Thomas Krinninger

One-Shot 3D Body-Measurement

MASTER'S THESIS

to achieve the university degree of
Diplom-Ingenieur

Master's degree programme
Telematics

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Dieter Schmalstieg
Institute for Computer Graphics and Vision

Dipl.-Ing. Dr.techn. Stefan Hauswiesner
Reactive Reality GmbH

Graz, Austria, Oct. 2016

Abstract

In recent years, online shopping has become very popular. A well-known problem for online cloth-retailers is the high return rate. One main reason for this high return rate is the customers inability to know which size will fit. If customer and retailer would have a consistent definition of anthropometric measurements, the fit or size selection could be automatically executed. Existing approaches that automatically obtain accurate and reproducible anthropometric measurements are either based on expensive non-portable hardware, are complex to use, or require significant computational power. Therefore, we developed a measurement system which aims to eliminate these problems.

Recent research in computer vision strongly relies on the use of Convolutional Neural Networks (CNNs). *CNNs* enable it to learn complex relations that can be evaluated with low computational effort. Therefore, we based our measurement pipeline on a cascade of different *CNNs*. To obtain anthropometric measurements using *CNNs*, we developed a data normalization strategy specifically for our pipeline. Using our measurement system, we were able to achieve a measurement accuracy comparable to specifically trained persons obtaining anthropometric measurements. The measurement pipeline we built was evaluated using synthetically generated human body models, which guarantees the reproducibility of all measurements.

Kurzfassung

Der Online-Handel mit Kleidung ist in den letzten Jahren stark gewachsen. Ein wesentliches Problem für die Händler besteht in der hohen Anzahl an Rücksendungen. Diese Rücksendungen sind oft dadurch begründet, dass der Kunde nicht wissen kann, welche Größe oder Passform die richtige für ihn ist. Diese Entscheidung kann automatisiert werden, wenn Kunde und Händler eine konsistente Definition der Körpermaße hätten. Bestehende Systeme, die Körpermaße automatisiert ermitteln können, basieren entweder auf teurer, nicht portabler Hardware, sind kompliziert anzuwenden oder sind sehr rechenaufwändig. Für diese Arbeit haben wir ein Messsystem entwickelt, das die beschriebenen Probleme elliminieren soll.

Aktuell sind Convolutional Neural Networks (CNNs) ein wesentlicher Bestandteil der Forschung im Bereich Computer Vision. *CNNs* ermöglichen es, komplexe Zusammenhänge zu lernen, welche danach mit wenig Rechenaufwand ausgewertet werden können. Deshalb haben wir uns entschieden, unser Messsystem aus einer Kaskade mehrerer *CNNs* aufzubauen. Um Messungen mittels *CNNs* zu ermöglichen, haben wir eine Datennormalisierung speziell für unser System entwickelt. Die erreichte Messgenauigkeit ist vergleichbar mit jener von Personen, die speziell auf die Ermittlung von Körpermaßen geschult sind. Die Evaluierung unseres Messsystems basiert auf synthetisch generierten Modellen des menschlichen Körpers. Dies garantiert die Reproduzierbarkeit aller Messwerte.

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used.

The text document uploaded to TUGRAZonline is identical to the present master's thesis dissertation.

Place	Date	Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Ort	Datum	Unterschrift

Acknowledgments

First and foremost I would like to thank Prof. Dieter Schmalstieg for giving me the opportunity to conduct research in this interesting field of computer vision and computer graphics at the Institute of Computer Graphics and Vision (ICG). Additionally, I want to thank my advisor Stefan Hauswiesner for the support. Whenever I needed help, he was there for me and supported me with his knowledge and his connections.

I think at this point it is also appropriate to thank my study colleagues and friends Georg, David, Erich, Reinmar, Johannes, Jörg, Christoph and Dominik for all the good times we had together at university and everywhere else. Additionally, I want to thank my colleagues and friends at Reactive Reality for providing the flexibility that is needed to work on the thesis and at the company at the same time.

Furthermore, I would like to thank my family for giving me the opportunity to finish university by supporting me whenever and however they could. Of course I also would like to thank my beloved girlfriend Vicky for being supportive and appreciative all the years I was studying at university. She was always a true enrichment for me since we met each other.

Contents

List of Figures

List of Tables

1 Introduction

Contents

1.1 Motivation

Online shopping is very popular today. A vast amount of clothes is bought over the internet. A major problem for online retailers is the high number of product returns. Customers often order the same garment in different sizes to choose and send back the others. This causes significant costs for the retailer.

If customers had an easy way of obtaining the anthropometric measurements of their own bodies, the return rate could be significantly reduced. Of course, the customer and the retailer would need to have a consistent definition of the anthropometric measurements relevant for selecting the correct clothes size. Current approaches for analyzing the accurate three-dimensional shape of the human body often require expensive hardware. Additionally, this hardware is often not portable and therefore not widely available.

Most existing approaches based on affordable consumer-hardware rely on user support. This means that the user needs to follow defined instructions, like for instance standing still for a specified amount of time, or perform specific motions.

We developed a measurement system that is capable to extract anthropometric measurements from a single depth-image with a minimum of computational expenses. Therefore, our approach is even feasible for performing anthropometric measurements on mobile devices.

Our measurement system is based on a combination of different Convolutional Neural Networks (CNNs). In state-of-the-art machine learning research, *CNNs* are a very important topic. These networks can learn very complex relations between their in- and outputs.

CNNs are computationally complex in the learning phase, but very efficient during runtime. This property makes them particularly interesting for anthropometric measuring. Once the system is trained, it can even run on mobile devices like smartphones or tablets.

1.2 Contributions and Outline

The goal of this project was to develop an anthropometric measurement system for the human body. The requirements of this system are that it should work with a single front-side depth-image only, and the computational expenses should be held at a minimum to make it suitable for mobile devices. The main challenge of a measurement system with these requirements is the lack of data of the back side of the human body. Therefore, circumference measurements are especially challenging.

We developed a solution based on a cascade of five *CNNs*. In the first place, we have a *CNN* detecting four regions in the depth-image that are important for anthropometric measurements. The other four *CNNs* extract the measurements from the corresponding depth-image regions. We developed a data-normalization strategy that enhanced the robustness as well as the accuracy of our measurement pipeline significantly.

The existing research and publications our work is based on is summed up in chapter 2. We describe the entire measurement pipeline, including the thoughts that led us to this solution in chapter 3. Chapter 4 contains an extensive evaluation of the results we achieved with our pipeline, including a comparison to the measurement accuracy of trained humans doing anthropometric measurements. Chapter 5 sums up our findings and provides an outlook on possible future research.

