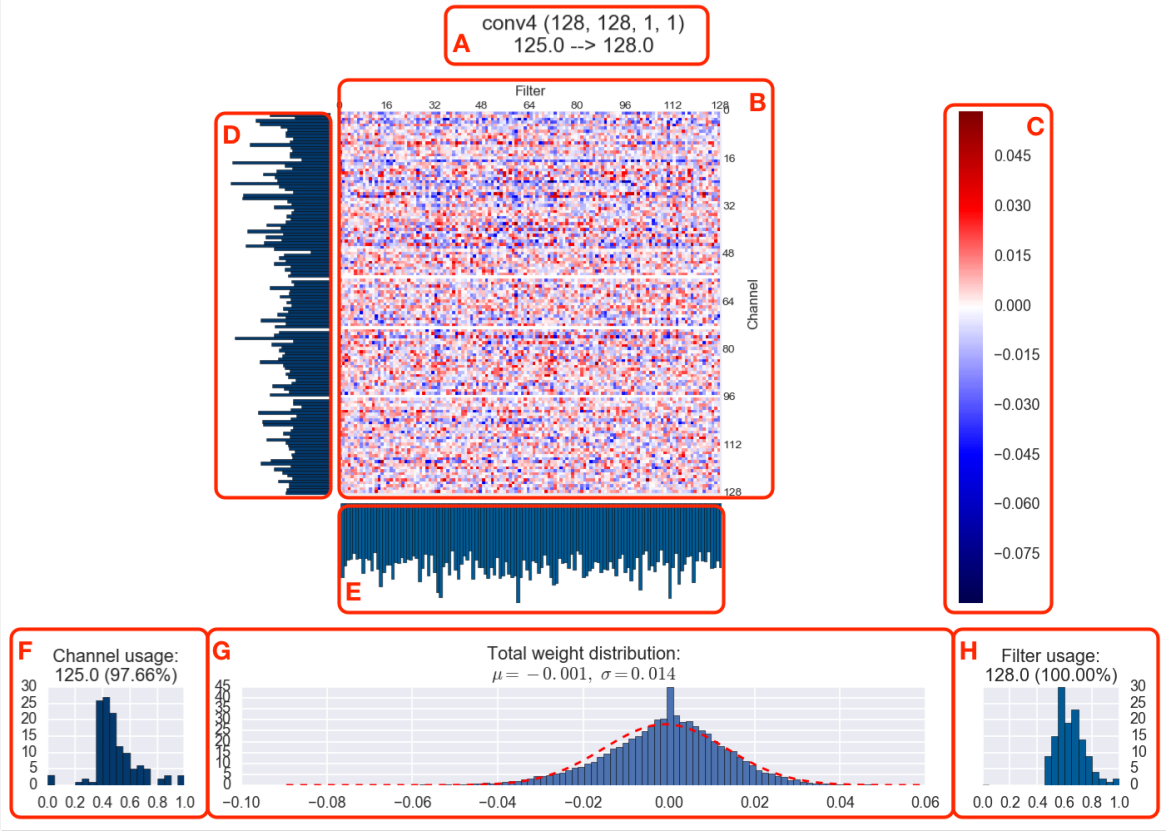


Figure 6.7: Overview of the visual interface of LIT

***(A) header**: layer name, (channel count, filter count, filter dimensions) channel usage –> filter usage*

***(B) weights matrix**: visualization of all weights in this layer (raw data)*

***(C) color key**: color coding of the weights matrix*

***(D) channel significance**: significance for each channel (input)*

***(E) filter significance**: significance of each filter (output)*

***(F) channel usage**: histogram of channel significance*

***(G) total wight distribution**: histogram of channel weights matrix*

***(H) filter usage**: histogram of channel significance*

coded like shown in the **color key (C)**. Each weight connects an input channel on the y-axis with a filter (output channel) on the x-axis. Weights close to zero appear white, strong positive weights appear red and strong negative weights appear blue.

The weight matrix allows for a first, quick visual inspection of the learned weights and helps to identify possible directions for improvements. For example, all weights for channels 56, 73 and 96 in figure 6.7 are zero, resulting in a white row. This indicates, that those channels have no influence on the layers output and therefore are not necessary. However, since there are no white columns, it can be concluded that all filters produce output that is useful for later layers. If there would be a great number of unused channels or filters, this can be spotted very quickly in the weights matrix. The **total weight distribution (G)** additionally displays the distribution of absolute weight values as a histogram with a fitted normal distribution (red dotted line).

For a more exact description of the importance of each channel and filter, **channel significance** $\mathcal{M}_c$ and **filter significance** $\mathcal{M}_f$ are calculated (see appendix A.4) and displayed as bar graphs **(D)** and **(E)** next to the weights matrix. Again, white gaps indicate unused channels and filters. Also, the distribution of bar lengths correlate to the distribution of strong and weak channels and filters. Fig. 6.7 shows the distribution of an average, healthy layer. By contrast, mostly equal bar lengths usually indicate that the layer hasn't learned anything at all and is still very close to its initialization configuration

A more easily readable display of this distribution is provided in the **channel usage histogram (F)** and the **filter usage histogram (H)**. They depict the number of channels and filters with a specific significance. Channels and filters with a significance below 0.05 are considered as unused. Their quantity is represented by the height of the left most bar and is also described numerically above the histograms. This is one of the most relevant pieces of information for weight reduction. A high quantity of unused filters for example suggests, that the layers number of filters can be reduced without impacting the performance.

Finally, the most important information like layer name, filter count, channel count, filter dimensions, used channels and used filters can be found in a brief summary in the **header (A)** on the very top.

Results

The indications of the LIT were used to conservatively remove weights, always leaving more weights than necessary. This was done by reducing the number of output channels for each layer according to the channel usage. As a result, the number of weights was reduced from 17,368,904 to 7,442,797, which corresponds to a removal of more than 55% of the original weights. The final, reduced architecture is presented in section 7.3 in more detail.
An evaluation of the resulting differences in accuracy and speed will be given in subsection 6.2.4. Further inspection has shown, that there is still a good amount of unused weights remaining. Additional less conservative reductions might be able to improve the framerate even further.

6.2.3 Layer Reduction: Weight Merging

Besides weight reduction as part of the architecture design process, additional layer reduction is possible post training by a technique called weight merging. This allows to decrease the inference time without any impact on the accuracy.
Weight merging can be applied for layers that perform static operations which are independent on the input data, for example multiplication or addition with a constant. Instead of performing these operations successively during inference time, they can be computed in advance by merging their weights with the weights of the previous layer. Because inference for the resulting combined layer can be calculated in a single step, weight merging allows to decrease the overall inference time by reducing the number of non parallelizable computation steps.
In particular, the batch normalization layer (see 3.2.2) is such a mergable layer and heavily utilized in the used CNN architecture. In this arcitecture, it always succeeds a convolutional layer.
As explained in section 3.2.2, a batch normalization layer uses the four internal parameters $\bar{\mu}, \bar{\sigma}, \gamma$ and β to create the output $\mathbf{y}$ by scaling the input $\mathbf{x}$ like shown in

equation 6.1.

$$\mathbf{y} = f(\mathbf{x}, \bar{\mu}, \bar{\sigma}, \gamma, \beta) = \gamma \frac{x - \bar{\mu}}{\bar{\sigma}^2 + \epsilon} + \beta \tag{6.1}$$

All four of them are learned during the training process. The parameters $\bar{\mu}$ and $\bar{\sigma}$ correspond to the inputs mean and variance and are accumulated over the whole training data. Like the actual scaling parameters γ and β, they remain constant after training. Therefore, they can be merged with the weights $\mathbf{W}_C$ and bias $\mathbf{b}_C$ of the preceding convolutional layer, creating the merged weights $\mathbf{W}_M$ and merged bias $\mathbf{b}_M$.

$$\begin{aligned} \alpha &= \frac{\gamma}{\sqrt{\bar{\sigma} + \epsilon}} \\ \mathbf{W}_M &= \mathbf{W}_C * \alpha \\ \mathbf{b}_m &= \alpha * \mathbf{b}_C + \beta - \alpha\bar{\mu} \end{aligned} \tag{6.2}$$

The equations in 6.2 were implemented as an algorithm that merged all batch normalization layers into the preceding convolutional layers. This decreased the number of layers from 363 to 185, which resulted in a layer reduction of 51%. Consequently, the framerate increased from 24.21 fps to 58.58 fps which extends an improvement of more than 140%. The number of weights, however, reduced only by 0.85%.
A comparison to the performance of the original **SHG vanilla** architecture is given in figure 6.8 in the next section.

6.2.4 Conclusion

In section 6.1, the SHG architecture was found to be the best performing for the task. Building upon this basic design, three approaches for improvements were presented. While the data augmentation described subsection 6.2.1 could increase the accuracy and detection rate, weight reduction as in subsection 6.2.2 and successive layer reduction as shown in subsection 6.2.3 aimed to increase the CNNs speed. The results of these improvements are displayed in figure 6.8.

Because weight reduction applies prior to training, the improved minimal architecture needed to be trained again. For this reason, a slight difference in the detection rate

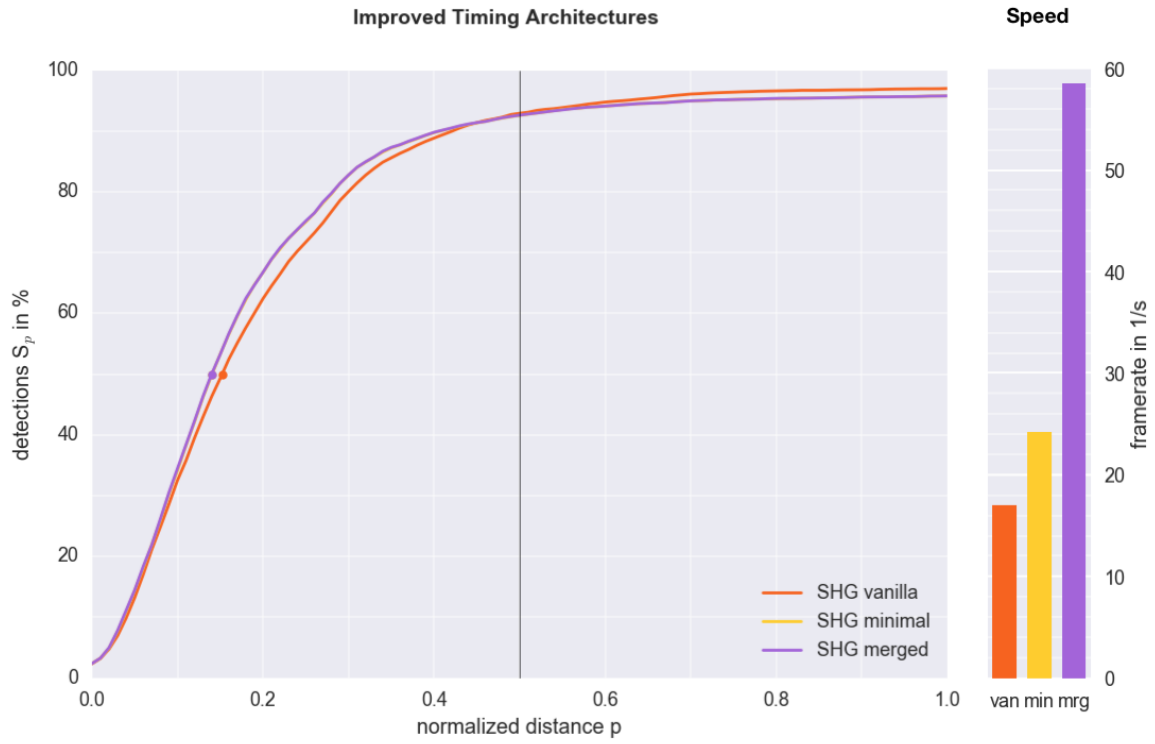


Figure 6.8: Improvements by weight and layer reduction

*The detection rates (left) and speed (right) of the original SHG design (**SHG vanilla**) are compared to the improvements by weight reduction (**SHG minimal**) and layer reduction (**SHG merged**). All variants were trained on the augmented C150K dataset. With no noticeable difference in detection rates, the speed could be increased by a factor of three. Because the merged and minimal architectures performed identical, their curves overlap each other.*

curves is apparent. However, this can be attributed to stochastical differences during the training process rather than actual differences in performance.

Layer reduction, on the other hand, was applied after training. The used process of weight merging is expected to preserve all mathematical characteristics of the CNN, resulting in identical detection rates. The performance curves of the minimal and merged architecture in figure 6.8 verify this assumption, because they overly each other exactly.

Even though the detection rates of all three architectures are virtually the same, there is a great difference in speed (see figure 6.8, left). The combination of weight reduction and layer reduction improved the framerates by a factor of three. Although weight merging only removed 0.85% of all weights, it resulted in the biggest speed improvement. This confirms that the number of layers has a bigger impact on inference speed for CNNs implemented in Caffe than the number of weights.

A visualization of the improvements presented in this section along with all experiments so far is shown in figure 6.9. Altogether, the median detection error could be decreased by about 50% while increasing the framerate by more than 250%. These results exceed the target specifications and therefore create space for further development. The next section will describe efforts to utilize this space for prediction results in 3D.

6.3 3D Pose Estimates

Besides the improvements discussed in section 6.2, multiple ways to produce 3D estimates were tested. Following the state of the art approaches presented in section 2, it might be possible to create a multitask CNN architecture for 3D prediction which can be trained end-to-end. For this, the base architecture determined in section 6.1 was extended in several ways, like shown in figure 6.10.

By simply appending multiple fully connected layers at the end, option **A** is the most straight forward extension. These additional layers are expected to predict 3D coordinates by processing the 2D heatmaps along with original input images. The original input images were also preprocessed for the 3D predictions by a varying amount of convolutions. Similar to [Bulat and Tzimiropoulos, 2016], [Martinez et al.] and [Toshev and Szegedy, 2013], two to four fully connected layers with 256 to 4,096 output channels were used for the 3D predictions, followed by a final fully connected layer with 24 outputs. The output corresponds to eight points (x, y, z) in 3D, containing one position for every joint.

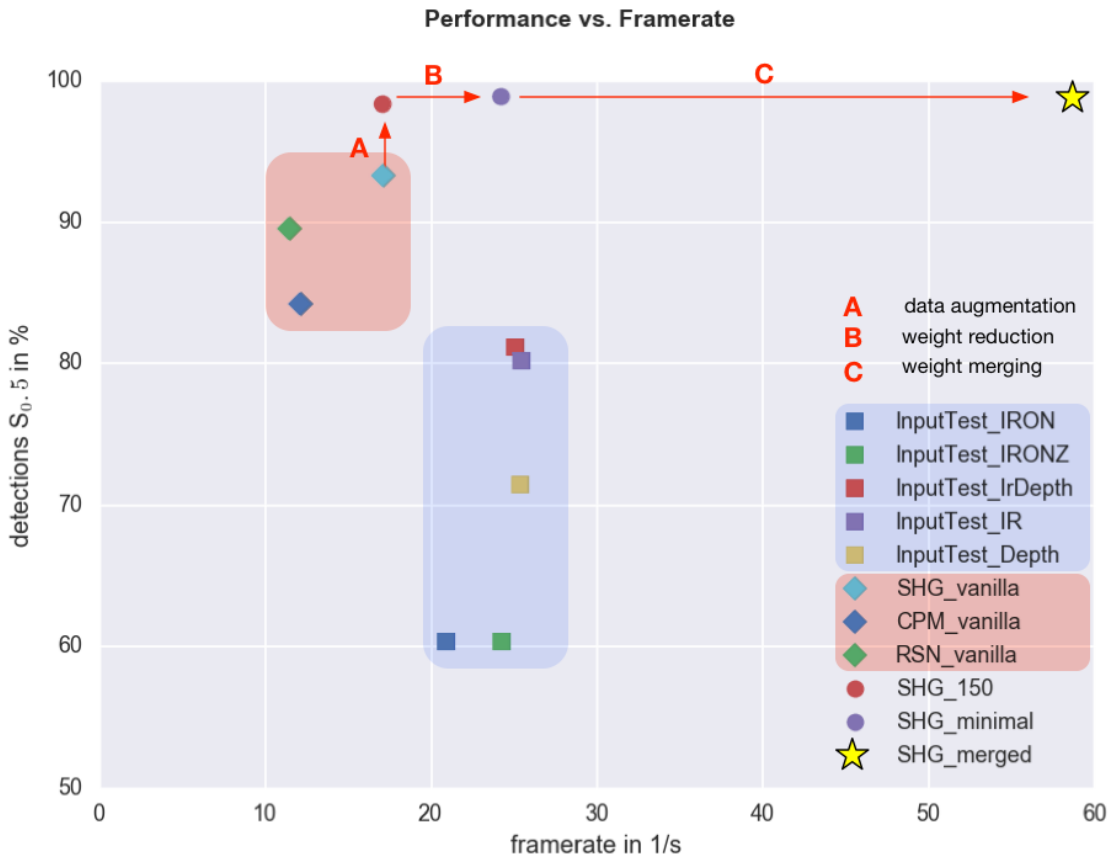


Figure 6.9: Overview of all experiments during the development

All tested architectures are plotted in regards to their accuracy and speed. The results highlighted in blue are the tests regarding the input data representation discussed in section 4.2. The base architecture comparison provided in section 6.1 is highlighted in red. The final performance improvements discussed in this section are indicated by the red arrows. Denoted by a star, SHG_merged is the final architecture after weight reduction and weight merging.

However, none of the tested configurations was able to produce reasonable results.