2 Related Work

Contents

This section gives an overview of related work. It describes tools we used, as well as the principles of the geometry and Convolutional Neural Networks (CNNs) we use. The order of the following sections is related to the order of steps in our pipeline. It starts with knowledge about our way of generating training- and test data of human bodies and their anthropometric measurements, followed by some geometric principles we use to calculate the data used to train our *CNNs*. Finally, we show the basic principles our *CNNs* rely on.

2.1 Make Human

MakeHuman [3] is a software for creating realistic 3D models of human bodies. *MakeHuman* is developed in Python and is open source. This fact makes in interesting for developers writing their own plugins for it. *MakeHuman* is based on 12 years of research in body topology. This software is a very crucial part in our research, since we rely on a huge amount of training data samples for our system. Getting real world depth images of more or less naked human bodies is difficult. Existing datasets of body models like *CAESAR*[1] would be more difficult to annotate conforming to our needs. Due to the ar-

[1] http://store.sae.org/caesar/

chitecture of our measurement system, we would need to manually annotate each single body model when using this dataset.

2.2 Camera Model

In this thesis, we work with the pinhole camera model. This model is just a simplified model of most real world cameras, which include lenses in their optics. Real cameras following the principles of this model also exist, but have limited usage due to their poor properties regarding light intensity. These cameras would theoretically have an infinitely small aperture. Figure 2.1 shows the geometric principle of the pinhole camera.

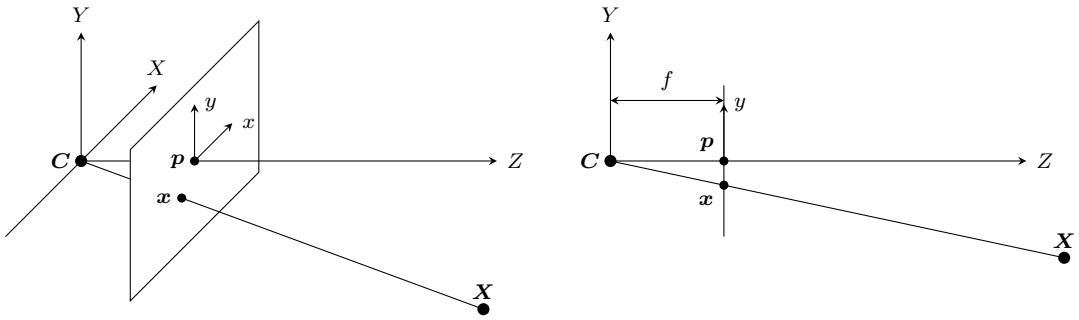

Figure 2.1: The pinhole camera model illustrated in 3D-perspective and 2D.

When observing the 2D illustration of the pinhole camera in figure 2.1 and taking similar triangles into account, we can derive the following relation:

$$\frac{\boldsymbol{X}_Y}{\boldsymbol{X}_Z} = \frac{\boldsymbol{x}_y}{f} \tag{2.1}$$

Accordingly, the x coordinates follow the relationship:

$$\frac{\boldsymbol{X}_X}{\boldsymbol{X}_Z} = \frac{\boldsymbol{x}_x}{f} \tag{2.2}$$

The pinhole camera model maps a three-dimensional point $\boldsymbol{X} = [X, Y, Z]^T \in \mathbb{R}^3$ to a two-dimensional point in homogeneous coordinates $\boldsymbol{x} = [x, y, w]^T \in \mathbb{R}^2$. This transformation can be described using a single matrix multiplication:

$$\boldsymbol{x} = \boldsymbol{K} \cdot \boldsymbol{X} \tag{2.3}$$

where $\boldsymbol{K}$ is defined as:

$$\boldsymbol{K} = \begin{bmatrix} f & 0 & -\boldsymbol{p}_x \\ 0 & f & -\boldsymbol{p}_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

The values $\boldsymbol{p}_{x,y}$ denote the coordinates of the principal point $\boldsymbol{p}$ in image coordinates.

2.3 Capturing Depth Images

Capturing three-dimensional information of real world scenes is a pretty large research topic. There are two basic concepts of capturing 3-dimensional scenes. One of the concepts aims to capture an entire scene, so that it can be rendered from arbitrary perspectives. The other concept is to just capture the scene from a single point of view. This way is similar to the human eye. The goal is to obtain the distance from the viewers position to any point in the scene that can be seen from there. Captured data of this approach is usually represented in rasterized depth images, also known as 2.5D images. In the following, we will present the two most common technologies to capture 2.5D images. Both technologies are affordable and available to the consumer market.

2.3.1 Stereo Triangulation

Stereo triangulation [7] uses two slightly different point of views in the same scene to obtain 3 dimensional data. The main challenge for this technology is to find the correspondences between two point of views. When a point in the real world scene is found in both images from both point of views, the 3 dimensional position can be calculated by triangulation.

Let $I_{1,2}$ be the image planes of two cameras with focal length f. Their centers of projection $C_{1,2}$ have a distance b, called the baseline. Both cameras project the point P on their image planes $I_{1,2}$. The x coordinate of the projection of P on the image planes $I_{1,2}$ is defined as $x_{1,2}$. Since the projected points $x_{1,2}$ can either be left or right in relation to their image centers, we need to define their signs. The sign of $x_{1,2}$ is defined positive, if the projection is right in relation to their respective image centers, and negative, otherwise. Figure 2.2 shows two different view points and illustrates the mentioned properties needed for triangulation. Using all these properties, the depth D can be estimated. D denotes the depth of the point P in relation to the camera centers $C_{1,2}$, as it is shown in this figure.

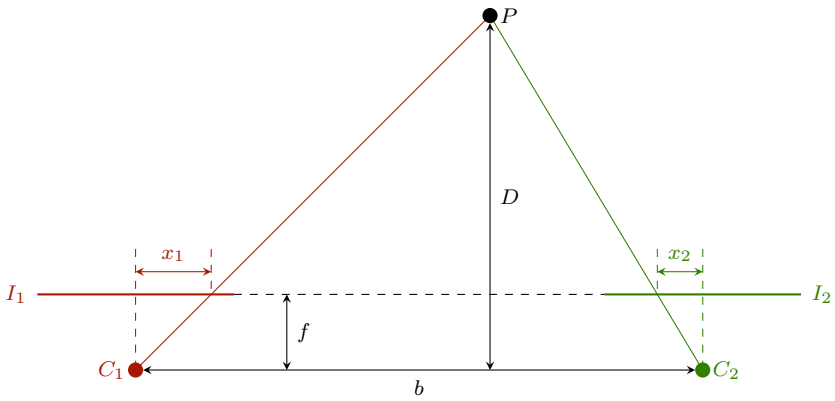

Figure 2.2: Normal case of stereo triangulation illustrated in 2D.

By observing similar triangles in figure 2.2, the following relation can be observed:

$$\frac{x_1 - x_2}{f} = \frac{b}{D} \tag{2.5}$$

When defining the disparity d as follows:

$$d = x_1 - x_2 \tag{2.6}$$

We can rewrite equation 2.5:

$$\frac{d}{f} = \frac{b}{D} \tag{2.7}$$

Solving for depth D then leads to:

$$D = \frac{f}{d} \cdot b \tag{2.8}$$

The described stereo triangulation is a simple case. The simplifications are that both cameras have the same intrinsic properties, namely the focal length f in this case. Additionally both cameras point in the exact same direction. Stereo triangulation is, of course, also possible for setups not compliant to these simplifications, but is slightly more complicated. The toughest challenge for this technology is the correspondence search needed to identify the position of a point in both camera images. Using this technology, there is also a pretty well known trade-off to tackle. A larger baseline b leads to higher accuracy, but can also lead to larger regions not visible to both cameras, which leads to regions in the depth image where no depth can be measured.

Figure 2.3: Photo of the well known *Kinect* RGB and Depth camera distributed by *Microsoft*. Image downloaded from web [2]

Figure 2.3 shows a photo of the well known *Kinect* [24] sensor distributed by *Microsoft*. What makes this sensor special is that it does not use two cameras to capture depth information. It has a built-in infrared projector, which projects a fixed pattern onto the scene. Additionally, there is a infrared-sensitive camera that detects the pattern. This

[2] https://en.wikipedia.org/wiki/Kinect Accessed on 24.8.2016

information can also be used to do stereo triangulation the way we described it. The *Kinect* sensor is widely used in computer vision and computer graphics research.

2.3.2 Time of Flight Cameras

Time of flight cameras use an entirely different approach than the cameras based on stereo triangulation. To capture depth images, they enlighten the scene with light pulses. For each pixel, the time until the light reaches the pixel is measured. The measured time is proportional to the distance of the object focused by the pixel. One can imagine the light going to the object and back to the pixel. This leads to the fact, that the light travels twice the distance between camera and object until it is measured again.

This technology eliminates the stereo triangulation's drawback of regions, where no depth can be measured due to the camera's baseline. One major challenge in time of flight camera technology is the filtering of light present in the scene to recognize the sent light pulse correctly. Reflections within objects in the scene can also lead to wrong measurements.

2.4 Convolutional Neural Networks

In this section, we describe the concept of *CNNs* as well as their components. *CNNs* are a state-of-the-art machine learning approach. *CNNs* build a specialization of feedforward neural networks [4, 15]. The basic idea of artificial neural networks originates from observations on the mammalian brain. Like the mammalian brain, artificial neural networks are built from neurons connected to each other. The task a single neuron does in the mammalian brain is considered rather simple. Since the brain enables complex learning using these rather simple elements for several different tasks, this principle has been adopted for machine learning problems.

The architectural specialization of *CNNs* is largely influenced by the work of Hubel & Wiesel [9]. In their experiments, they observed the neural activity of a cat's brain, depending on patterns they showed the cats. During these experiments, they found that for neighbouring cells in the visual cortex, the corresponding regions on the cat's retina are also spatially related. This fact was an inspiration for the topology of *CNNs*. Therefore, the neurons of a convolutional layer in a *CNN* are spatially related to each other. More specific, they are organized in a grid-layout like digital images. According to this, *CNNs* are especially relevant to computer vision research. Since *CNNs* take the spatial relationship of pixels into account, they provide a powerful tool for complex computer vision problems.

2.4.1 Overview

In this section we want to give an overview of artificial neural networks. We describe the topology these networks are built from, as well as their basic building blocks.

2.4.1.1 Neurons

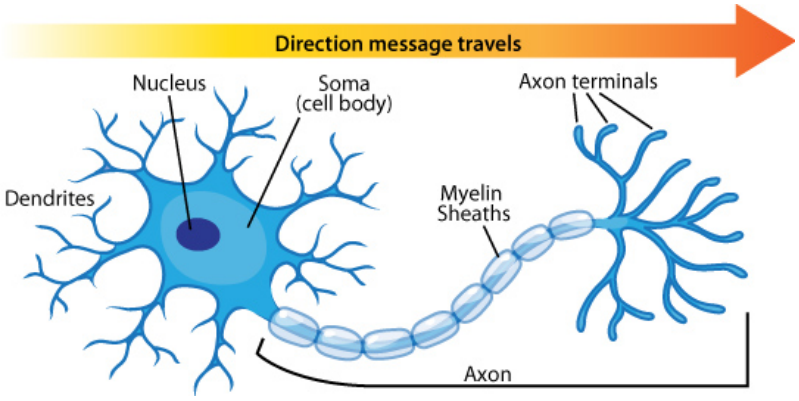

Figure 2.4: Illustration of the biological neuron. It can be seen, that there are multiple inputs (dentrites), but just a single output (axon). The axon is again connected to the dentrites of other neurons. Image downloaded from web [3]

The structure of the artificial neurons within the neural networks we use for machine learning is inspired by the biological neurons of the mammalian brain [20]. Figure 2.4 shows an illustration of the biological neuron. It can be seen that there are several inputs, but just a single output for a biological neuron. This fact hast been adopted when designing the artificial neurons used for machine learning.

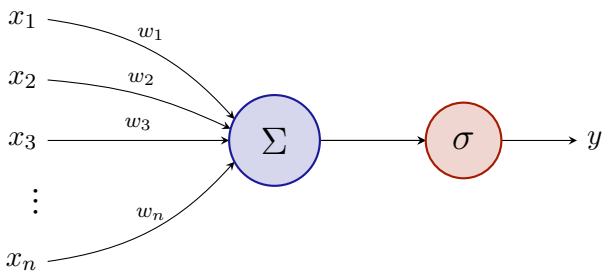

Figure 2.5: Illustration of the mathematical relations of an artificial neuron. The scalar output y results from feeding the weighted sum of the n input values x_1, ... ,x_n into an activation function $\sigma(x)$.

Figure 2.5 illustrates how the artificial neurons used in artificial neural networks are designed. Their functionality is just defined by calculating a weighted sum of x_1, ... ,x_n using the weights w_1, ... ,w_n and using this result as input for an activation function σ.