Alternatively, both the 2D detection and 3D regression can be performed in parallel, similar to [Tekin et al., 2016] and [Li et al., 2014] (option **B** in figure 6.10). Like in the serial architecture described above, the 3D regression was implemented using

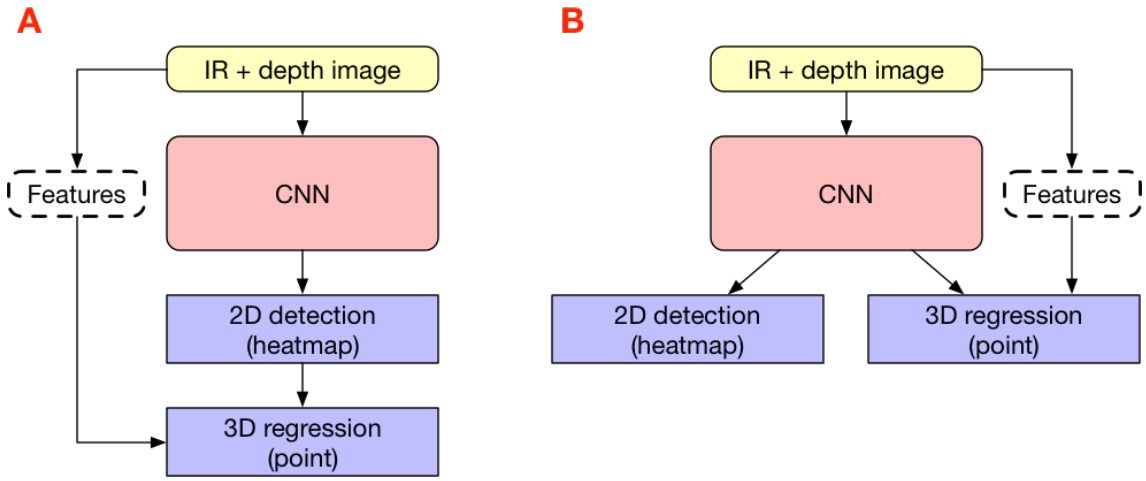


Figure 6.10: Modified architectures used to create 3D pose estimates
*The two tasks of 2D detection and 3D regression can either be tackled successively (**A**) or in parallel (**B**). In both cases, the 3D prediction uses the original depth image input either directly or by computing additional features.*

multiple fully connected layers and exploited the original input images as well. This design, however, was also not able to produce good results.

As [Sun et al., 2017] suggests, the regression might work better for bones than for joints. Following this principle, the training labels were adapted so that the regression result (x, y, z) did not denote the position of a joint, but rather a vector describing a bone. This approach was tested on all of the above architectures, but did not result in sufficient improvements.
Additionally, the composite loss function proposed by [Sun et al., 2017] was implemented as well, but without the desired effect.

Because none of the described approaches lead to sufficient results, the design of an end-to-end system was abandoned in favor of a more simple concept. Like described later in section 7.4, a lookup method was implemented. It takes the 2D position denoted by the heatmaps and reads the depth information for this position from the depth image directly.

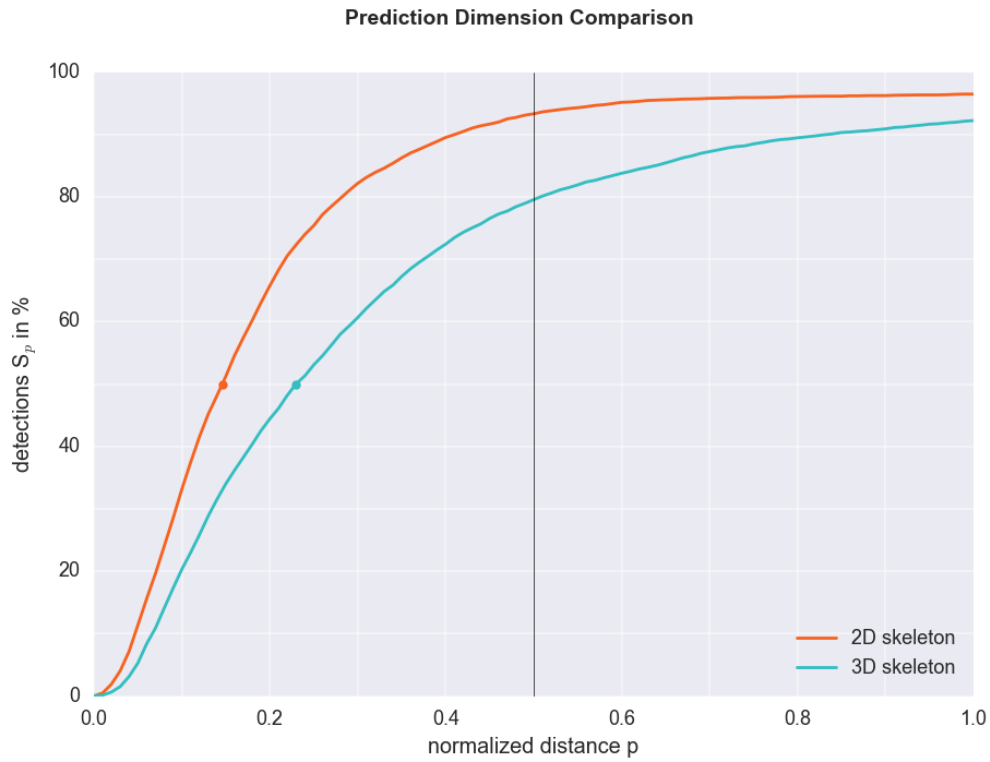


Figure 6.11: Comparison of 2D and 3D prediction performance
The prediction in 3D is not as strong as the 2D prediction. Although the maximum detection rate at $p = 1.0$ *is only slightly lower, the median detection error is about* 50% *higher.*

The performance of this 3D prediction is compared to the previous 2D prediction in figure 6.11. The results are noticably weaker in 3D than in 2D. This might be partially due to the high noise in the depth image, which results in errors while retrieving the depth information.

Chapter 7

Final Design

In the previous chapter, the process of developing a CNN architecture for Human Pose Estimation was described. This chapter will present its results by illustrating the final system for Human Pose Estimation in detail.
After the over all concept is introduced in the first section, all individual parts will be explained in the following sections.

7.1 Overview

The final system is able to produce pose estimates in 2D and 3D from single depth images. A more detailed placement in comparison to the state of the art is given in figure 7.1.

Like shown in figure 7.2, the system consists of three main parts. The focus of this thesis is heavily on the CNN that is used for the second part (**B**). Everything else is just an example to demonstrate the capabilities. In the following sections, the individual components will be explained along the signal path shown in figure 7.2.

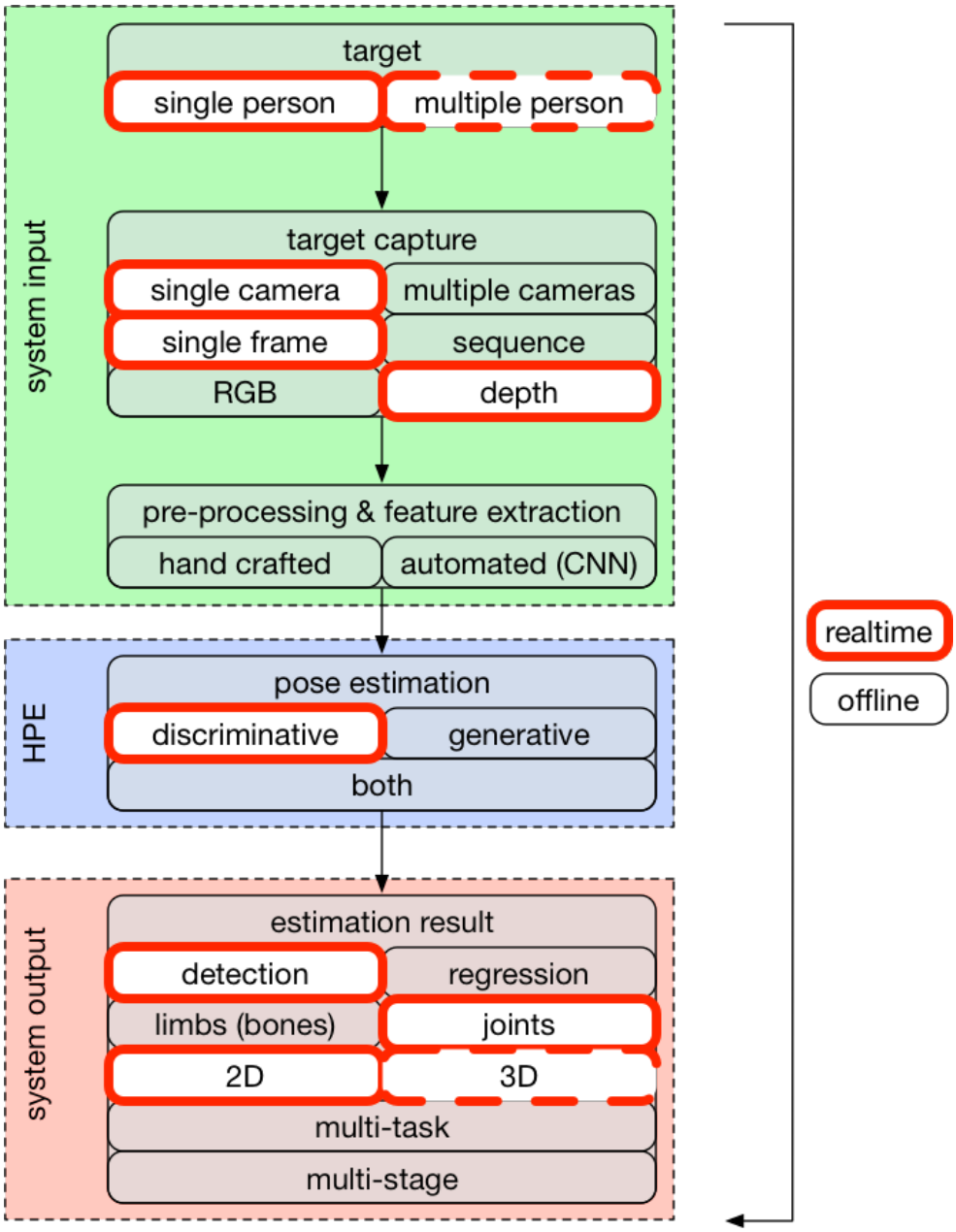


Figure 7.1: Placement of the presented approach within the state of the art structure

The presented system creates 2D joint estimates for a single and multiple persons from single depth images by utilizing a deep CNN for discriminative detection in realtime. Based on that, a 3D estimate for a single person is created as well.

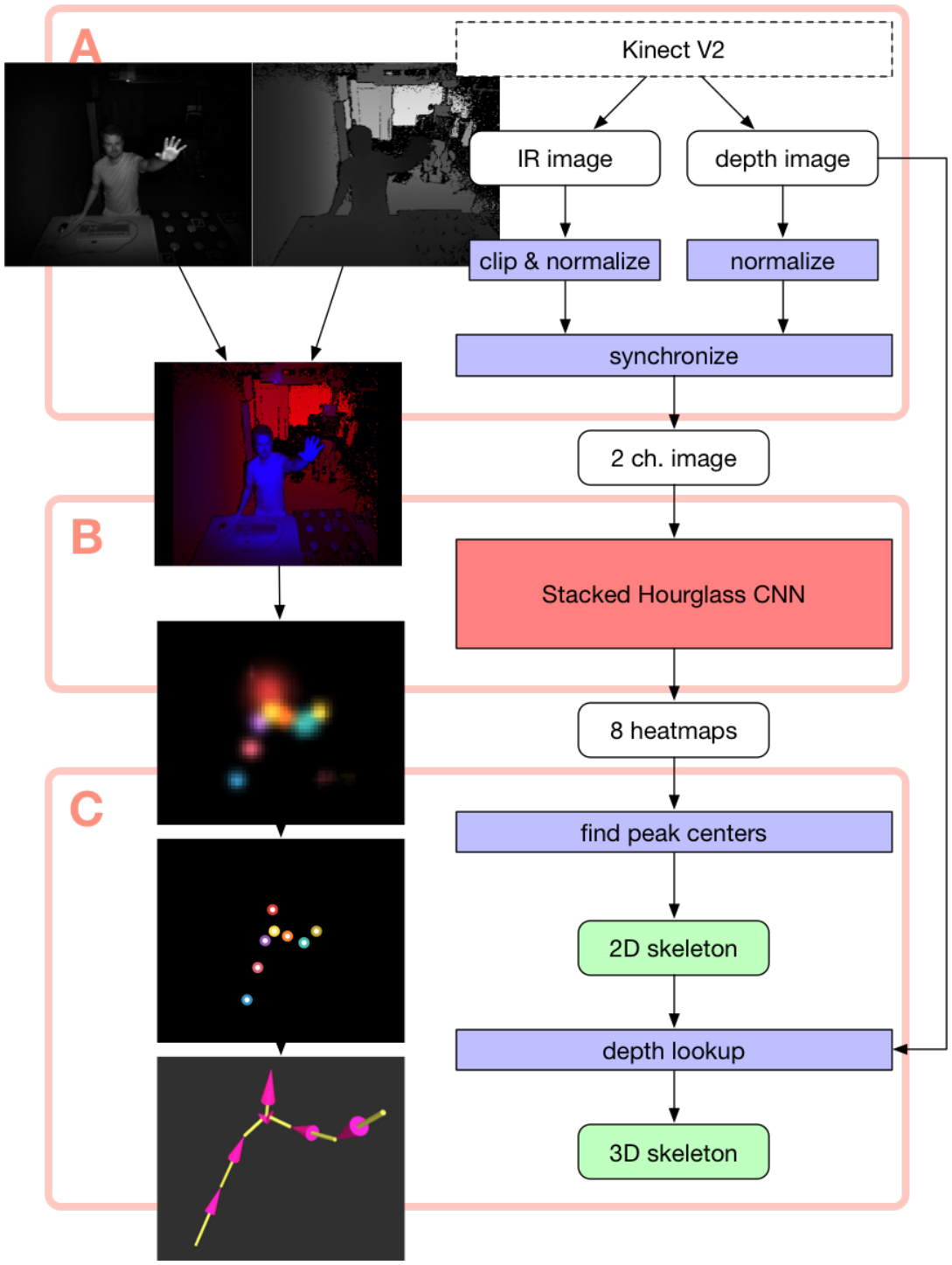


Figure 7.2: Overview for the whole pose estimation system

*The system can be divided into three different parts: a simple interface for image capture, normalization and synchronization (**A**), a deep CNN for 2D pose estimation (**B**) and two post-processing modules to create joint position estimates in 2D and 3D from the CNN output (**C**).*

7.2 Image Capture and Preprocessing

A single Kinect V2 camera is used as image source, providing both the IR and depth images. The following preprocessing steps are necessary in order to use them for pose estimation with the succeeding CNN.
First, both images are resized to 256 x 256 px. Additionally, the IR images are clipped at intensity 7,000 to be robust against bright light sources that might be visible (for example the IR emitter of another Kinect V2 camera). Next, the values of both images are normalized to be between 0 and 1. Since both images are created independently by the Kinect V2 camera, a small ring buffer is used to find pairs of images with the smallest time difference. Finally, both images are combined into a data format that can be understood by Caffe (which is the the framework used for the CNN).
All of the above steps are implemented in Python using the Robot Operating System (ROS)-Framework as middleware to communicate with the depth sensor and take about 5ms in total to compute.

7.3 Deep CNN for Human Pose Estimation

For pose estimation, a deep CNN architecture was developed (see chapter 6) and trained with the data described in section 5.5. Its task is is to produce heatmaps from depth and IR images.
In this section, the basic architecture will be introduced first. Then, the specific improvements and the resulting final architecture are demonstrated.

7.3.1 Basic Architecture

The architecture is based on the Stacked Hourglass design described in section 6.1.2. As the high level overview in figure 7.3 indicates, it can be divided in three main parts: In the first part (**A**), a set of Convolutions creates a lower resolution representation of the image. The second part (**B**) is the actual hourglass that calculates combined features with the help of ResBloks on several resolutions. Finally, multiple fully connected

layers take those combined multi-resolution features and create the final predictions (**C**).