[3] http://biofoundations.org/?p=3283 Accessed on 19.7.2016

We can simply write this relation as follows:

$$y = \sigma\left(\sum_{i=1}^{n} w_i \cdot x_i\right) = \sigma\left(\boldsymbol{w}^T \cdot \boldsymbol{x}\right) \tag{2.9}$$

2.4.1.2 Activation Functions

Activation functions play a very important role for artificial neurons. In the beginning of artificial neural networks, there were linear activation functions present. An example for this is the Adaptive Linear Neuron (ADALINE) [23] neuron. One can consider that even neural networks with multiple layers can be replaced by a single linear function, if every neuron is based on a linear activation function. However, it is common practice to use non-linear activation functions. Only when using these non-linear activation functions, we are able to learn non-linear input/output relations, which makes the neural networks as powerful as they are.

Theoretically one can use any arbitrary function as activation function. In artificial neural network applications, some common activation functions have evolved. A very convenient fact about those common activation functions is, that they are implemented in most of the available neural network toolboxes. In the neural networks we use in our measurement pipeline, we always use the Rectified Linear Unit (ReLU) [13] activation function. In figure 2.6 a plot of the *ReLU* function can be seen. Its mathematical formulation is $y = max(x, 0)$. One problem that can arise when using the *ReLU* activation function called the dying *ReLU* [6]. This can occur, if none of the training samples generates an input to the *ReLU* greater than zero. Then the gradient of the *ReLU* is defined zero, which will cause its output to remain zero forever.

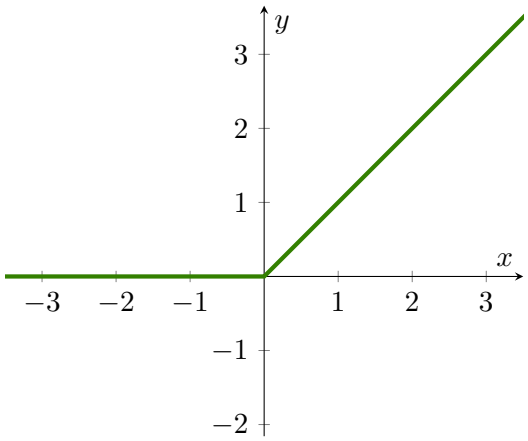

Figure 2.6: A plot of the ReLU activation function for artificial neurons. The mathematical relation is defined as $y = max(x, 0)$

2.4.1.3 The Layer Concept

Let $\boldsymbol{x}$ be an input sample, and $\boldsymbol{y}$, an output sample. We want to find a function $f(\boldsymbol{x}) = \boldsymbol{y}$ that holds for all possible samples. This is a basic definition of a common machine learning problem. *CNNs* form a set of possible functions that can be iteratively optimized to fit the desired data. These networks are built from layers. There is an input layer, an output layer and an arbitrary number of hidden layers in an artificial neural net. Figure 2.7 shows an illustration of the layers in a neural net, as well as their respective input and output values. The last layer shown in this figure is the loss layer, which is just needed when training the network to fit the sample data. The loss layer outputs a scalar value L, called the loss. The loss represents the error of the network. The learning process therefore tries to minimize the loss L.

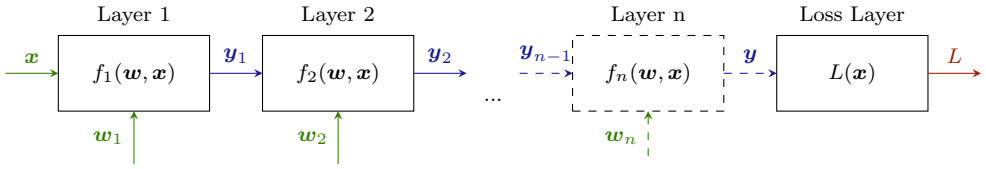

Figure 2.7: Block diagram showing the basic concept of neural networks. It shows the chaining of layers, as well as their respective input and output parameters.

2.4.2 Training

In this section, we describe the principles of learning the parameters of multi-layer neural networks. The learning process begins with a fully defined neural network. This means that its topology as well as all its parameters are defined. The goal is to modify the parameters of the network in a way that we can approximate a set of training samples as accurately as possible.

2.4.2.1 Loss Function

The loss function is necessary to train the parameters of a neural network. Consider the output for our neural network, defined as a vector $\boldsymbol{y}^1$. The desired output for a training sample is defined as $\boldsymbol{y}^2$. We want to define a loss function $L(\boldsymbol{y}^1, \boldsymbol{y}^2)$ and try to minimize its value during training. There are several different common definitions of L. The loss function should be carefully selected depending on the problem you want to solve.

For the training of our *CNNs*, we used the Euclidean loss function. The Euclidean loss function is a common choice for regression problems. Euclidean loss is defined as follows:

$$L(\boldsymbol{y}^1, \boldsymbol{y}^2) = \frac{1}{2n} \cdot \sum_{i=1}^{n} \left|\left| y_i^1 - y_i^2 \right|\right|^2 \tag{2.10}$$

This definition originates from the *Caffe Toolbox* [10], which we use to train and implement our neural networks. It can also be defined as the average Sum of Squared

Differences (SSD). It should be mentioned that this definition is based on the idea of batch learning. Therefore, the loss is computed for a set of n training samples.

2.4.2.2 Backpropagation

The process of optimizing the neural net's parameters to fit the desired training data is called learning. Artificial neural nets are trained using a gradient-based approach called backpropagation [16]. Figure 2.8 shows a block diagram illustrating the principle of backpropagation.

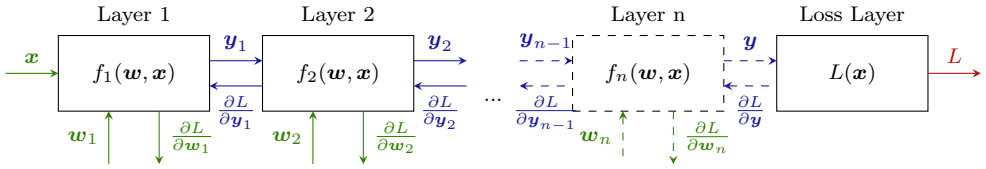

Figure 2.8: Block diagram showing the concept of backpropagation.

One can see that figure 2.8 is an extended version of figure 2.7. There are the derivatives of the Loss L with respect to the inputs y_i of each layer as well as the derivatives of L with respect to the layers weights $\boldsymbol{w_i}$ added. According to the chaining rule of derivatives, we are able to compute these derivatives as follows:

$$\frac{\partial L}{\partial \boldsymbol{w}_i} = \frac{\partial L}{\partial \boldsymbol{y}_i} \cdot \frac{\partial \boldsymbol{y}_i}{\partial \boldsymbol{w}_i} = \frac{\partial L}{\partial \boldsymbol{y}_i} \cdot \frac{\partial f_i(\boldsymbol{w}_i, \boldsymbol{y}_{i-1})}{\partial \boldsymbol{w}_i} \tag{2.11}$$

$$\frac{\partial L}{\partial \boldsymbol{y}_{i-1}} = \frac{\partial L}{\partial \boldsymbol{y}_i} \cdot \frac{\partial \boldsymbol{y}_i}{\partial \boldsymbol{y}_{i-1}} = \frac{\partial L}{\partial \boldsymbol{y}_i} \cdot \frac{\partial f_i(\boldsymbol{w}_i, \boldsymbol{y}_{i-1})}{\partial \boldsymbol{y}_{i-1}} \tag{2.12}$$

Assuming that we have the derivative of the loss L with respect to $\boldsymbol{y}$, we are now able to compute all gradients shown in figure 2.8, if we have the derivatives of all layer functions $f_i(\boldsymbol{w}_i, \boldsymbol{y}_{i-1})$ with respect to $\boldsymbol{w}_i$, as well as the derivatives with respect to $\boldsymbol{y}_{i-1}$. The name backpropagation comes from the fact that we propagate the gradients from the back of the network (loss layer) to the front of the network (first layer).

2.4.2.3 Gradient based Learning

Based on the gradients described in the previous section, it is possible to optimize the neural networks parameters. The main idea on gradient-based learning is that the negative gradient of the cost function $E(\boldsymbol{w})$ with respect to $\boldsymbol{w}$ guides to a minimum of the cost function $E(\boldsymbol{w})$. Therefore, a common problem for gradient based learning are local minima, where the learning gets stuck. To deal with this and other problems, there exist several gradient-based learning algorithms for *CNNs*.

2.4.3 Layers

CNNs are built from layers. Each layer can be defined by the following equation:

$$\boldsymbol{y} = f(\boldsymbol{x}, \boldsymbol{w}) \tag{2.13}$$

where $\boldsymbol{x}$ denotes the input of the layer and $\boldsymbol{w}$ denotes the parameters of the layer. Figure 2.7 shows how several layers can be combined to form a neural network we can use to tackle machine learning problems. In the following, we briefly describe the principles of the layers we used to build the *CNNs* for this thesis.

2.4.3.1 Fully Connected

Fully connected layers represent the basic concept of neural networks. In *CNNs*, these layers are usually used in the later layers of the network. In these layers, each scalar value of the input $\boldsymbol{x}$ is connected to each single value of the output $\boldsymbol{y}$. Considering $\boldsymbol{x} \in \mathbb{R}^{i \times 1}$ and $\boldsymbol{y} \in \mathbb{R}^{j \times 1}$ as column vectors, $\boldsymbol{y}$ can be evaluated as follows:

$$\boldsymbol{y} = \boldsymbol{W} \cdot \boldsymbol{x} + \boldsymbol{b} \tag{2.14}$$

where $\boldsymbol{W} \in \mathbb{R}^{j \times i}$ denotes the weight matrix, and $\boldsymbol{b} \in \mathbb{R}^{j \times 1}$ denotes the output bias. Referring to the basic layer formulation in equation 2.13, the weight matrix $\boldsymbol{W}$ and bias $\boldsymbol{b}$ together form the parameters $\boldsymbol{w}$ of the fully connected layer.

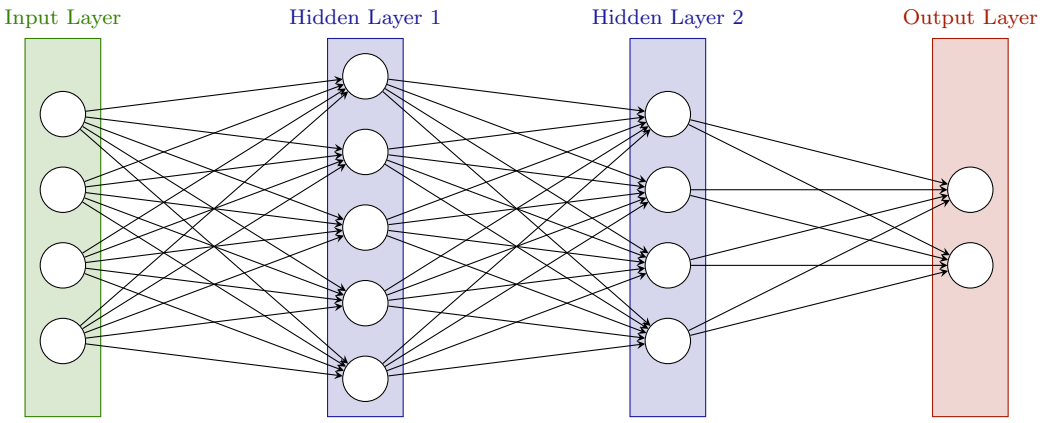

Figure 2.9: Fully Connected Neural Network with two hidden layers. The illustrated network includes 4 input and 2 output neurons.

Figure 2.9 illustrates how a fully connected neural network could look like. Each connection between two neurons is represented by a scalar weight. When using image pixels as input, one can imagine that the number of weights to learn increases drastically. This is a major reason why networks based on images usually start with convolutional layers to extract high level information from the rather low level pixel information.

2.4.3.2 Convolution

The convolution layer is especially relevant for networks trained on images. It can be considered as a special case of fully connected layers. This layer takes images as input and outputs images as output. Its parameters $\boldsymbol{w}$ are defined by a set of filter kernels. The output of the layer is defined as a set of images, where each output image is computed using one filter kernel. A filter kernel $\boldsymbol{K}$ is defined as a single image. The output image $\boldsymbol{O}$ computed using a filter kernel $\boldsymbol{K}$ and the input image $\boldsymbol{I}$ is defined as:

$$\boldsymbol{O}(i,j) = \sum_{k=1}^{m}\sum_{l=1}^{n} \boldsymbol{I}(i+k-1, j+l-1) \cdot \boldsymbol{K}(k,l) \tag{2.15}$$

where the filter kernel $\boldsymbol{K}$ dimension is $m \times n$. If we define the dimension of the input image $\boldsymbol{I}$ as $M \times N$ the output image $\boldsymbol{O}$ is of dimension $(M-m+1) \times (N-n+1)$.