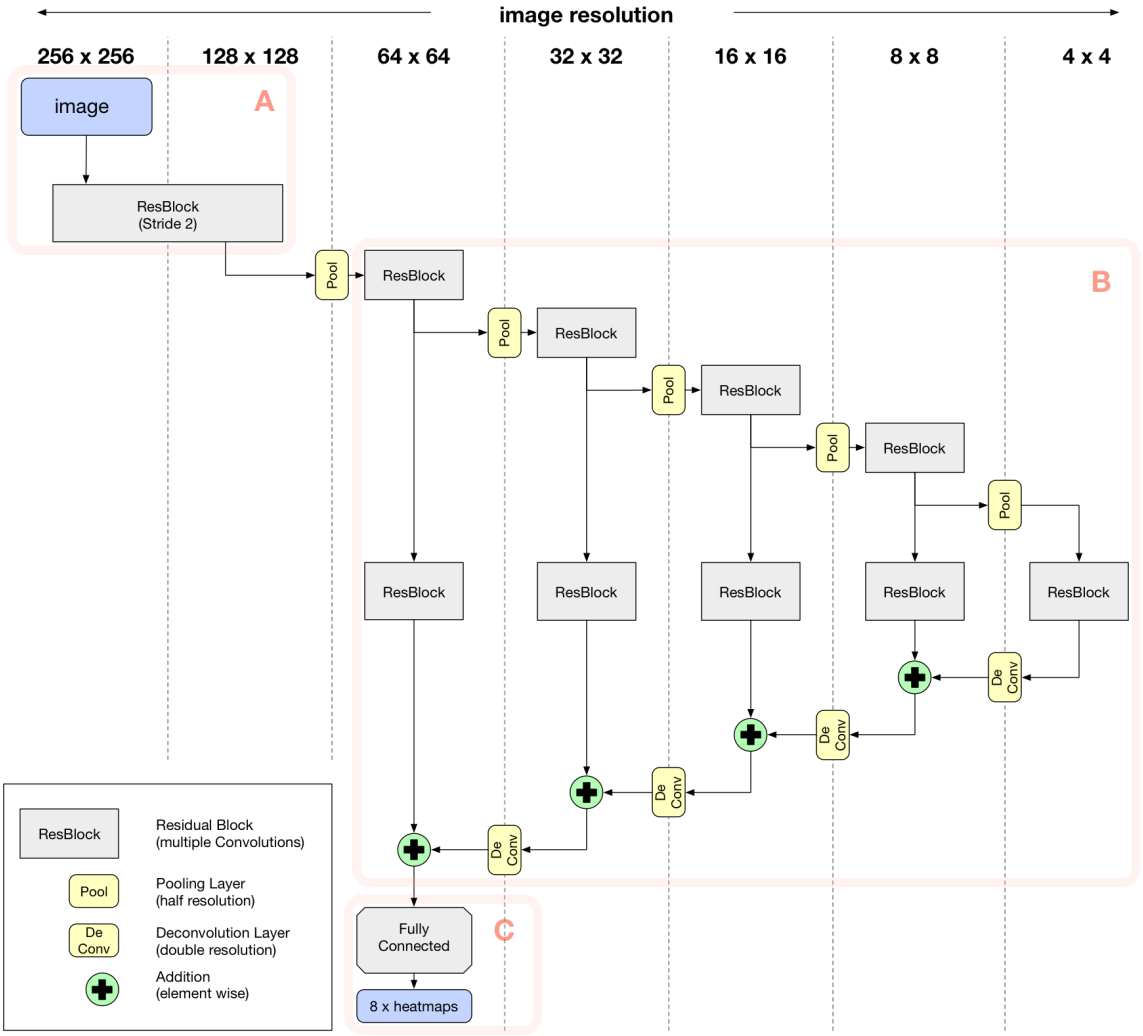


Figure 7.3: High level structure of the CNN architecture

*The architecture is based on the stacked hourglass shown in figure 2.5. After creating a lower resolution feature representation of the input (**A**), the image will be step wise downscaled (pooling) and features are calculated at each resolution by the use of ResBlocks. The resulting feature maps are then upscaled (deconvolution) and combined with the features of the next higher resolution by element wise addition (**B**). Finally, the heatmaps are generated by multiple fully connected layers (**C**).*

7.3.2 Approach for Improvements

The basic architecture was not altered and corresponds to the one described by [Newell et al., 2016]. However, the implementation of the ResBlocks was optimized to improve inference times by a factor of three. While the architecture by [Newell et al., 2016] used the exact same ResBlock with the same number of channels throughout their whole network, the design presented here was tailored to the specific task.

Following the principle of residual learning (see section 3.2.2), the features of all ResBlocks are combined eventually by element wise addition. Because of this, ResBlocks that will be added need to have the same number of output channels. However, some ResBlocks actually compute quite simple features using only a very small number of channels internally. This can be exploited for optimization by introducing bottlenecks which decrease the number of internal channels while keeping the number of output channels the same. Also, in some cases the number of output channels could be lowered as well.

Decreasing the total number of channels (and therefore weights) is desirable because it also decreases inference- and training times. Bottlenecks can be introduced at different scales between multiple ResBlocks or even within a single one. The specific implementation of those bottlenecks will be analyzed in the next section.

7.3.3 Improvements

To illustrate the made improvements, the components within a ResBlock will be described first. Each ResBlock consists of one or more successive Residual Modules. Like shown in figure 7.4, these Residual Modules differ in the number of output channels, the number of bottleneck channels and weather or not an additional Convolution is used. In [Newell et al., 2016], all Residual Modules within a ResBlock are identical and the number of bottleneck channels is always half the number of output channels. Here, these numbers are adjusted for all Residual Modules to better align the quantity of available channels with the actually used channels.

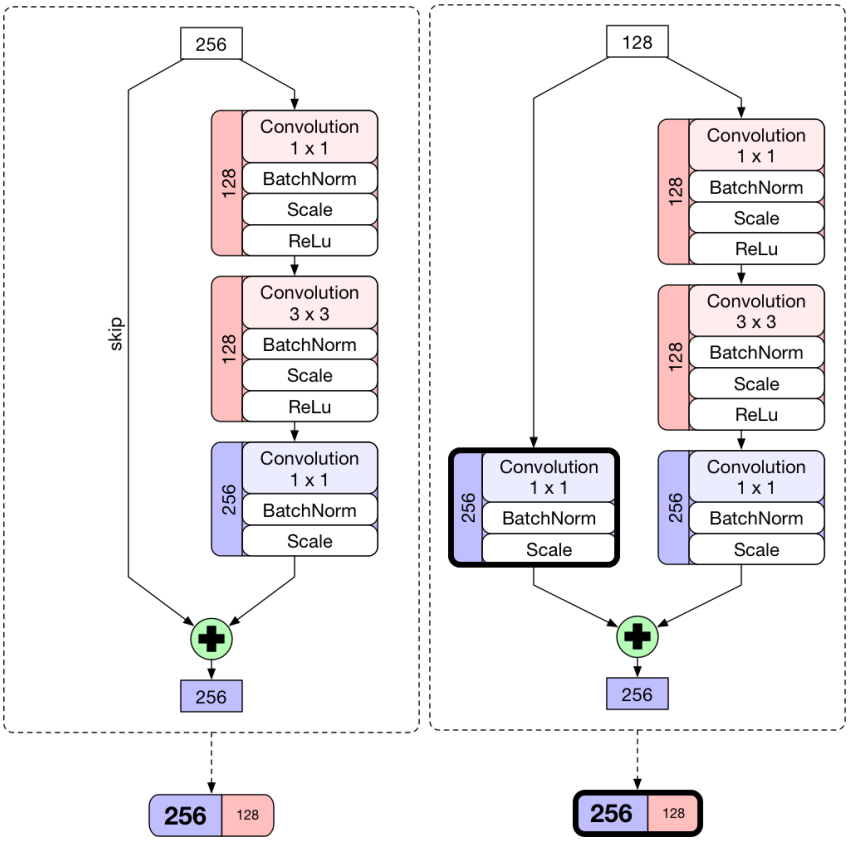


Figure 7.4: The two variations of the Residual Modules that are used throughout the network

*The used residual Module is a Caffe implementation of the one described by [*Newell *et al., 2016] (see section 2.2.2 and figure 2.4).*

left: *The number of input channels (256) matches the number of output channels. Therefor, the original input and the output of the last Convolution can be added directly.*

right: *The number of input channels (128) is different from the number of output channels (256). Therefor, an additional Convolution is needed to match the number of channels before adding.*

bottom: *This short notation is used hereafter to symbolize a Residual Module. The first number highlighted in blue denotes the output channel count of the last Convolution(s) and with that of the whole Residual Module. The second number highlighted in red stands for to the bottleneck which corresponds to the output channel count of the first two Convolutions. A thick border indicates an additional Convolution to match the channel count before adding.*

By analyzing the channel usage with the help of the LIT (see section 6.2.2) for all the Convolutions in all Residual Modules, unused channels could be identified. The minimum number of output channels for every Residual Module can then be determined by backtracing channel usage, going from the CNNs output to the input. Similarly, the minimum number of bottleneck channels within a Residual Module can be determined by backtracing its internal channel usage, going from its output to the input.
By removing about 90% of the channels that were labeled as unused by LIT, the optimized architecture shown in figure 7.5 was created. As examined in section 6.2.4, this increased the framerate by a factor of three while having no influence on the accuracy and detection rate.

7.4 Post Processing for 2D and 3D Pose Estimates

The post processing produces pose estimations from the CNNs heatmaps in 2D and 3D. To simplify this process for the outlined use case, it is assumed that at most one person is present at any time. Therefore, it is sufficient to produce exactly one estimate per joint.
This is done in 2D first by finding the highest peak in the heatmap with the algorithm shown in 7.6.
The second step done in post processing is to create 3D estimates from the 2D estimates and the original input data. This means to extend every 2D joint location by a matching depth value. The algorithm shown in 7.7 is used to extract this information from the original depth image.
Eventually, a skeleton is created based on these 3D joint locations. To place this skeleton as close as possible to the actual skelton, a static depth offset for each joint was calculated based on the training data. It corresponds to the difference between the depth image distance (distance to skin) and the actual distance to the joint "within" the human.
Finally, the resulting skeleton is transformed into world coordinates by using the

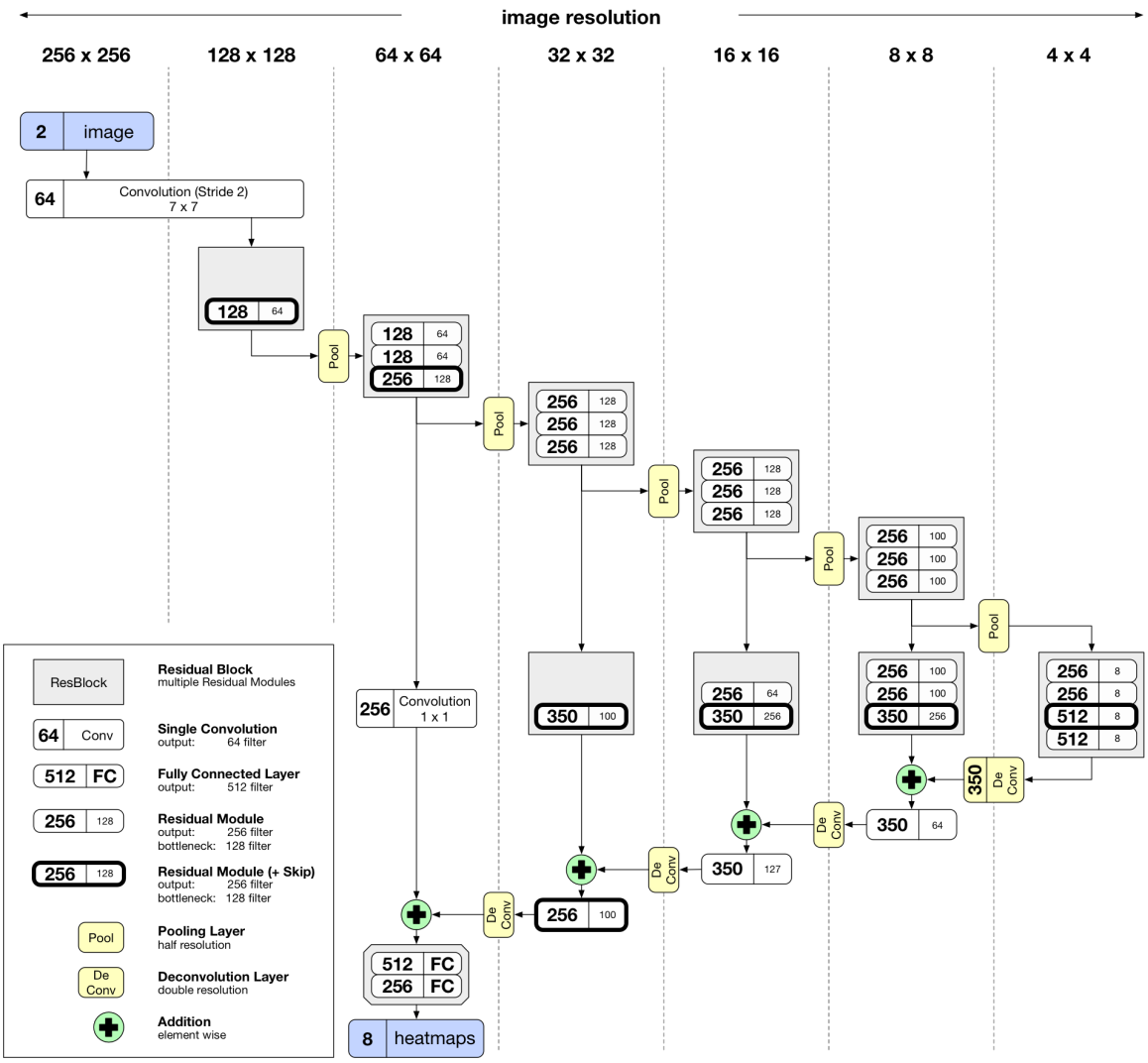


Figure 7.5: Detailed overview of the improved, final architecture
About 10,000 weights could be removed. Compared to the original architecture, this extends more than 55% of all weights.

known camera location and characteristics (intrinsic and extrinsic) for a better visualization.

In the next chapter, the performance of the described final system will be evaluated.

Input

$H_j \in \mathbb{R}^{X \times Y \times 1}; H_{jxy} \in [0, 255]$ // *heatmap image at resolution* $X \times Y$ *px for joint* j

Initialization

$T_a \leftarrow 75$; // *absolute threshold (minimum activation for a joint to be present)*

$T_r \leftarrow 0.75$; // *relative threshold (percentage of activation contributing to center)*

noCenter $\leftarrow$ [-1,-1] ; // *return value if no center was found*

$P_j \leftarrow$ [] ; // *point containing the location of the joint in image coordinates*

Algorithm

$H_j =$ `clip_below`(T_a, H_J); // *remove activations* $< T_a$

`if get_maximum_value`$(H_j) == 0$; // *no activation* $> T_a$ *is present*

  $P_j = noCenter$; // *no joint detected*

  `break`; // *return noCenter*

$H_j =$ `normalize`(H_j); // *make* $H_{jxy} \in [0, 1]$

$H_j =$ `clip_below`(T_r, H_j); // *remove activations* $< T_r$

`if count_nonzero_values`$(H_j) == 0$; // *no activation* $> T_r$ *is present*

  $P_j = noCenter$; // *no joint detected*

  `break`; // *return noCenter*

$P_j =$ `get_center_of_mass`(H_j); // *center of remaining values* $> T_r$

Return

$P_j = [x_j, y_j]; x_j \in [0, X]; y_j \in [0, Y]$ // *location of joint in image coordinates*

Figure 7.6: Algorithm to deduce the 2D joint location from a heatmap

To detect if a joint is occluded or not present, only activations $> T_a$ *are considered. Then, only the strongest relative activations above* T_r *are used for a center of mass calculation. Besides detecting if a joint is present at all, both thresholds also improve the accuracy by suppressing noise.*

Input

$I_D \in \mathbb{R}^{X \times Y \times 1}$ // *depth image at resolution* $X \times Y$ *px*

$S_{2D} = [(x_1, y_1), \ldots, (x_N, y_N)]$ // *2D skeleton with* N *joints*

Initialization

$O \leftarrow$ [0.098, 0.086, 0.030,0.057, 0.063, 0.041,0.045, 0.077] ; // *static offset for each joint*

$S_{3D} \leftarrow$ [] ; // *empty 3D skeleton*

Algorithm

$I_D =$ `erode_image`$(D, size = (3, 3))$; // *reduce noise*

$I_D =$ `dilate_image`$(D, size = (5, 5))$; // *thicken human silhouette*

`for` j `in count_joints`(S_{2D}) `do`; // *for each joint of the skeleton*

  $P_j = S_{2D}[j] = (x_j, y_j)$; // *2D point of joint in image coordinates*

  $patch =$ `get_region_around`(P_j, I_D); // *depth in circle around joint*

  $mean =$ `get_mean_depth`$(patch)$; // *mean depth in circle around joint*

  $z_j = mean + O[j]$; // *add specific static offset for joint*

  $S_{3D}[j] = (x_j, y_j, z_j)$; // *add 3D joint to skeleton*

Return

$S_{3D} = [(x_1, y_1, z_j), \ldots, (x_N, y_N, z_N)]$ // *3D skeleton in image coordinates and depth in* m

Figure 7.7: Algorithm create the 3D skeleton by lookup

The original depth image is eroded and dilated to fill in gaps introduced by noise and widen the human silhouette for a more robust result. In a circle around the 2D joint location, the average distance to the camera is calculated. Finally, a joint specific static offset is added to the result. This puts the skeleton "inside" the silhouette rater than on the outline.

Chapter 8

Experiments

In this chapter, the capabilities of the final design described in section 7 will be examined closely in three parts. First, the evaluation dataset is used for a quantitative assessment. Next, the performance will be compared against two state of the art approaches. Finally, the limitations of the presented system are examined qualitatively. All graphics in this chapter have a range of $p \in [0, 0.5]$. An upper limit of $p = 0.5$ is commonly used as a metric and therefore allows for better comparison with the state of the art. All tests were conducted with people that did not appear in the training data.

8.1 Evaluation

For evaluation of the final system, the CEval dataset was used. As described in section 5.5, it also contains a mirrored version of each frame to compensate for imbalances in the distribution of left and right body joints.
In order to allow for a good evaluation, the following three aspects need to be considered:

- The detection task gets harder along the kinematic chain (see figure 1.2). Because of that, the hand's detection rate is a better quality indicator than the head's detection rate.

- Poses can be of varying difficulty. More difficult poses are a better quality indicator than easy ones. For example, the number of occluded joints increases the difficulty of a pose. Additionally, poses which greatly differ from the training data can also be labeled as difficult. They allow an evaluation of the ability to generalize.

- The precision is correlated with the distance between person and camera. The higher a person stands away from the sensor, the higher is the error which results from a deviation of one pixel.

A breakdown of the average necessary computation time for a single frame is provided in figure 8.1. While the 2D estimation takes a total of 22 ms, the depth lookup increases the duration for a 3D estimate to 57 ms. The latter is only implemented as a proof of concept and requires further optimization to reach higher speeds.

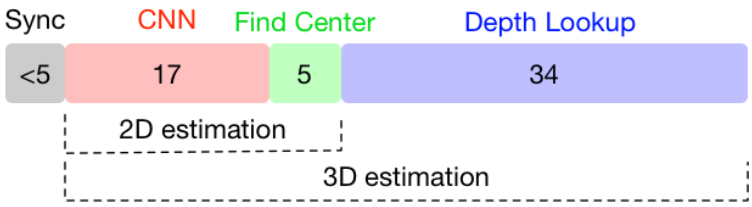


Figure 8.1: Breakdown of the average computation time in ms for a single frame *Less than five seconds are necessary for image synchronization and normalization. Because this is mainly due to the specific setup, it will not be considered for comparison. The CNN achieved 17 ms using caffe 1.0 [caf, 2017] on a single GeForce TitanX graphics card. Both the 2D (find Center) and 3D (Depth Lookup) postprocessing were implemented as ROS modules using Python.*

The overall performance for the CEval datasets is shown in figure 8.2. Since the dataset provides three different individual pose categories, the performance for each category will be described in the next subsections separately.

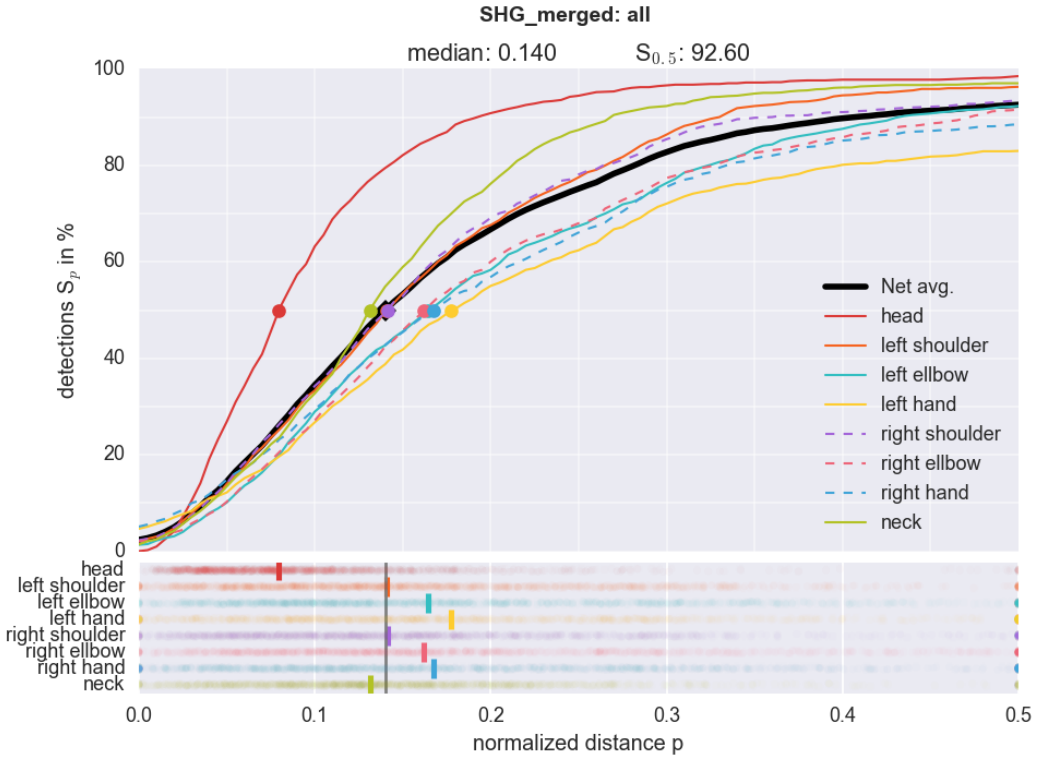


Figure 8.2: Performance across the entire CEval dataset

Overall, a successful detection was possible over 90% *of the time. The head can be detected very robust and precise, but the performance decreases along the kinematic chain. Conclusively, the hands proved to be the most difficult to detect with a failure rate of up to* 20%.

Basic Poses

All poses in this category were performed at varying distances to the camera and include common idle poses as well as especially difficult ones. The detection rates are shown in figure 8.3. Because of the high number of failed detections and the broad range of error distances, the performance on the "basic" category is the worst out of all three. The failed detections ($\approx$ 20%) were mostly caused by bending the upper body sideways, which was not present in the training data. These poses disrupted the detection for all joints (see section 8.3). However, bending the body forwards and backwards less than 90° could be detected correctly, even though is was not present in

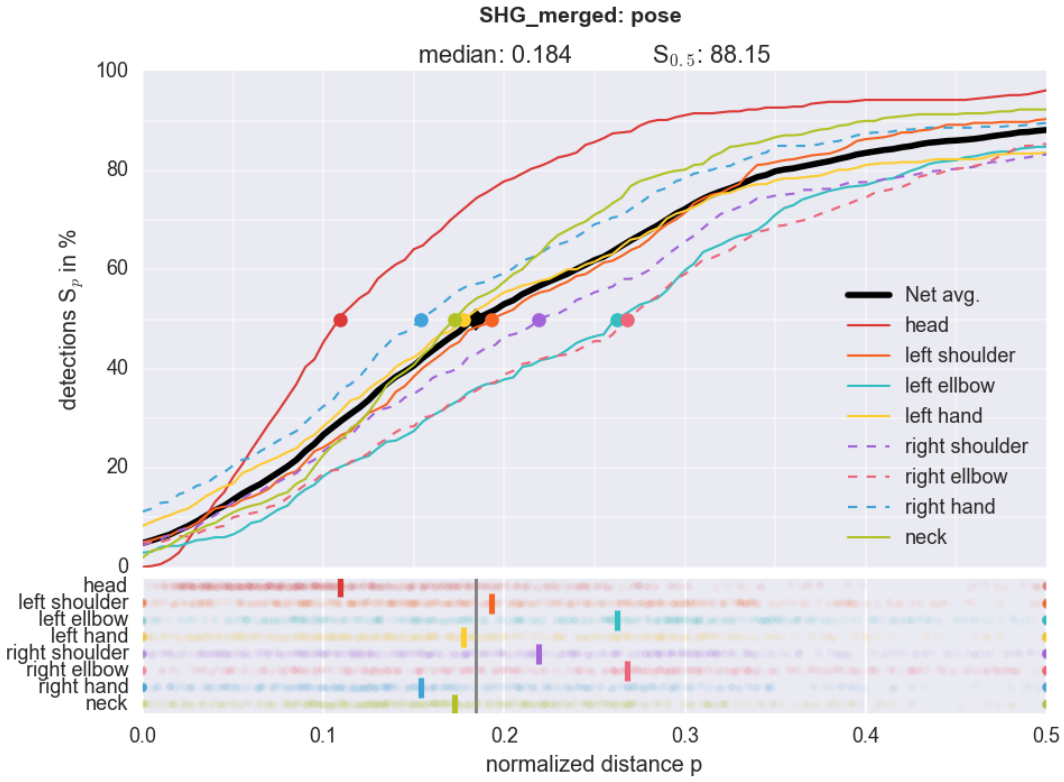


Figure 8.3: Performance on the "basic" category of the CEval dataset
All joints get detected reasonably well for basic poses. The error distribution is equally spread out for all joints except for the head, resulting in almost linear curves. There is no distinct average error for these joints, but a decline in precision can be noticed along the kinematic chain.

the training data as well.

The high diversity of error distances might be due to the high diversity of poses and positions. Some of the poses were performed close to the table while others were performed as far away from the camera as possible. Because the detection error is normalized to the head size, an error of a few pixels results in a higher detection error for poses further away.

In conclusion, poses in varying distances result in varying errors. Basic poses are generally detected well except for sideways bending, which cannot be detected.

Body Touching

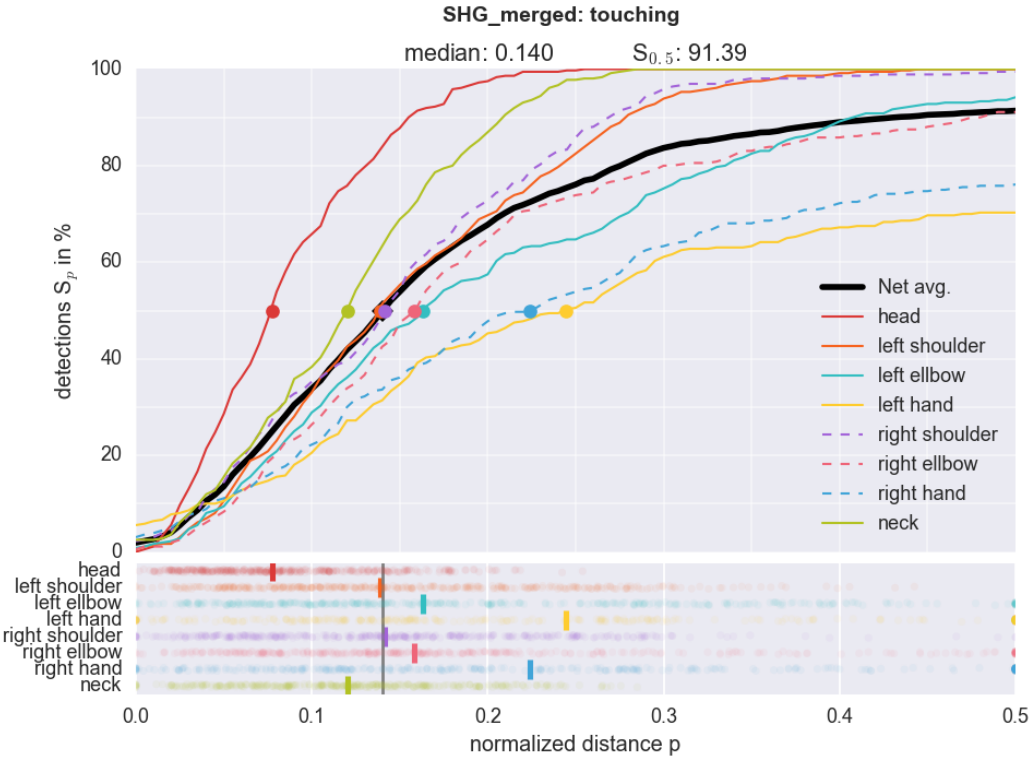


Figure 8.4: Performance on the "body touching" category of the CEval dataset
For body touching, a big spread in detection rates across the joints can be noticed, with decreasing precision along the kinematic chain. While the head detection succeeded in all frames with an median error of only 0.075*, the hand detection failed in roughly 25% of all cases with an median error almost three times that high.*

In this category, the hands are either at the hips, behind the back, crossed in front of the chest or behind the head. The resulting detection rates are displayed in figure 8.4. Even though the hands were mostly not visible, they were still estimated $\approx 70\%$ of the time with a median accuracy of about 0.25. Even though this category is expected to be more difficult than the previous one, the overall average error distance of 0.14 is lower. This might be due to two reasons. First, there were less poses in a great distance to the camera. Therefore, the error distances are smaller and more uniform. Secondly, there were no poses that caused the detection to fail completely

(like sideways bending). This becomes apparent in the maximum detection rate of $\approx 100\%$ for the head, neck and even shoulders.
In conclusion, the head, neck and shoulders detection is robust even if the hand's position cannot be estimated. In difficult situations where the hands are close to the body or hidden entirely, a hand detection is still possible in most cases with acceptable precision.

Work

The "work" category resembles the use case described in section 1.2.2. The poses in this category were performed in front of the table (close to the camera) and include manipulating and moving some objects with both hands. As can be seen in figure 8.5, the neck, shoulders and elbows get detected almost equally well without failures and a median precision of ≈ 0.11. This behavior suggests that the system is facing nearly ideal conditions and performing at its highest possible precision. Even the hands have a failure rate of less than 10% with median precision of ≈ 0.16. The lower precision originates from the object manipulation. Holding an object or placing the hands on the table results in a slight offset in the hands detection.
The "work" category also includes poses with crossed hands which were all detected correctly.
Summing up, the performance for the work category is the best of all three. This might be due to three reasons:

- The work category is the most prominent in the training data because it corresponds to the use case.
- The person is the closest to camera, therefore detection derivations of a few pixels have less impact.
- The category includes the fewest occlusions and unseen poses.

In conclusion, poses in front of the table get detected $\approx 98\%$ of the time with a median error of 0.113. Object manipulation only introduces a small error to the hand estimation, but does not result in any failures.

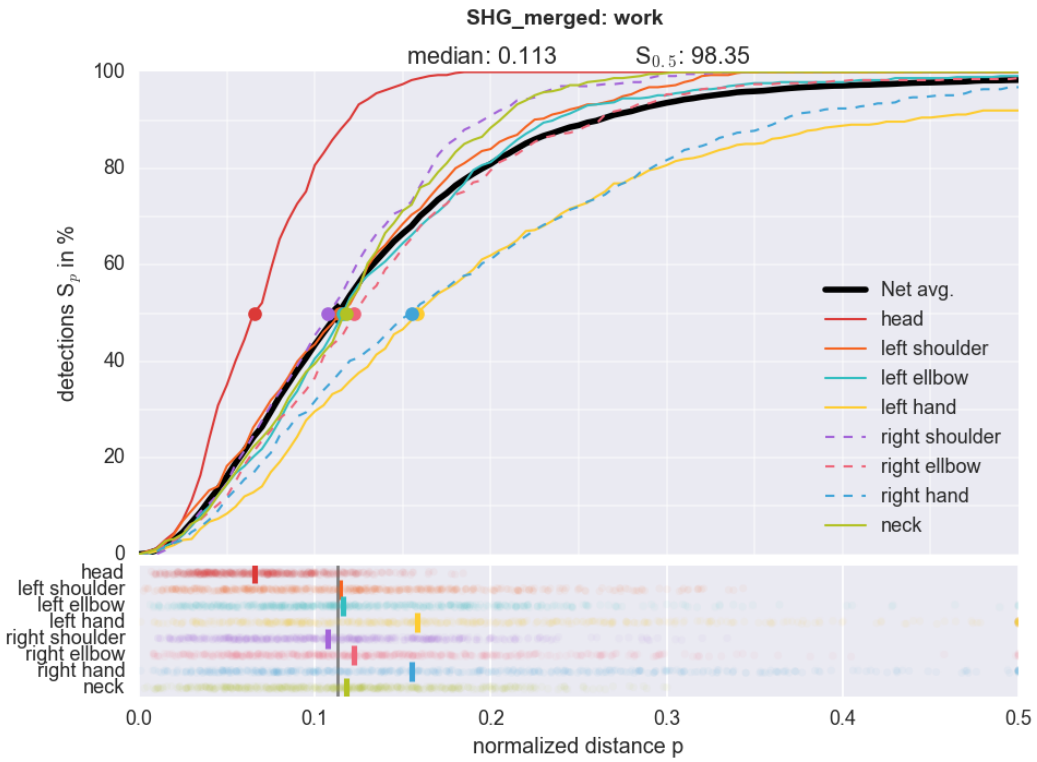


Figure 8.5: Performance on the "work" category of the CEval dataset
The spread in precision across the joints is quite low. The detection rates for the neck, shoulders and elbows are almost identical, only the head and hands diverged noticeably. The maximum detection rate for all joints is almost 100%, even the hands are above 90%.

Conclusion

The presented system reached all objectives outlined in section 1.2.3:

- **Joint detection:** The 2D detection of all visible upper body joints is possible with an overall detection rate of 92%. When sticking to the use case, a detection rate of 98% was achieved. With an average detection rate of 80%, 3D pose estimation is also possible.

- **Real time:** While the CNN is able to almost double the target framerate of 30 fps, 2D pose estimations can be produced in real time with 22 ms per frame.

With an average duration of 57 ms per frame an estimation in 3D is also possible.

- **Accuracy:** The median error for joint estimation is about 2.8 cm overall and about 2.3 cm for the use case.

- **Versatility:** The pose estimation also performs well outside the use case constraints. This will be examined more closely in section 8.3.

8.2 Comparison to State of the Art Approaches

In order to better interpret the described results, the performance of the final design was compared against two reference systems. For this, an implementation of the Convolutional Pose Machine (CPM) and OpenPose was chosen and evaluated on a newly created dataset.
The CPM was implemented like described in [gitb]. In contrast to the **CPM vanilla** architecture described in section 6.1.2, this one used six stages and processed three different resolutions by three successive passes. Further, the available pretrained models from [gitb] were used. This design needed a total of 1.28 seconds (including post processing) on average to process a single RGB frame, running on a single GeForce Titan X graphics card.
For OpenPose, the implementation of [Ope, 2017] was used. It processed the IR images with an average framerate of 30.59 fps, resulting in an average processing time of 32.7 ms per frame. In order to achieve these results, it needed to run on four cores of a GeForce GTX 1080 graphics card.

As already mentioned, the CEval dataset was not used for this comparison and the recording of a new dataset was necessary. This was mainly because of two reasons:

- The used CPM only works with RGB images, which are not available in any of the recorded datasets. Therefore a new dataset containing RGB images was necessary.

- All three systems were trained on different datasets with distinct label policies

(for example labeling the wrist instead of the palm of the hand). For a fair comparison, an independent labeling policy must be used.

The newly recorded dataset includes RGB, IR and depth images, along with 127 frames labeled by hand. However, it was found that the difference in head labeling across the three contestants was to big. Specifically, the CPM detected the very top of the head, the SHG final detected the center of the head and OpenPose the nose. For this reason, the head label was excluded from the evaluation. The difference for the remaining joint labels was up to 5 cm and therefore acceptable.
The performed poses are mainly within the work category (see section A.3.2) and feature a lot of occlusions, manipulations of larger objects, movements with crossing hands and also some poses far away.