2.4.3.3 Pooling

Pooling layers basically just perform down-sampling of their input. There exist several different kinds of pooling layers. In general, they perform a nonlinear downsampling operation on their input. One of the most common pooling layers is the max-pooling layer. This kind is defined by its window size. The input image is divided in sub-windows of the defined size. For each of these windows, the maximum contained value is written to a pixel in the output image. Within this thesis, we always use the max-pooling operation for pooling layers. The principle of max-pooling is illustrated in figure 2.10. This illustration is based on the parameters $size = 2$ and $stride = 2$.

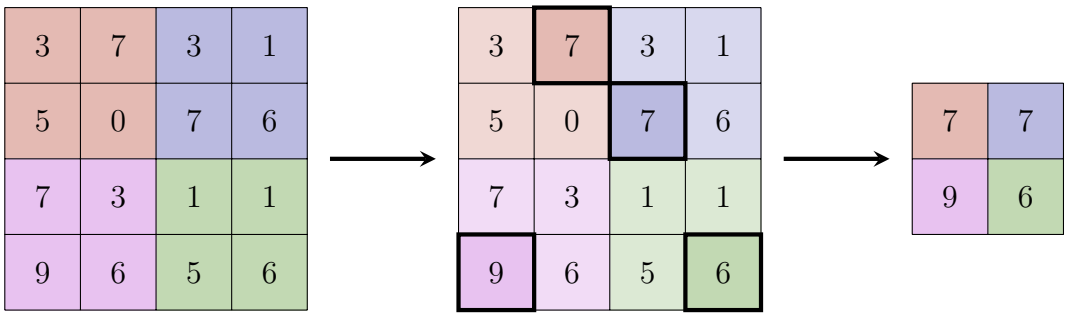

Figure 2.10: Max Pooling operation illustrated for a 4×4 image. The parameters are defined as follows: $size = 2$ and $stride = 2$.

2.4.4 Regularization

One of the challenges in machine learning tasks is to prevent overfitting. When considering *CNNs*, overfitting is a common problem due to their complexity. This means that they often can approximate large training data-sets by overfitting. This can be detected, if the training error is decreasing, but the test error does not decrease during training. Basically,

there are two ways two avoid overfitting. The first, obvious one would be to increase the size of the training data-set, which can be difficult depending on the data used. The second one is weight decay. In the following sections, we describe different approaches for regularization in *CNNs*.

2.4.4.1 Dropout

Dropout [18] is a regularization technique for artificial neural networks. The idea is to define a probability for a layer that defines the likelihood that each individual neuron gets dropped during a training iteration. Dropping in this context means to ignore the neurons output, or, in other words, set it to zero.

When testing the network, no neurons are dropped. To compensate for the dropping when testing the network, the dropout probability needs to be taken into account. This is simply achieved by scaling the outputs of the layer according to the dropout probability. Using this technique, it can be prevented that single neurons become too important for a small training error, making the solution more robust. In other words, this can be interpreted as a fusion of many different neural networks that share weights with each other.

2.4.4.2 Weight Decay

Weight decay is a regularization approach that penalizes large absolute weight values. The regularization is parameterized by a single scalar λ. This parameter changes the cost function $E(\boldsymbol{w})$ the following way:

$$\tilde{E}(\boldsymbol{w}) = E(\boldsymbol{w}) + \frac{\lambda}{2} \cdot \boldsymbol{w}^T \cdot \boldsymbol{w} \tag{2.16}$$

where $\boldsymbol{w}$ denotes the weights inserted in a column vector. This enforces the weights to have less higher peaks, the higher the parameter λ is defined.

2.4.5 Caffe Framework

During this project we used the *Caffe Deep Learning Framework* [10] for training and evaluation of our neural networks. We have chosen this framework because of its popularity in related research topics and the perfect suitability to our requirements. The framework delivers possibilities to easily define neural network topologies for your own needs. Additionally, it includes powerful learning algorithms out-of-the-box. The training process can also be accelerated using CUDA API [14] on hardware powered by suitable graphics processors.

2.5 Kaggle Facial Keypoint Detection

Developing the topology of a *CNN* from scratch is a rather difficult challenge. It is a common way to use an existing topology, that was used for a problem comparable to the problem one wants to solve. Therefore, we think that the *Kaggle Facial Keypoint Detection Challenge* [11] is closely related to our problem. To understand why we think that this challenge is closely related to our problem, one must consider that the keypoint detection and our measurement system both are regression problems. It should be mentioned that most publications on *CNNs* are related to classification problems. Additionally, the result of the keypoint detection *CNN* relies on single channel images as input, just like in our system.

Figure 2.11: Sample images of the kaggle facial keypoint detection. First row shows the images without keypoints. Second row shows the images with their corresponding keypoints. Image taken from [11]

The challenge consists in detecting keypoints in grayscale images of human faces. Each of the provided images of faces are 96×96 pixel in size. There are 15 keypoints to detect within each face. During the writing of this thesis, the challenge was still in progress. It ends on 31. Dec 2016. It is possible to find several solutions that were submitted to the challenge on the internet. We found that a solution that was published by *Balakrishnan*

Prabhu[4] works well for our problem. We have chosen to use their *CNN* as a starting point for our research.

2.6 Anthropometric Measuring

There are several different existing approaches to extract anthropometric measurements from 3D scanning data. In this section, we want to give an overview of these approaches. Additionally, we want to emphasize relevant differences to our approach.

The key difference between existing approaches and our system originates from the use of *CNNs* for extracting the anthropometric measurements. Our system is the first one performing anthropometric measurements using *CNNs*.

Basically, there are two major different kinds of input data possible for anthropometric measurement systems relying on 3D-scans. First, there are view-independent high resolution 3D-scans with good signal-to-noise ratio. These are usually created using expensive, professional hardware like laser scanners. It is also possible to create high resolution scans using consumer hardware, modern software tools and a controlled setup, for example, where the user is instructed to perform certain movements. In contrast, there are low resolution 3D-scans from consumer hardware that just include information visible from a certain point-of-view. The data captured from consumer hardware usually powers a much worse signal-to-noise ratio than the one captured from state-of-the-art laser scanners.

2.6.1 View-Independent 3D-Scans

In 3D-reconstruction systems, usually the goal is to create 3D-representations of an entire object. This means the object's shape is known all around the object. According to this, the appearance is known from any arbitrary point-of-view. Figure 2.12 shows an example of a pointcloud of a human body model shown from different viewing angles.