A comparison of the results on this dataset is shown in figure 8.6. It shows that the presented system (SHG final) has a higher detection rate than the CPM, but cannot match up to the precision of OpenPose. A more detailed qualitative analysis is provided below.

Convolutional Pose Machine (CPM)

As mentioned above, the CPM performed the poorest. The main problem seems to be that the detection failed completely about 30% of the time. In these cases, not even a single joint could be estimated. Apparently, this happens when some of the upper body joints are occluded, for example while standing sideways. Possible reasons for this might be the used training data and architecture features. The CPM was trained on full body poses which also contain joints for lower body. When standing in front of a table the lower body is already occluded. Additionally occluding upper body joints seems to be too much occlusion ($> 50\%$) for a proper pose estimation. Furthermore, one feature of CPM is exploiting the relations between joints for detections. If these relations are missing because of occlusions, a reliable estimation might not be possible anymore. Maybe training on a pure upper body dataset like presented in section 5.5 might enable to tackle these problems.

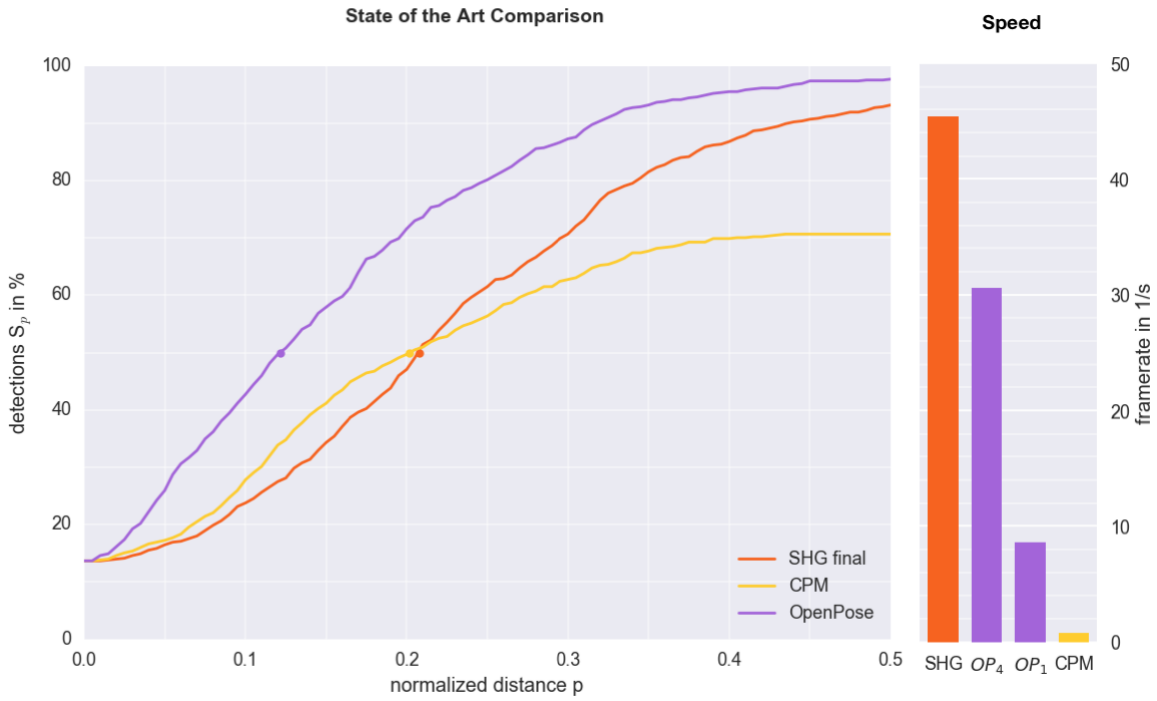


Figure 8.6: Comparsion of the presented system against two state of the art approaches

Because often times the detection failed completely with a resulting maximum detection rate of $S_{0.5} = 70.65$*, the CPM performed the worst. However, if the detection was successful, its accuracy is higher than the SHG final. Combining both of these effects, the median detection error of the CPM and SHG final is almost identical at* ≈ 0.2.

The performance of OpenPose, on the other hand, is better with an median detection error of 0.12 *and a maximum detection rate of* $S_{0.5} = 97.67$*. Although the SHG final's maximum detection rate is only* 5% *worse with* $S_{0.5} = 93.17$*, its median detection error is about* 73% *higher.*

In the speed comparison on the right side, OP_4 *and* OP_1 *correspond to OpenPose being run on four cores and a single core respectively.*

Nevertheless, it takes about 1.28 seconds for the CPM to process a single frame, making it almost two orders of magnitudes slower than the presented system. Even if it would be possible to reach better detection rates, it would still not be the favorable approach

for the intended use case.

OpenPose

OpenPose could reach the highest maximum detection rate but also, more importantly, the highest precision. In order to derive opportunities for future improvements of the presented system, possible causes for the differences in performance will be considered. Like for the CPM, the main differences are concerning the training data and architecture details. First of all, OpenPose was trained on a much more extensive data set. Specifically, the CMU Panoptic Dataset [CMU, 2017] [Joo et al., 2016] was used which was recorded in a dome with 480 VGA cameras, 31 HD cameras and 10 Kinect V2 sensors. In this setup, multiple actors performed over 60 sequences, resulting in over 1.5 million annotated 3D skeltons. Because these skeletons can be used in combination with each camera, the quantity of possible training images exceeds the dataset presented in section 5.5 by several orders of magnitude. Considering the high number of cameras and the professional setup, the annotations can be assumed to be superior in quality as well. The difference in precision between the presented system and OpenPose is very likely correlated to the described differences in the datasets.
As of now, the CMU dataset does not contain depth images and can therefore not be used for this thesis. Nevertheless, a dataset with a higher quantity of more accurate annotations might be able to improve the precision of the presented system.
Besides the difference in training data, [Cao et al., 2017] describe the use of part affinity fields for disambiguation in multi-person detection. This approach might additionally be able to improve the results of the presented system or even extend it to multi-person detection.
Utilizing four GPU cores, OpenPose was able to process a single frame in 32.7 ms. The other two contestants, however, only used a single core. When running OpenPose on a single core as well, the computation time for a single frame increases to 116 ms, making it five times as high compared to the presented system (see figure 8.6).

In conclusion, the system presented in this thesis performs well compared to state

of the art approaches while needing significantly less time to process a single frame. Further research is needed to determine if different training data might be able to overcome the described deficits in precision.

8.3 Limitations

In the previous section, the performance within use case conditions was examined. In order to investigate the capabilities of the presented system more closely, deliberate violations of certain use case constraints will be examined in this section.

8.3.1 Appearance of a Person

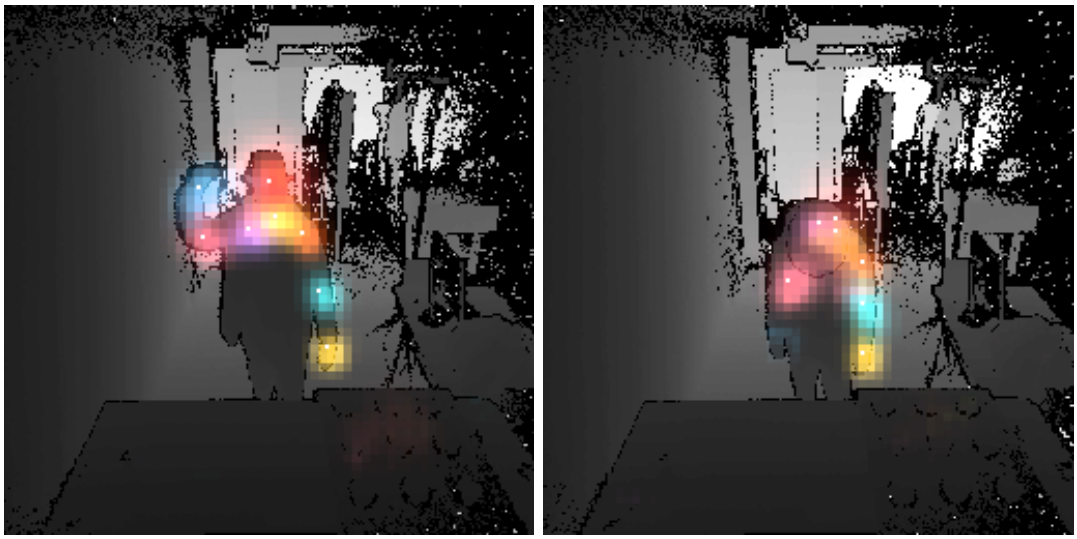


Figure 8.7: Altering the body shape and appearance with clothing
*The appearance and shape was altered by wearing a big jacket, ski mask and a hat. As shown in the **left** image, this seems to have no influence on the detection results. Solely when looking down (**right**) problems occur due to occlusions introduced by the hat.*

The focus of this subsection is on the appearance of the person, both visually (regarding IR images) and in shape (regarding depth images). First, the appearance

was modified by different types of clothing. Altering the visual appearance of the face with a ski mask, beanie or a hood showed to have no effect. Only when changing the head's shape by wearing a straw hat, detection errors occurred when the person was looking straight down so that the head is fully occluded by the hat (see figure 8.7). Different clothing (t-shirts, sweaters, jackets) seem to cause no problem in general. Nevertheless, some difficulties in arm detection could be noted when the clothing was very uniform in color and structure (like an all white lab coat or an all black jacket). Especially when the arms are close to the body and therefore difficult to detect in the depth image, uniform clothing decreases the detection quality slightly. This might be because edges become less obvious with this kind of clothing.
Since all persons in the training data were male with short hair, persons with different features were tested. Specifically, the gender, hair length, body size, facial hair and glasses were found to have no impact on the performance at all. The detection worked equally good for all these persons.

In conclusion, the approach is robust against almost all aspects of the person's appearance. Only minor problems were found when the clothing is very uniform in color and structure or occluding body parts. Including such clothing in the training data could possibly fix these problems.

8.3.2 Pose Limitations

In this subsection, the pose constraints of section 1.2.2 were intentionally violated in an effort to find poses that systematically cause failures.
Since only the upper body is estimated, the pose of the feet was found to be completely irrelevant. Knee bending, kneeling or even sitting have no influence on the detection at all. Solely lying flat on the floor could not be detected. Regarding the upper body, only a high angle of bending was found to be of relevance. Like shown in figure 8.8, the detection fails if the whole upper body is bent more than $90°$ sideways or forward. Additionally, crossing the hands when further away from the desk could not be detected, switching left and right hands and elbows. Because this kind of pose is

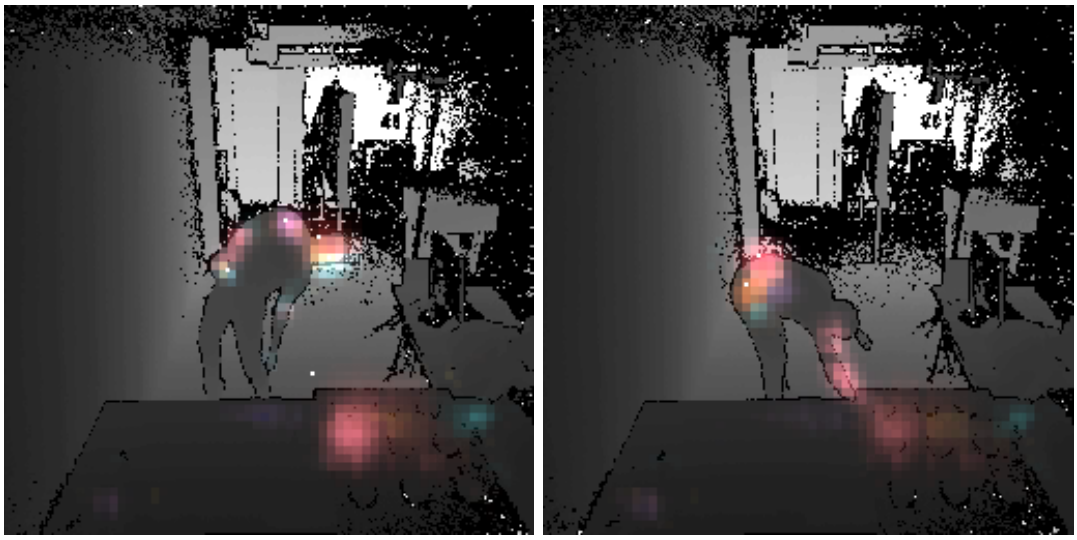


Figure 8.8: Failed detections due to extensive body bending
*Bending the upper body heavily sideways (**left**) or forward (**right**) causes the detection to fail.*

successfully detected when close to the table, this is assumed to be correlated to the training data. Specifically, this crossing pose only appeared in the training data when in front of the table. Furthermore, having the hands touching the head or body is sometimes harder to detect correctly, but this is more due to occlusions (see section 8.3.4). Besides that, no other poses were found to systematically disturb the detection.

In conclusion, only a few uncommon poses could be found to systematically disturb the detection. All of them can be related to the training data. Adding examples of these poses to the training set will likely enable a correct detection in these cases.

8.3.3 Detection of Multiple People

As defined by the use case, only one person ever appears in the training data simultaneously. Nevertheless, it might be beneficial to investigate the detection capabilities for multiple people. For this, the postprocessing module was deactivated because is not designed for this task. The bare CNN was tested with one, two, three and four

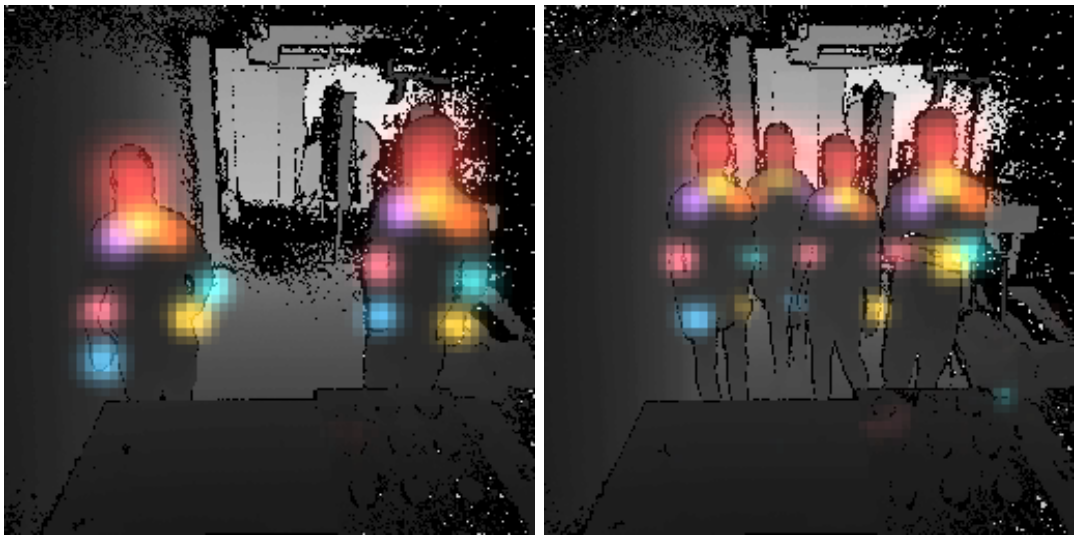


Figure 8.9: Detection of multiple people

*Multiple people are detected as good as a single person (**left**). Even when close to each other, the joints are assigned to correctly (**right**). Despite the person in the background (second from left), the arms and hands are correctly assigned to the persons in front.*

persons in the same picture. Because of the limited space within the detection area, it was not possible to fit more then four persons in a single frame with only minor occlusions. Like shown in figure 8.9, the detection works very good without added computation time as long as there is some space between the persons body parts. However, if they are really close to or occluding each other, not all upper body joints could be detected correctly anymore. Nevertheless, the detection of multiple people works very good in general.
To also enable the postprocessing module for multi-person detection, further adjustments are necessary. For example, a technique using part affinity fields could be implemented like in OpenPose.

In conclusion, the CNN is able to reliably handle multiple person detection, even though it was never trained for that. The postprocessing module, on the other hand, would require modifications.

8.3.4 Distance and Occlusion Limitations

First, the detection was tested at varying distances. It works equally good for all distances within the specified range, but decreases in precision with greater distance to the camera. As described before, this is because a deviation of one pixel results in a higher error if the person appears smaller. At a distance of about 2.4 m the detection quality reduces drastically (see figure 8.10, top left). The hands are no longer detected and the precision for the head, shoulders and neck estimation is not very high. At a distance of 2.8 m the detection stops completely. This might be mainly because poses at this distance never appeared in the training data. Additionally, a hand at this distance is only three to six pixels wide and therefore hard to detect. To increase the maximum detection range, training data at higher distances should be provided, either newly recorded or augmented. Also, the input and output resolutions could be increased.

Secondly, different kinds of occlusions were tested with varying results. When slowly lowering the body below the table, the hand are often estimated to be at the edge of the table. Moving outside the frame results in very poor and noisy detections, as soon as the head is not visible anymore (see figure 8.10 bottom left). In general, the detection almost always fails if the head is not visible. This might be because the head was always visible in the training data.
If the head is visible and some joints are occluded, the estimation will be attempted anyway. Like shown in figure 8.10 (top right), this even works for extreme occlusions, even though the precision is quite low. For less extreme occlusions like shown is figure 8.10 (bottom right), the estimation is quite close.
To further investigate the effects of occlusions, joints could be occluded systematically in post processing in the future.

In conclusion, the detection works well up to a range of 2.4 m. Minor occlusions are still estimated reasonably well, but occluding the head results in failure.

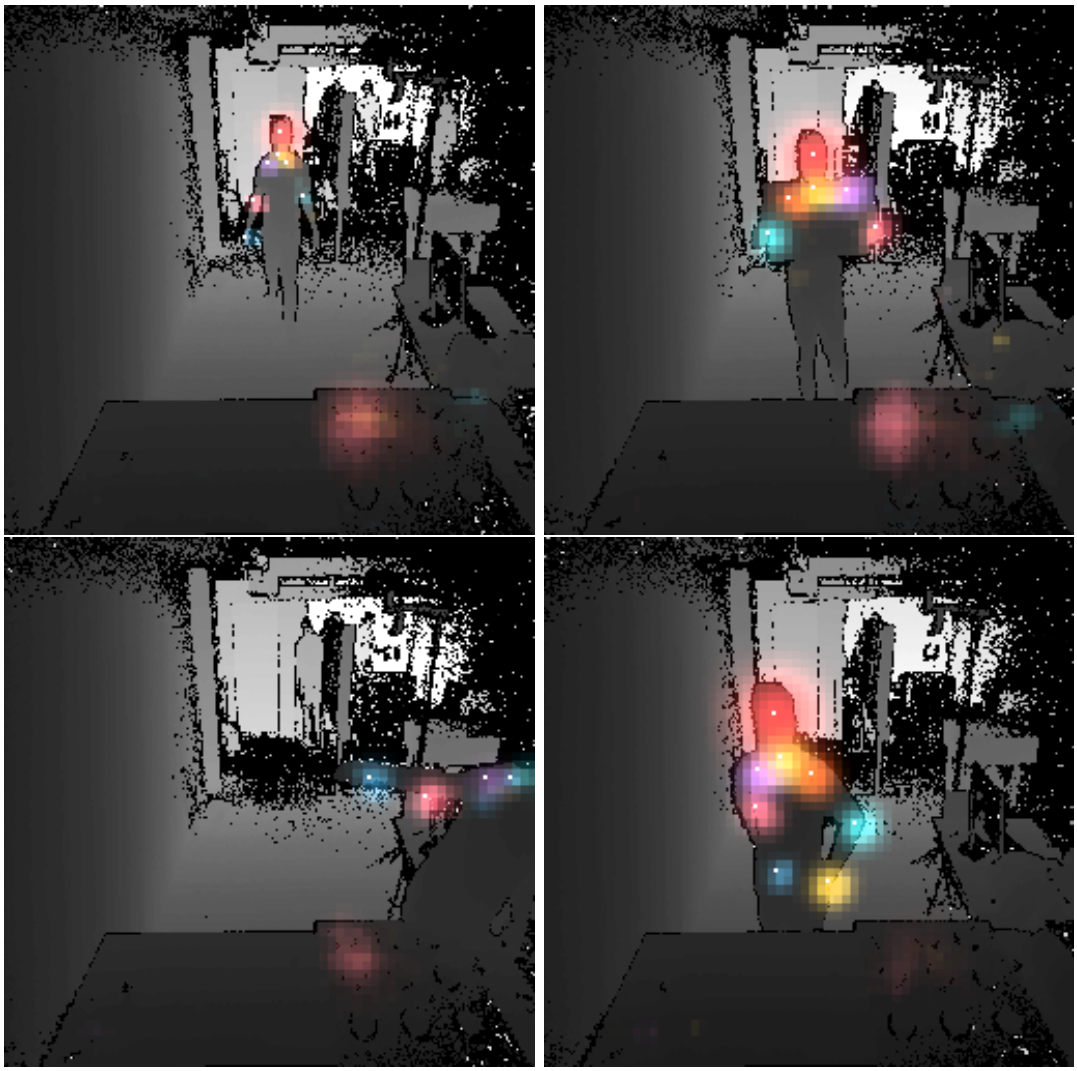


Figure 8.10: Examples for different occlusions

Top left: *Maximum distance to the camera at which a detection is possible.*

Top right: *Occluding the neck, shoulders and elbows by holding a board results in erroneous detection. The estimation is close considering the joints are not visible, but the left and right sides are switched.*

Bottom left: *Only the right arm is visible, resulting in a very noisy detection. The head is outside the right edge of the frame.*

Bottom right: *The right hand is behind the back and not visible. The detection is correct anyway.*

8.3.5 Background and Camera Position

To investigate the influence of the background and camera position, a Kinect V2 was installed in another room at a different angle. As shown in figure 8.11, the detection works equally good as long as the person is within the maximum detection distance of 2.5 m. Therefore, the approach is robust against changes of background and camera angle.

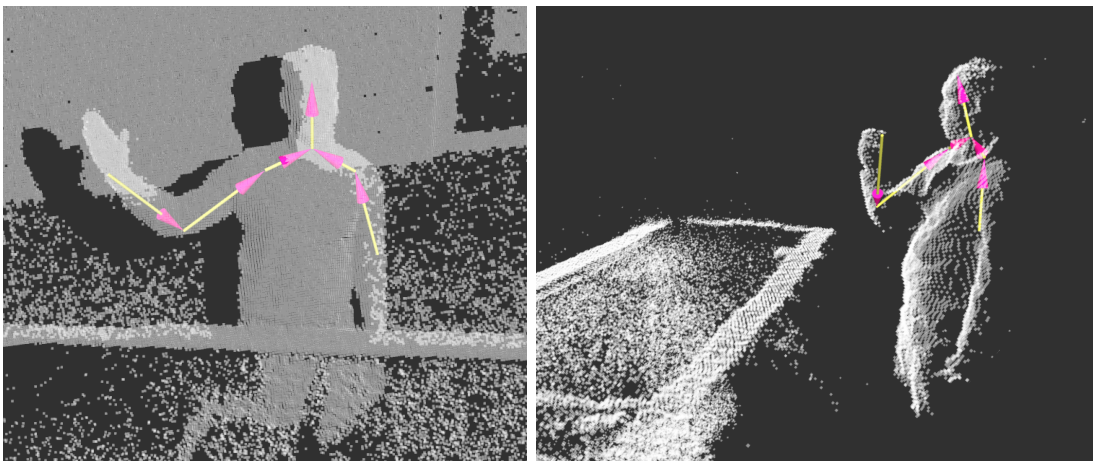


Figure 8.11: Using a different camera setup and background

The new setup utilized the pose estimations in combination with point clouds. Even though the used camera was installed a a different angle (higher up) in a different room, the estimation worked equally good.

Chapter 9

Summary and Perspective

9.1 Summary

This master thesis discussed the design of a system for real time body joint estimation within an industrial workbench scenario using a depth sensor. Its main contributions are the creation of a training dataset, the introduction of novel tool for weight optimization and a system for real time Human Pose Estimation based on depth data. First, different possibilities for depth data representation and processing were explored and compared. Being the best approach in both speed and precision, a Convolutional Neural Network (CNN) for direct processing of the sensor's image data was chosen. By further comparing three different state of the art CNN architectures, the Stacked Hourglass (SHG) design was found to be best suitable and used as a basis for further refinements. A novel visual tool for weight reduction was developed and used to optimize this initial architecture. In combination with weight merging, the average framerate could be increased by a factor of three. The final CNN was embedded in a pose estimation pipeline using the ROS framework. Besides a module for image capture and preprocessing, a post processing module was developed. This module produces joint estimations in both 2D and 3D.

Finally, the systems performance was examined closely and compared to two state of the art approaches. It was able to reach nearly state of the art performance with an

average detection rate of 92% and a median detection error of $0.14 \approx 2.8\,\text{cm}$. These results include difficult and unseen poses up to a range of 2.5 m. Regarding solely the use case, a higher performance was reached with a detection rate of 98% and a median detection error of $0.113 \approx 2.26\,\text{cm}$. It takes an average of 22 ms to produce a 2D estimate which surpasses the speed of state of the art systems. With the additional postprocessing module, a 3D estimation needs 56 ms. The trained CNN is also able to detect multiple people. Only extensive occlusions or head occlusion as well as a few unseen poses were found to disturb the detection.
In order to guide the development process, train the CNN and evaluate the results, a novel training dataset with 25,000 annotated samples was recorded. The preparation, recording, annotation, post processing and error correction for this dataset was described in detail. By additionally applying data augmentation, the size could be increased to 150,000 samples. Training on this dataset reduced the median detection error by 50%.

9.2 Perspective

Several options would allow to improve the presented system.

First of all, an end-to-end system for direct 3D estimation with a CNN might be possible. As of now, the seperate 3D prediction takes the longest to compute and introduces errors. For example, a design like presented in [Martinez et al.] or [Bulat and Tzimiropoulos, 2016] could be used to transform 2D estimates into 3D estimates. Alternatively, [Li et al., 2014] suggests that this should also be possible using a single network architecture.

In order to achieve higher framerates, the current Python implementation of both the pre- and post-processing modules could be exchanged.
To further improve the CNN's speed multiple passes of layer inspection and weight reduction can be employed. By extending the capabilities of the Layer inspection

Tool, this process could also be automated.

Regarding the training data, a more extensive dataset could be used or created. This should additionally include poses that were found to be difficult, performed at greater distances and feature more occlusions.

Appendix A

Additional Documents

A.1 Use Case Constraints

- **sensor constraints:**
  - The data of a single Time of Flight depth sensor must be used exclusively.
  - The characteristics of the depth sensor are assumed to be constant.
  - The input data arrives at a frequency of at least 30 Hz.
- **room constraints:**
  - The location of the detection area is fixed within the room.
  - The location and orientation of the depth sensor is fixed within the detection area
  - The lighting is fixed within the room.
  - There are no signal or light sources present that might interfere with the Time of Flight measurements
  - The background is mostly static (small changes like repositioning objects might occur).
- **detection area constraints:**
  - The detection area has a fixed size of 1.8×2.05 m

  - There is a workbench at a fixed position.
  - There is a desk (robot manipulation space) at a fixed position.
  - There is at most one person present in the detection area.
  - There might be no person in the detection area at all.
  - There is a keyboard on the workbench.
  - There are nine buttons on the workbench at a fixed location.
  - There may be the following objects somewhere on the workbench:
    * a cubic box of size $0.32 \times 0.32 \times 0.28\,\mathrm{m}$.
    * several big LEGO pieces roughly $0.16 \times 0.07 \times 0.07\,\mathrm{m}$ in size.

- **person constraints:**
  - The person is an adult with short hair.
  - The person's clothing is limited to long pants, T-shirts and sweatshirts
  - The person may wear glasses.

- **pose constraints:**
  - The person is standing in an upright position at all times, but is free to bend over.
  - The person must not bend their knees.
  - The upper body joints (head, shoulders, elbows, hands) are above knee level at all times.
  - Some upper body joints may be outside the detection area or occluded.
  - Some or all lower body joints (hips, knees, feet) might be outside the detection area or occluded.
  - Objects must be manipulated roughly at workbench height (above knee level and below head level) and in front of the body.
  - the person is free to:

* interact with the workbench (e.g. resting hands on it).
* interact with the keyboard and buttons on the workbench.
* manipulate the objects on the workbench with one or two hands.
* take any pose within the above described limitations.

A.2 Convolutional Pose Machine

As the first important CNN architecture for Human Pose Estimation the CPM will be presented in this section. It was introduced in [Wei et al., 2016] and since then utilized multiple approaches. One of the most recent and also popular usages of this architecture is OpenPose [Cao et al., 2017].
The CPM is able to predict 2D body joints in 2D images. Since its design is basically a CNN implementation of a Pose Machine [Ramakrishna et al., 2014], the underlying Pose Machine will be described first.

A.2.1 Original Pose Machine

On a high abstraction level, the main principle of the Pose Machine is to process multiple resolutions (levels) simultaneously and successively (stages) and therefore utilizing more complex, intra-join relations.
Like shown in figure A.1 (a), the basic building block is a multi-class predictor where each class corresponds to one of the joints. Its task is to predict the likelihood of of a patch belonging to each class. The prediction is based on features that are calculated from a specific region (patch) of the input image. Based on this per-patch prediction, one belief map per class is created that contains the likelihood for each patch to belong to this class. Multiple of these predictors are arranged in stages and levels like displayed in figure A.1 (b). The first stage computes features based on the original image, all subsequent stages compute features based on the output of previous stages instead. Therefore, each stage is refining the previous result by also considering inter-joint-relations.
For each stage, there are multiple levels which differ in the patch size used for the

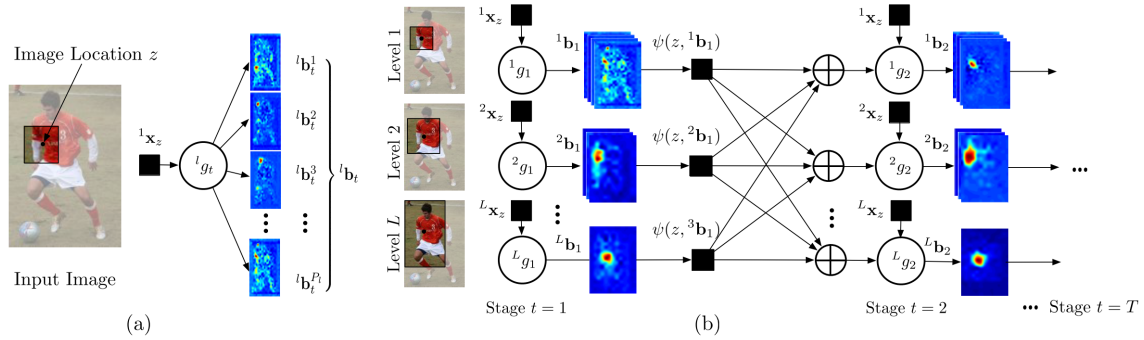


Figure A.1: Overview of the original Pose Machine

"(a) Multi-class prediction. A single multiclass predictor is trained for each level of the hierarchy to predict each image patch into one of $P_l + 1$ classes. By evaluating each patch in the image, we create a set of confidence maps ${}^l\boldsymbol{b}_t$.

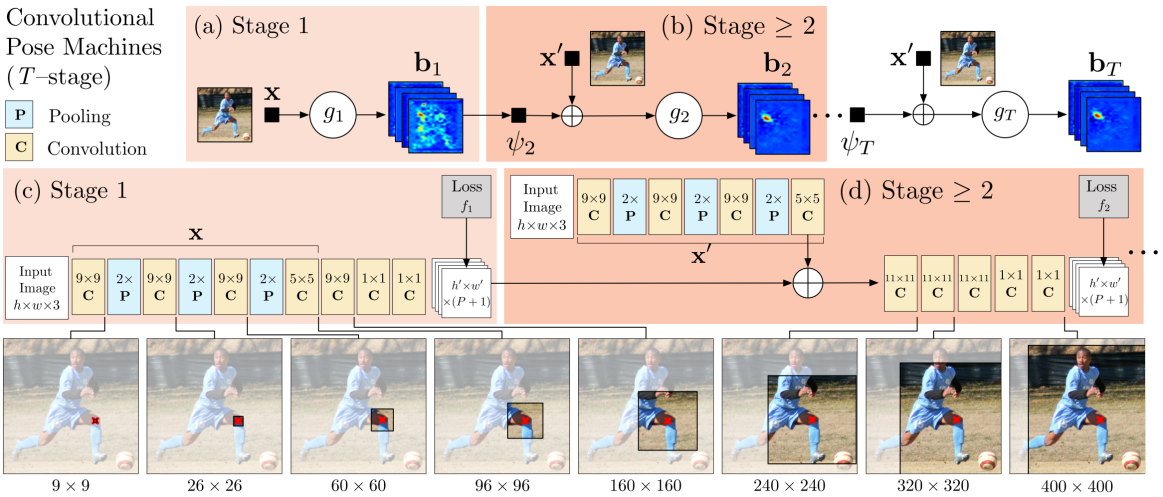
(b) Two stages of a pose inference machine. In each stage, a predictor is trained to predict the confidence of the output variables. The figure depicts the message passing in an infer- ence machine at test time. In the first stage, the predictors produce an estimate for the confidence of each part location based on features computed on the image patch. Predictors in subsequent stages, refine these confidences using additional information from the outputs of the previous stage via the context feature function ψ."

*source: [*Ramakrishna *et al., 2014]*

feature creation. This allows the pose machine to consider features on multiple scales and combine them for predictions in later stages.

A.2.2 CNN Implementation of the Pose Machine

While the features in [Ramakrishna et al., 2014] were hand crafted, [Wei et al., 2016] adopt the basic principle and implement it using a deep CNN. This allows to learn the features by training. Similarly to [Ramakrishna et al., 2014], their system architecture (see figure A.2) consist of multiple stages, each refining the previous result. The multiple levels (patch sizes) are realized by multiple successive convolutions which gradually increase the effective receptive field.