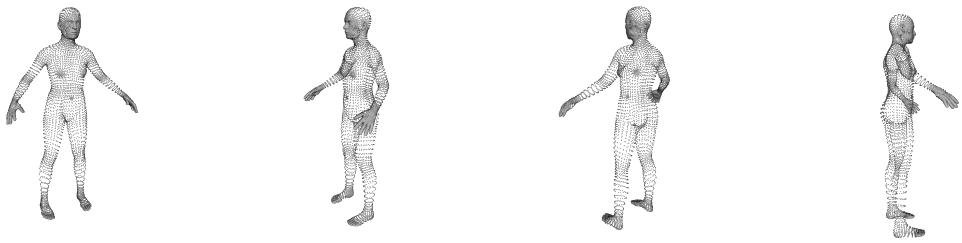

Figure 2.12: Rendered point-cloud of a human body model shown from different viewing angles.

One possibility to obtain anthropometric measurements from these scans, is to detect landmarks of the human body. These landmarks can be positions like elbow, shoulder and similar important anthropometric measurement positions of the human body. Using

[4]http://corpocrat.com/2015/02/24/facial-keypoints-extraction-using-deep-learning-with-caffe/

these landmarks, one can obtain distance measures like shoulder distance, leg length and so forth. To obtain circumferences, one needs to find the plane in which to measure the circumferences like upper-arm or bust circumference. Then one can try to measure these circumferences in the raw point cloud.

The mentioned approach for anthropometric measurement has the major drawback of relying on the accuracy of a rather small amount of points in the point-cloud. To overcome this limitation, other systems try to find a human body fitting the measured data. The trade-off is to find an anatomically correct human body model that fits the scan data by minimizing the distance between the model and the scan data [1]. There is a recent publication [19] on extracting anthropometric measurements based on model fitting and mesh registration [8]. The human body model they used is based on the SCAPE [2] body model.

The major advantage when registering a body model to the scan data is that one can get a realistic 3D body model of the captured body without any holes. Even when using professional 3D-scanning systems, holes are a common problem that is not yet entirely solved.

When finding an accurate body model for the scanned body, there are several different known approaches to obtain the anthropometric measurements of the human body. The approaches range from directly measuring distances and circumferences in the 3D-body model representation to learning the measurements relations to the shape parameters of the body model.

2.6.2 2.5D Scans

Using 2.5D scans of the human body, the rough estimation of the human body pose has already been implemented several different times with reasonable accuracy and runtime performance [17]. In contrast, the estimation of the shape of the human body is still a current research topic [22].

Comparing to view-independent 3D-scans, 2.5D scans not just suffer from holes in the scanned data, but are missing entire regions of the object's surface. The back side and all occluded regions of the object are not present in the scanned data. Theoretically, this would make the measurement of circumferences impossible due to the absence of scan data for important body regions. In practice, this leads to the fact that this problem can only be solved by using statistical models of the human body. One can imagine that the approach where a parameterized body model is registered to the measured data mentioned in section 2.6.1 is also a practicable way to attack the problem of the imperfect scan data.

2.7 Discussion

The sections in this chapter give an overview of principles and existing research our work is based on. One can see that our project is mainly based on computer vision, computer

graphics and machine learning research. We combine the mathematical definitions of camera geometry with the machine learning tools from *CNNs*.

3 Approach

Contents

In this chapter, we present the principles this project is based on. The technical ideas and processes are explained within this chapter. This includes a basic explanation of the process we use to do the anthropometric measurement, as well as technical details about every step in measurement itself and the training process of the system.

3.1 System Overview

This section gives an overview of the measurement system we developed. Our measurement system gets a depth image as input, and outputs anthropometric measurements like arm length, hip circumference and many others. It should be mentioned that within this project, we rely on synthetic data, since we were not able to capture enough humans for developing the system. For the reason of keeping the effort of this project in a reasonable range and focus on the measuring problem, we use already segmented depth images. This means that we know for every pixel of the depth image if the data comes from of the human body or background. Considering the segmented depth image as input to our measurement pipeline, figure 3.1 shows a block diagram of the stages to calculate the anthropometric measurements.

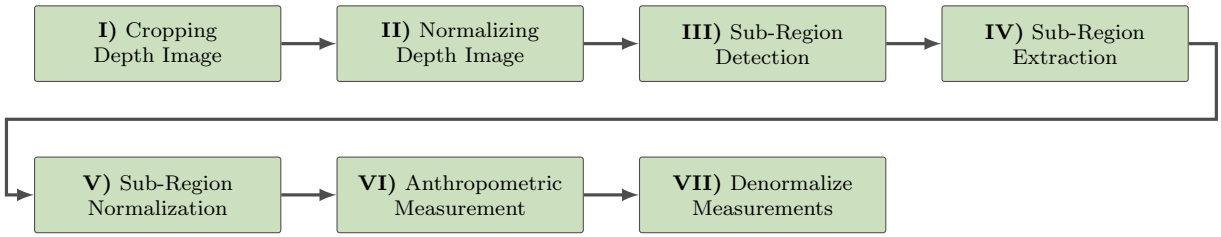

Figure 3.1: Block diagram illustrating the anthropometric measurement process. This process relies on segmented depth images as input.

Figure 3.2 illustrates the data for the individual steps of our measurement pipeline. It begins with the segmented depth image. We assume that the human body is roughly positioned in the center of the image. In the real world case, where one person would capture a depth image of another person, we would need to guide them to take the image that way. This could, for instance, be achieved by displaying a body silhouette in the camera viewfinder, which would also give the opportunity to tell the user which pose we desire.