(e) Effective Receptive Field

Figure A.2: Architecture and receptive fields of the Convolutional Pose Machine (CPM)

"We show a convolutional architecture and receptive fields across layers for a CPM with any T stages. The pose machine [Ramakrishna et al., 2014] is shown in insets (a) and (b), and the corresponding convolutional networks are shown in insets (c) and (d). Insets (a) and (c) show the architecture that operates only on image evidence in the first stage. Insets (b) and (d) shows the architecture for subsequent stages, which operate both on image evidence as well as belief maps from preceding stages. The architectures in (b) and (d) are repeated for all subsequent stages (2 to T). The network is locally supervised after each stage using an intermediate loss layer that prevents vanishing gradients during training. Below in inset (e) we show the effective receptive field on an image (centered at left knee) of the architecture, where the large receptive field enables the model to capture long-range spatial dependencies such as those between head and knees. (Best viewed in color.)"

source: [Wei et al., 2016]

A.3 Training Data

A.3.1 Existing Options

As examined in the section 4.3, a dataset by [Arenknecht, 2016] is available, but not suitable for this thesis. In addition, several different data sets for the task of Human Pose Estimation are publicly available. By reviewing the literature, it was found that most approaches use one of the following datasets for training:

- The **Frames Labeled In Cinema (FLIC)** dataset contains 5003 images from Hollywood movies and was introduced by [Sapp and Taskar, 2013]. It was labeled using the crowdsourcing marketplace Amazon Mechanical Turk. For each image, five independent labels for the upper body parts were combined using their median for a robust annotation. Successful uses of this dataset include [Tompson et al., 2014], [Ramakrishna et al., 2014], [Toshev and Szegedy, 2013], [Rafi and Leibe, 2016] and [Newell et al., 2016].

- The **MPII Human Pose (MPII)** dataset contains about 25.000 images extracted from YoutTube videos and was introduced by [Andriluka et al., 2014]. It includes over 40.000 different people with annotated body joints performing over 410 human activities in total. Besides being a popular dataset for benchmarks (see [MPI] and section 2.4), several approaches use it for training, including [Rafi and Leibe, 2016],[Tekin et al., 2016], [Mehta et al., 2017], [Insafutdinov et al., 2016], [Elhayek et al., 2015], [Bulat and Tzimiropoulos, 2016], [Newell et al., 2016] and [Sun et al., 2017].

- The **Human3.6M** dataset introduced in [Ionescu et al., 2014] contains 11 professional actors performing 16 scenarios altogether in a studio setup. Four cameras were used to capture 3.6 million human poses while 3D joint positions and joint angles were provided by a high-speed motion capture system. The dataset also contains Time of Flight data (depth images) and was used, besides others, by [Tekin et al., 2016], [Mehta et al., 2017], [Li et al., 2014] and [Sun et al., 2017].

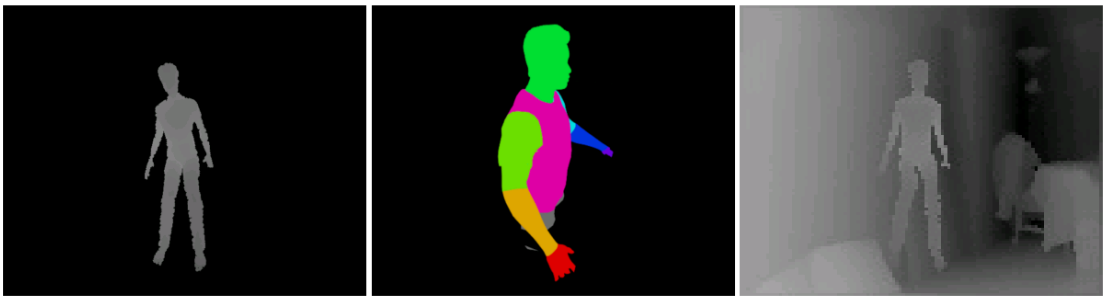
- The **Leeds Sports (LSP)** dataset was introduced by [Johnson and Everingham, 2010] and consists of 2.000 images from Flickr displaying mostly sports people. The images were cropped around the person and annotated with 14 joint locations. Usages of this dataset include [Toshev and Szegedy, 2013], [Tompson et al., 2014], [Rafi and Leibe, 2016], [Tekin et al., 2016], [Ramakrishna et al., 2014], [Mehta et al., 2017], [Toshev and Szegedy, 2013], [Insafutdinov et al., 2016] and [Bulat and Tzimiropoulos, 2016].

All of the aforementioned datasets provide RGB images as input, but only the Human3.6M dataset additionally includes depth data. This depth information is provided as Time of Flight measurements (depth images), but does not include IR images. However, the experiments presented in section 4.2 suggest that using IR images can significantly increase the performance.
Furthermore, most of the poses in the Human3.6M dataset are full body poses and don't include a lot of lower body occlusions or work poses (object manipulation). Many scenarios additionally contain multiple persons, which does not match the requirements.
In conclusion, none of the described datasets is suitable for the task because the input data format and the available poses are not sufficient. Therefore, alternatives need to be considered.

Facing similar problems, [Huang and Altamar] created a synthetic dataset of depth images. Their dataset creation pipeline utilizes a 3D human model with varying parameters that gets manipulated by motion capturing data like shown in figure A.3. Apart from that, [Shotton et al., 2013] have also created a synthetic dataset in a similar fashion. However, those datasets are not publicly available.
The creation of synthetic datasets would be possible, but is rather time consuming since there is no ready-to-use pipeline for it yet. Also, it might require some extensive effort until it matches the use case data close enough.
In conclusion, no sufficient dataset is available and synthetic dataset creation would exceed the scope of this thesis.

(a) Skeleton for Human Model (b) Retargeting to Mocap Data (c) Labeling Different Parts

Figure 6. Human Animation: (a) We create a skeleton for each human model. (b) We retarget the created human model with Motion Capture data. (c) We paint different body parts with different colors for segmentation task in the future.



(a) Depth Image (b) Per-Pixel Labeling (c) Background

Figure 7. Dataset Results: (a) We create kinect-simulated depth image. (b) We create per-pixel labeling of different body parts. (c) We blend depth images with background.

Figure A.3: Process for synthetic dataset creation

[Huang and Altamar] describe their process of creating a dataset of labeled depth images displaying humans of varying shape in front of varying backgrounds.

Image source: [Huang and Altamar]

A.3.2 Complete List of Motion Sequences

1. Basic Poses

   A: straight arms

      A1: hands motion: hanging → T-Pose → top

      A2: hands motion: hanging → front → top

      A3: hands motion: front → T-pose (different heights)

   B: bent arms

      B1: hands motion: hanging → T-Pose → top

      B2: hands motion: hanging → front → top

B3: hands motion: front → T-pose (different heights)

C: body touching

C1: hands motion: holding front → resting hip → holding back

C2: hands motion: crossed → both behind head

C3: head with single hand (l/r) at many different angles

C4: head with both hands touching head at many different angles

D: random movement

D1: reaching imaginary points in 3D with l/r/both hands (full upper body motion)

D2: completely random movement of full upper body

2. Work Poses

E: idle poses (l/r/both hands), different positions, both tables):

E1: hands resting on table

E2: hands supporting body on table

E3: typing on keyboard while slightly changing position

F: button interaction (l/r hand)

F1: pressing every button once at a time from different locations

G: box and LEGO interaction (both hands)

G1: walking around detection area with box in both hands

G2: picking up box from desks and putting it at random locations on desks

G3: getting LEGOs out of box, stacking them, moving the box to a different location, putting them in again

G4: moving the LEGOs around on the tables one at a time

3. Validation Poses

V: poses for performance validation

V1: basic: hanging –> T-pose, turning around, upper body bending

V2: work: moving LEGOs on table, including arms crossing

V3: body touching and hands hiding while walking around

A.3.3 Error Detection, Synchronization and Filtering

A Captury skeleton consits of 29 joints in 3D space (x,y and z coordinates) which resemble a human skeleton like shown in figure 5.1. First of all, the skeleton is transformed in such a way, that only 8 joints remain: head, neck, left shoulder, left elbow, left hand, right shoulder, right elbow, right hand. Additionally, the coordinates are converted to be relative to the viewpoint of the depth camera and normalized, so that they are independent from specific image sizes.
Since the Kinect runs with 30 Hz and Captury Live with 50 Hz, the data was not synchronized at recording time. As found out after recording, there are also some static delays between the images and skeletons which vary between recordings (displayed in figure A.4). Furthermore, the labels showed to be incorrect sometimes. Those errors need to be excluded and the data needs to be synchronized before training

Figure A.4: Timing offset between depth images and Captury skeletons
This image is a still frame during a motion where the hands move downwards quickly. Because of the timing offset, the hands of the Captury skeleton are slightly higher than in the depth image.

It was found that Captury had mainly two failure modes that were problematic for the intended data usage. Sometimes, the skeleton is slightly offset so that a body joint would be on the edge of a body part or even outside (see figure A.5, left). Sometimes,

the detection failed completely so that the skeleton was not even close to the actual body pose (see figure A.5, right). Especially at the edges of objects an persons, the depth image was very noisy.

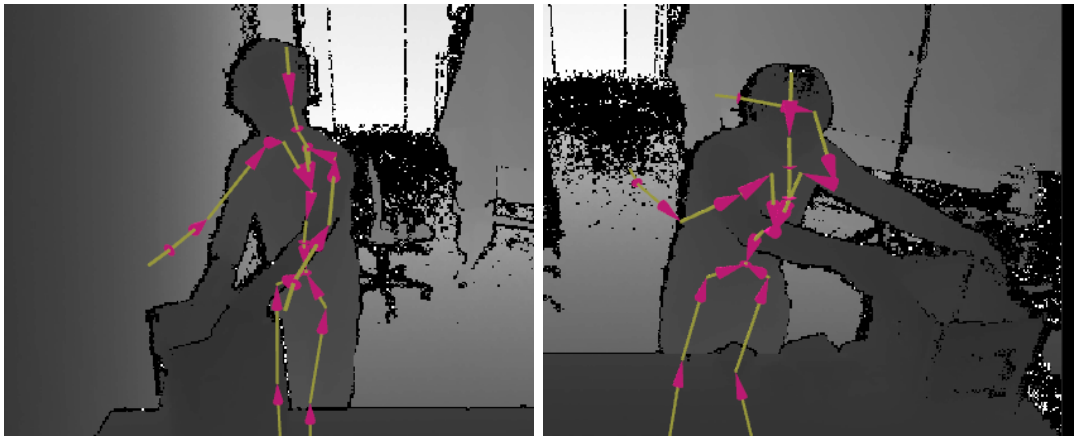


Figure A.5: Examples for Captury skeleton failures
Left: *One limb of the skeleton is offset too much so that it's not inside the actual limb anymore. This happens rarely but usually affects multiple frames.*
Right: *The detection failed entirely and the skeleton is mostly wrong. This happens very rarely and usually affects only 1-5 frames*

Such errors need to be filtered before the frames can be used for training. Since the quantity of frames is fairly high and small offsets might be hard to notice, an automated process was developed that assigned each joint into one of three categories:

- **-1**: joint label is incorrect and must be ignored
- **0**: joint label is occluded (not visible to camera)
- **1**: joint label is valid and can be used for training

Input

$S_{3D} = \{(x_1, y_1, z_1), \ldots, (x_N, y_N, z_N)\}$ // *skeleton with N joints*

$I_D \in \mathbb{R}^{H \times W \times 1}$ // *depth image at resolution* $H \times W px$

$CM \in \mathbb{R}^{3 \times 3}$ // *camera matrix for depth camera*

Initialization

$C_J \leftarrow \{\}$; // *used to store the correctness class for each joint*

Algorithm

`for` j `in` S_{3D} `do;` // *for each joint of the skeleton*

  $(x_j, y_j, z_j) =$ `get_img_coord(`j, CM`);` // *transform joint to image coordinates*

  $patch =$ `get_region_around(`$(x_j, y_j), I_D$`);` // *depth in circle around joint*

  $mean =$ `get_mean_depth(`$patch$`);` // *mean depth in circle around joint*

  $min =$ `get_minimum_depth(`$patch$`);` // *minimum depth in circle around joint*

  $max =$ `get_maximum_depth(`$patch$`);` // *maximum depth in circle around joint*

  `if` $mean - z_j > 0.1$`m or;` // *joint "in front" of person, closer to camera*

  `if` $max- > 3.5$`m or;` // *patch further away than possible –> on background*

  `if` $min- < 0.2$`m then;` // *patch closer than possible –> too much noise*

    $C_J(j) = -1$; // *joint is incorrect*

  `else if` $z_j - mean < 0.17$`m then;` // *joint "behind" person*

    $C_J(j) = 0$; // *joint is occluded*

  `else then;`

    $C_J(j) = 1$; // *joint is correct*

Return

$C_J = \{c_0, \ldots, c_N\}; c_{0 \ldots n} \in \{-1, 0, 1\}$ // *correctness for every joint*

Figure A.6: Algorithm used to automatically detect errors in the recorded label data

Depending on what the corresponding depth image I_D displays at the location (x_j, y_j) of a joint j, each joint is assigned to one of three classes: ***-1*** *for incorrect,* ***0*** *for occluded and* ***1*** *for correct. A joint is incorrect, if it is between the camera and the person, at the edge of a limb or not on the person at all . It is occluded, if something at the same location is closer to the camera than the joint. Otherwise, it is correct.*

A.3.4 Heatmap Generation

Input

$S_{3D} = [(x_1, y_1, z_1), \ldots, (x_N, y_N, z_N)]$ // *skeleton with N joints*

$R_H = [X, Y]$ // *desired resolution of the resulting heatmap*

$CM \in \mathbb{R}^{3\times3}$ // *camera matrix for depth camera*

Initialization

$H \leftarrow []$; // *used to store the heatmap for each joint*

$scale \leftarrow [0.95,0.5,0.5,0.45,0.5,0.5,0.45,0.5]$; // *how much each joint scales with distance*

Algorithm

`for` j `in count_joints`(S_{3D}) `do;` // *for each joint of the skeleton*

$H_j =$ `get_empty_image(`R_H`);` // *create empty heatmap*

$(x_j, y_j, z_j) =$ `get_img_coord(`$S_{3D}[j], R_H$`);` // *transform joint to image coords*

$\sigma = 2.5 \times scaling[j] \times (4 - z_j) \times \frac{R_{H_x}}{100}$ `;` // *calculate size of blob*

$H_j =$ `put_gaussian_at`(x_j, y_j, σ, H_j)`;` // *place gaussian blob at joint position*

$H[j] = H_j$ `;` // *store heatmap for joint*

Return

$H \in \mathbb{R}^{X\times Y\times8}$ // *8 heatmaps, one for every joint*

Figure A.7: Algorithm to create heatmaps from 3D skeletons

A gaussian blob is placed at the location of each joint in the image. The size of the blob depends on the type of body part and its distance to the camera.

A.4 Layer Inspection Tool: Mathematical Background for Visualization

From a mathematical perspective, each convolution layer basically applies a number of F filters with size $X \times Y$ px to an input image with C channels (for example, for an RGB-image $C = 3$). Further, each filter is a collection of C filter patches with size $X \times Y$ - one patch per channel. These filter patches are stored as a matrix $\mathbf{w}_{f,c} \in \mathbb{R}^{X \times Y}$ containing weights. To produce the output of a layer, each filter f applies its patches to the input on channel c by performing multiplicative operations. A filters output is calculated by summarizing the results of each patch. So if a channel has little significance for the result of a specific filter, the weights of its patch will be close to zero. If, on the other hand, a channel contains important information for the specific filter, the weights of its patch will have relatively high absolute values.
For weight usage optimization, it is of interest how much those patches influence the final output of a layer. Therefor, an activation measure $\mathcal{A}_{f,c}$ of a patch is introduced and defined as the sum of its absolute weights.

$$\mathcal{A}_{f,c} = ||\mathbf{w}_{f,c}||_1 = \sum_{x=1}^{X} \sum_{y=1}^{Y} |w_{f,c}^{x,y}| \tag{A.1}$$

Channel activation $\mathcal{A}_c$ and filter activation $\mathcal{A}_f$ can now be defined as the sum of activation across all channels and filters respectively.

$$\mathcal{A}_c = \sum_{f=1}^{F} \mathcal{A}_{f,c} \tag{A.2}$$

$$\mathcal{A}_f = \sum_{c=1}^{C} \mathcal{A}_{f,c} \tag{A.3}$$

Since the absolute values of the weights can differ greatly between layers, the activation is normalized. The normalized activation across all channels and filters respectively is called channel significance $\mathcal{M}_c \in [0, 1]$ and filter significance $\mathcal{M}_f \in [0, 1]$ (1 meaning

most significant).

$$\mathcal{M}_c = \frac{\mathcal{A}_c}{\max\limits_{c'}(\mathcal{A}_{c'})} \tag{A.4}$$

$$\mathcal{M}_f = \frac{\mathcal{A}_f}{\max\limits_{f'}(\mathcal{A}_{f'})} \tag{A.5}$$

The higher the significance of a channel or filter, the higher is its influence on the layers output. A channel or filter is labeled as unused, if its significance is below 0.05. This is the most piece of information for weight reduction, since it indicates how many weights filters can be removed from the layer without significantly decreasing its performance.

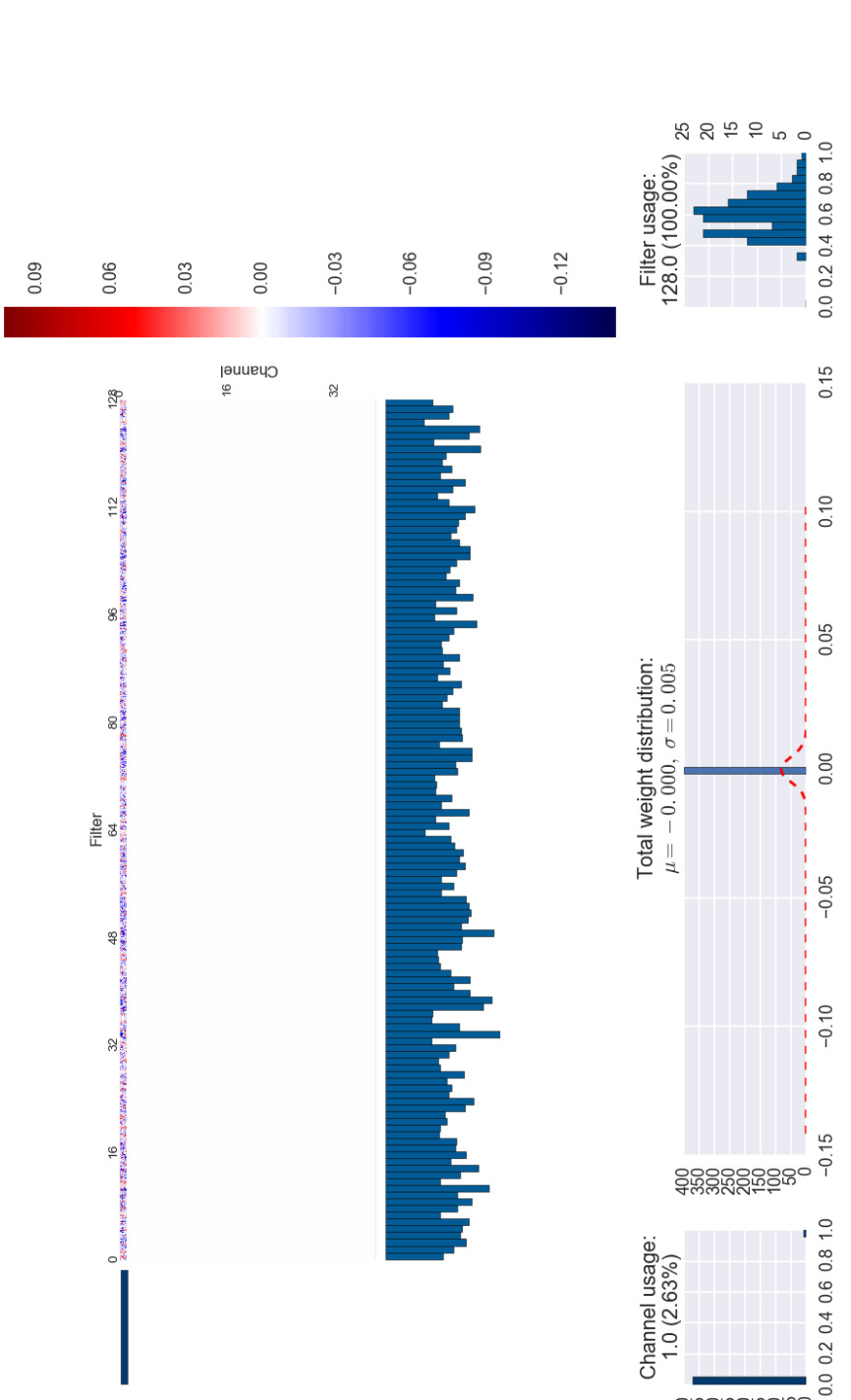


Figure A.8: Input channel usage when providing IRON-Features as input for a CNN

Only the first input channel containing the depth information (z) is used. The weights for all other channels (containing shape information) are zero. This visual interface is explained in section 6.2.2 in more detail.

Acronyms

cm centimeter. 31, 102, 103, 114

CNN Convolutional Neural Network. 10–15, 17–23, 26–29, 34–37, 39–42, 44, 45, 47, 48, 53, 57, 61–66, 70, 73, 76–79, 83–87, 90, 96, 101, 108, 109, 113, 114, 119, 120, 133, 137, 138, 142, 143, *Glossary:* Convolution Neural Network

CPM **C**onvolutional **P**ose **M**achine (CPM). 12, 36, 45, 62–64, 67, 68, 102–105, 119, 121, 142, 143, *Glossary:* Convolutional Pose Machine

fps frames per second. 30, 45, 77, 101, 102

Hz Hertz. 3, 4, 24, 50, 117, 127

IR infrared. 24, 25, 37, 38, 42–45, 47, 50, 58, 61, 86, 102, 103, 106, 123, 142

LIT Layer inspection Tool. 39, 73, 74, 76, 90, 114, 142, *Glossary:* LiT

m meter. 3, 41, 49, 51, 110, 112, 114, 117, 118

MPII MPII Human Pose Dataset (MPII). 13, 21, 22, 141, *Glossary:* MPII

ms milliseconds. 35, 96, 101, 102, 105, 114, 142

PCK$_\mathbf{h}$ **P**robability of **C**orrect **K**eypoint. 30–32, *Glossary:* Probability of Correct Keypoint

px pixel. 13, 14, 24, 37, 50, 63, 64, 66, 86, 131

RGB Red, Greed, Blue channels of an image. 9, 10, 17, 20, 45, 57, 63, 64, 66, 102, 103, 123, 131, 138

RGB-D Red, Greed, Blue and Depth channels of an image. 9, 10

ROS **R**obot **O**perating **S**ystem. 86, 96, 113, *Glossary:* ROS

RSN **R**e**SN**et50. 66–69, 142, *Glossary:* ResNet50

s seconds. 104

SHG **S**tacked **H**our**G**lass. 12–15, 21, 22, 62, 64–70, 77, 78, 113, 141, 142, *Glossary:* Stacked Hourglass

Glossary

C150K is a dataset of various human poses that was recorded for this thesis and is meant for training. It was created using data augmentation on the C25K dataset and contains 150,000 samples (see 6.2.1). 58, 71, 78

C25K is a dataset of various human poses that was recorded for this thesis and is meant for training. It was labeled using CapturyLive and contains 25,000 samples (see 5.5). 57, 58, 62, 63, 65, 66, 68, 70, 71, 137

Caffe is a deep learning framework. Version 1.0 [caf, 2017] of it was used to create and run all CNNs in this thesis. 79, 86

CEval is a dataset of various human poses that was recorded for this thesis and is meant for evaluation. It was labeled using CapturyLive and contains 1,400 samples (see 5.5). 58, 62, 63, 65, 66, 68, 95–99, 101, 102, 143

Convolution Neural Network is a specific form of Neural Networks that is especially useful for tasks where the data can be represented as images. It is described briefly in section 3.2.1. 10, 26, 113

Convolutional Pose Machine (CPM) is a network architecture for CNNs. It models the basic behavior of the Pose Machine described in [Ramakrishna et al., 2014] with a Neural Network. 62, 63, 67, 102, 121, 143

Human Pose Estimation is the process of estimation the configuration of the body pose. Here, this is realized by upper body joint detection. 1–3, 6–13, 15, 16, 22, 28, 29, 33, 34, 46–48, 62, 70, 83, 113, 119, 122, 138, 139, 141

IRON is a Fast Interest Point Descriptor for Robust NDT-Map Matching intoduced by [Schmiedel et al., 2015]. 10, 33–41, 43–45, 133, 142, 143

Kinect V2 is a camera that is able to capture RGB images as well as depth information using Time of Flight measurement. Here, only it's depth sensor is used as a data source. 3, 23–25, 105, 141

LiT is a visual tool to analyze the channel (weight) usage of a CNN as a guide for weight optimization. It is introduced in section 6.2.2. 39, 73

MPII is a human pose dataset for containing 25,000 annotated images from youtube with over 40,000 different people and 410 human activities. It is also a basis for benchmarks of Human Pose Estimation systems. 21

OpenPose is a state of the art real-time multi-person system to jointly detect human body, hand, and facial keypoints on single images [Ope, 2017]. 62, 102, 103

Probability of Correct Keypoint (PCK_h) is a metric for body joint detection. It describes the percentage of correct detected joints, where a detection is considered correct if its distance to the ground-truth position is smaller than a fraction p of the length of a specific body part. This fraction p is usually chosen to be 50%. The h in PCK_h denotes that the specific body part used for length reference is the head. 30

ResNet50 (RSN) is a basic CNN architecture that is often used for feature computation as a basis for succeeding tasks [He et al., 2016]. 66, 67

ROS (**R**obot **O**perating **S**ystem) is a set of software libraries and tools that act as a middleware for robotics applications. 86

Stacked Hourglass (SHG) is a network architecture for CNNs. It exploits the benefits of residual learning on multiple resolution scales to allow for a reliable recognition. Since its introduction in [Newell et al., 2016] in 2016, it quickly

gained high popularity in the field of Human Pose Estimation. 13, 62, 64, 67, 113

Time of Flight is the property of an object that can be exploited for distance measurement. This is briefly explained in section 3.1.1. 23–25, 117, 122, 123, 138

List of Figures

Bibliography

[gita] *GitHub - KaimingHe_ deep-residual-networks Deep Residual Learning for Image Recognition*. https://github.com/KaimingHe/deep-residual-networks.

[gitb] *GitHub - shihenw_ convolutional-pose-machines-release Code repository for Convolutional Pose Machines, CVPR'16*. https://github.com/shihenw/convolutional-pose-machines-release.

[MPI] *MPII Human Pose Database Benchmark*. http://human-pose.mpi-inf.mpg.de/#results.

[cud, 2015] (2015). *Cuda 7.5*. https://developer.nvidia.com/cuda-75-downloads-archive.

[caf, 2017] (2017). *Caffe BVLC 1.0*. https://github.com/BVLC/caffe/releases/tag/1.0.

[cap, 2017] (2017). *Captury Live*. http://thecaptury.com/captury-live/.

[CMU, 2017] (2017). *CMU Dataset*. http://domedb.perception.cs.cmu.edu.

[Ope, 2017] (2017). *No Title*. https://github.com/CMU-Perceptual-Computing-Lab/openpose.

[kin, 2017] (2017). *Official Kinect Specifications*. https://developer.microsoft.com/de-de/windows/kinect/hardware.

[Alejandro Newell, Kaiyu Yang and Deng, 2017] Alejandro Newell, Kaiyu Yang and J. Deng (2017). *Stacked Hourglass Networks for Human Pose Estimation (Demo Code)*. https://github.com/anewell/pose-hg-demo.

[Andriluka et al., 2014] Andriluka, Mykhaylo, L. Pishchulin, P. Gehler and B. Schiele (2014). *2D human pose estimation: New benchmark and state of the art analysis*. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 3686–3693.

[Arenknecht, 2016] Arenknecht, Robert (2016). *3D Person Sensing for Interactive Industrial Process Monitoring Robert Arenknecht*. PhD thesis, Technische Universität Ilmenau.

[Baak et al., 2011] Baak, Andreas, M. Muller, G. Bharaj, H. P. Seidel and C. Theobalt (2011). *A data-driven approach for real-time full body pose reconstruction from a depth camera*. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1092–1099.

[Bulat and Tzimiropoulos, 2016] Bulat, Adrian and G. Tzimiropoulos (2016). *Human pose estimation via convolutional part heatmap regression*. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9911 LNCS, pp. 717–732.

[Cao et al., 2017] Cao, Zhe, T. Simon, S.-E. Wei and Y. Sheikh (2017). *Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields*. In *CVPR*.

[Chu et al., 2017] Chu, Xiao, W. Yang, W. Ouyang, C. Ma, A. L. Yuille and X. Wang (2017). *Multi-Context Attention for Human Pose Estimation*.

[Eichner et al., 2012] Eichner, M., M. Marin-Jimenez, A. Zisserman and V. Ferrari (2012). *2D Articulated Human Pose Estimation and Search in (Almost) Unconstrained Still Images*. International Journal of Computer Vision, 99:190–214.

[Elhayek et al., 2015] Elhayek, A, E. De Aguiar, A. Jain, J. Tompson, L. Pishchulin, M. Andriluka, C. Bregler, B. Schiele and C. Theobalt (2015). *Efficient ConvNet-based marker-less motion capture in general scenes with a low number of cameras*. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 07-12-June, pp. 3810–3818.

[Gkioxari et al., 2016] Gkioxari, Georgia, A. Toshev and N. Jaitly (2016). *Chained Predictions Using Convolutional Neural Networks*. arXiv preprint arXiv, pp. 1–17.

[He et al., 2016] He, Kaiming, X. Zhang, S. Ren and J. Sun (2016). *Deep residual learning for image steganalysis*. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Huang and Altamar] Huang, Jingwei and D. Altamar. *Pose Estimation on Depth Images with Convolutional Neural Network*.

[Ian Goodfellow, Yoshua Bengio, 2015] Ian Goodfellow, Yoshua Bengio, Aaron Courville (2015). *Deep Learning Book*. Deep Learning, 21(1):111–124.

[Insafutdinov et al., 2016] Insafutdinov, Eldar, L. Pishchulin, B. Andres, M. Andriluka and B. Schiele (2016). *DeeperCut: A Deeper, Stronger, and Faster Multi-Person Pose Estimation Model*.

[Ioffe and Szegedy, 2015] Ioffe, Sergey and C. Szegedy (2015). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*.

[Ionescu et al., 2014] Ionescu, Catalin, D. Papava, V. Olaru and C. Sminchisescu (2014). *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE Human3.6M: Large Scale Datasets and Predictive Methods for 3D Human Sensing in Natural Environments*.

[Johnson and Everingham, 2010] Johnson, Sam and M. Everingham (2010). *Clustered Pose and Nonlinear Appearance Models for Human Pose Estimation.*. In *BMVC*, vol. 2, p. 5.

[Joo et al., 2016] Joo, Hanbyul, T. Simon, X. Li, H. Liu, L. Tan, L. Gui, S. Banerjee, T. Godisart, B. Nabbe, I. Matthews, T. Kanade,

S. Nobuhara and Y. Sheikh (2016). *Panoptic Studio: A Massively Multiview System for Social Interaction Capture*. pp. 1–14.

[Kadkhodamohammadi et al., 2017] Kadkhodamohammadi, Abdolrahim, A. Gangi, M. de Mathelin and N. Padoy (2017). *A Multi-view RGB-D Approach for Human Pose Estimation in Operating Rooms*.

[Li et al., 2014] Li, Sijin, A. B. Chan and A. B. C. Sijin Li (2014). *3D Human Pose Estimation from Monocular Images with Deep Convolutional Neural Network*. Accv, pp. 332–347.

[Martinez et al.] Martinez, Julieta, R. Hossain, J. Romero and J. J. Little. *A simple yet effective baseline for 3d human pose estimation*.

[Mehta et al., 2017] Mehta, Dushyant, S. Sridhar, O. Sotnychenko, H. Rhodin, M.-H. Shafiei, H.-P. Seidel, W. Xu, D. Casas and C. Theobalt (2017). *VNect: Real-time 3D Human Pose Estimation with a Single RGB Camera*. To Appear in ACM TOG.

[Newell et al., 2016] Newell, Alejandro, K. Yang and J. Deng (2016). *Stacked Hourglass Networks for Human Pose Estimation*. Eccv.

[Rafi and Leibe, 2016] Rafi, Umer and B. Leibe (2016). *An Efficient Convolutional Network for Human Pose Estimation*. pp. 1–11.

[Ramakrishna et al., 2014] Ramakrishna, Varun, D. Munoz, M. Hebert, J. Andrew Bagnell and Y. Sheikh (2014). *Pose machines: Articulated pose estimation via inference machines*. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8690 LNCS, pp. 33–47.

[Sapp and Taskar, 2013] Sapp, Benjamin and B. Taskar (2013). *Multimodal Decomposable Models for Human Pose Estimation*. In *In Proc. CVPR*.

[Sarafianos et al., 2016] Sarafianos, Nikolaos, B. Boteanu, B. Ionescu and I. A. Kakadiaris (2016). *{3D} human pose estimation: A review of the literature and analysis of covariates*. Computer Vision and Image Understanding, p. 1.

[Sarbolandi et al., 2015] Sarbolandi, Hamed, D. Lefloch and A. Kolb (2015). *Kinect range sensing: Structured-light versus Time-of-Flight Kinect*. Computer Vision and Image Understanding, 139:1–20.

[Schmiedel et al., 2015] Schmiedel, Thomas, E. Einhorn and H. M. Gross (2015). *IRON: A fast interest point descriptor for robust NDT-map matching and its application to robot localization*. In *IEEE International Conference on Intelligent Robots and Systems*, vol. 2015-Decem, pp. 3144–3151.

[Shotton et al., 2013] Shotton, Jamie, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman and A. Blake (2013). *Real-time human pose recognition in parts from single depth images*. Studies in Computational Intelligence, 411:119–135.

[Sun et al., 2017] Sun, Xiao, J. Shang, S. Liang and Y. Wei (2017). *Compositional Human Pose Regression*.

[Tekin et al., 2016] Tekin, Bugra, P. Márquez-Neila, M. Salzmann and P. Fua (2016). *Fusing 2D Uncertainty and 3D Cues for Monocular Body Pose Estimation*.

[Tompson et al., 2014] Tompson, Jonathan, A. Jain, Y. LeCun and C. Bregler (2014). *Joint Training of a Convolutional Network and a Graphical Model for Human Pose Estimation*. Advances in neural information processing systems, pp. 1799—-1807.

[Toshev and Szegedy, 2013] Toshev, Alexander and C. Szegedy (2013). *DeepPose: Human Pose Estimation via Deep Neural Networks*. 2014 IEEE Conference on Computer Vision and Pattern Recognition, pp. 1653–1660.

[Wei et al., 2016] Wei, Shih-En, V. Ramakrishna, T. Kanade and Y. Sheikh (2016). *Convolutional Pose Machines*. 2016 IEEE Conference on Computer Vision and Pattern Recognition, pp. 4724–4732.

[Wu, 2016] Wu, Jianxin (2016). *Introduction to Convolutional Neural Networks*. pp. 1–28.

[Yosinski et al., 2015] Yosinski, Jason, J. Clune, A. Nguyen, T. Fuchs and H. Lipson (2015). *Understanding Neural Networks Through Deep Visualization*.

[Zhang et al., 2012] Zhang, Licong, J. Sturm, D. Cremers and D. Lee (2012). *Real-time human motion tracking using multiple depth cameras*. Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, pp. 2389–2395.