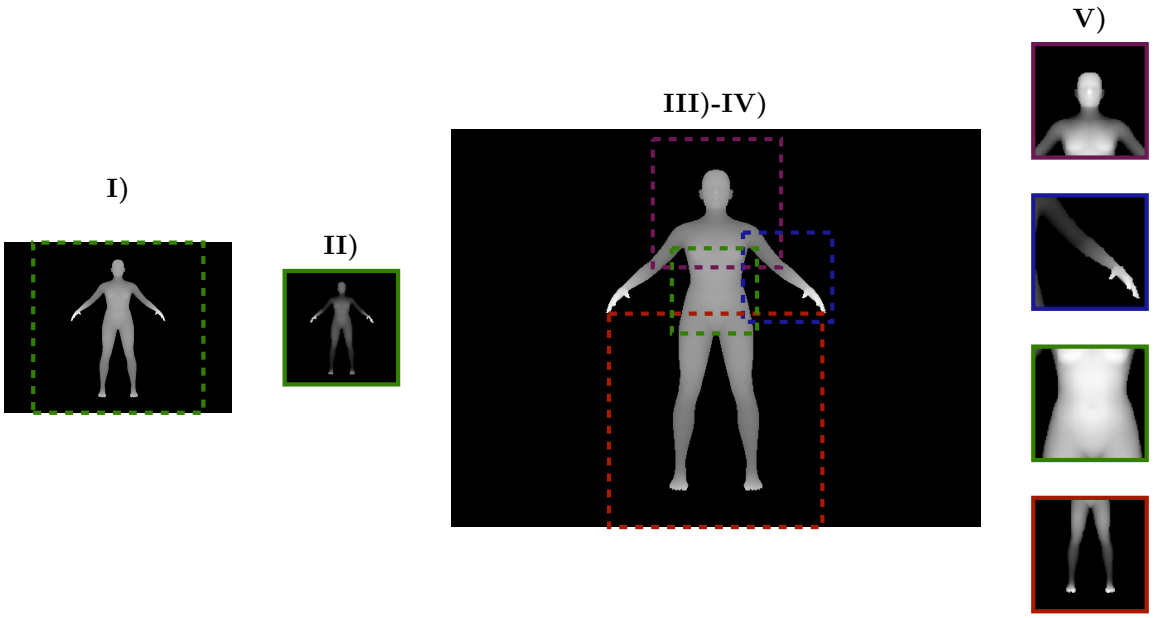

Figure 3.2: Images showing the intermediate steps of the measurement. The process is illustrated from left to right. **I)** shows captured depth image and the crop region illustrated in green. Followed by **II)**, which is the normalized crop region to feed the Convolutional Neural Network (CNN) for body region detection. **III)-IV)** illustrates the detected body regions and their extraction. Finally, **V)** shows the four normalized body region depth-images, which are again fed to *CNNs* to compute the anthropometric measurements.

Relying on the centered position of the human, we extract a centered quadratic region

of the depth image. A sample of the region we extract is shown in the leftmost image within figure 3.2. The next image in this figure shows the extracted region, normalized in a way our *CNN* can deal with. This normalization procedure is explained in section 3.5. Using this normalized depth image of the whole body, we find sub-regions using a *CNN*. We find 4 sub-regions, each aimed to deliver the information for extracting specific anthropometric measurements. The reason for extracting sub-regions is that *CNNs* usually deal with rather small image resolutions. If we would train a *CNN* to extract the anthropometric measurements from the depth image of the whole body, we would have to deal with a much more limited spatial resolution compared to our extracted sub-regions. When looking at the third image in figure 3.2, the four extracted body regions are highlighted. Using these regions of the full resolution depth image, we apply the normalization described in section 3.5, which results in the rightmost images shown in figure 3.2. Each of these images serves as input to individual *CNNs*. These networks are trained to extract a set of different anthropometric measurements each. The measurements, outputted from the networks, are normalized depending on the normalization step of each individual depth image sub-region. So the last step is to invert normalization and obtain the metric representations of the anthropometric measurements.

3.2 Training Data

This section contains information about the creation of data for training, testing and validating our measurement pipeline. This was a crucial part of our project, since we need a lot of data to train the *CNNs* included in the measurement pipeline. Considering collecting real world data, to get a single sample, we would need to find a person willing to be captured with depth sensors, just wearing tight fitting clothes. Additionally, we would need a possibility to take the anthropometric measurements of the person. Therefore we would need a reproducible measurement concept for anthropometric measurement of the human body. These challenges make it impractical for us to capture real word samples for training our *CNNs*.

There already are evaluations of the accuracy of anthropometric measurement [5] we will compare our results with. This Anthropometric Survey (ANSUR) is of interest for our evaluation, because it shows the best case of anthropometric measurement. It evaluates the accuracy for taking anthropometric measurements from U.S army soldiers measured by special trained personnel.

Publicly available data sets did not match our requirements. This forced us to create our own synthetic data set of human body depth images and their anthropometric measurements. To get anatomically correct human body models, we used the *MakeHuman* [3] application. Since *MakeHuman* is open-source, we were able to add some functionality to make it easier obtaining anthropometric measurements of created human body models automatically. We used the built-in scripting interface to create random human body shapes. The bodies vary in shape and size. The script we implemented creates a user-definable

number of arbitrary models, saves their 3D-model into *OBJ* files [12] and creates a text file for each model including the anthropometric measurements we need. Figure 3.3 shows four examples of human body models, randomly created using *MakeHuman*.

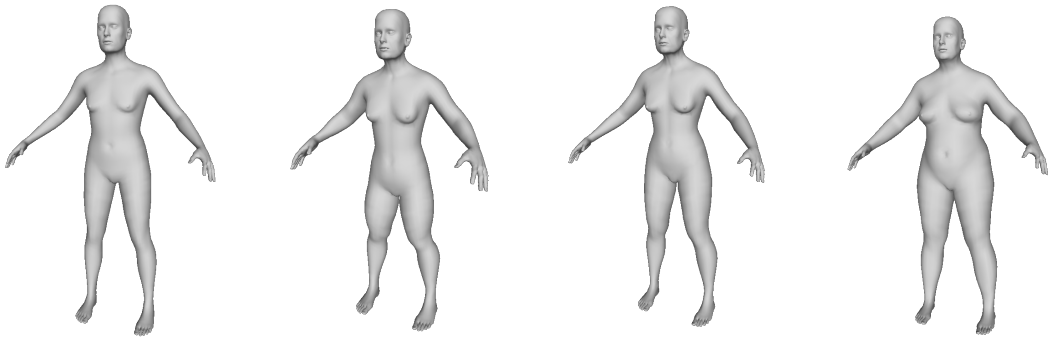

Figure 3.3: Renderings of four 3D body models created with *MakeHuman*. These are the models we use for training and evaluating our measurement pipeline

Since our approach relies on depth images of the human body, we had to render depth images of the created 3D-models. To achieve this, we used the *OpenSceneGraph* [21] API. We have chosen this API because of its built in functionality for loading and rendering *OBJ* files. We built an application that renders depth images for all previously created human body models.

3.2.1 Rendering Depth

OpenSceneGraph does not provide an API to render depth images by default. To render depth images, we used the fact that *OpenSceneGraph* is based on *OpenGL*. After rendering one of the 3D-models, we read the content of the *OpenGL* depth buffer. Since the content of the depth buffer is in *OpenGL* clipping space, we need to calculate the transformation back to metric space. To simulate the behavior of common depth sensors, we want each pixel to contain the distance between the corresponding 3D-position and the camera, more precisely, defined as the distance between the projection center of the camera and the normal projection of the 3D-Point to the viewing direction of the camera.

An *OpenGL* projection matrix $\boldsymbol{T}_{Proj}$, transforms a 3D-Position in camera coordinates $\boldsymbol{P}_{Cam}$ to a 3D-Position $\boldsymbol{P}_{Clip}$ following the relation:

$$\boldsymbol{P}_{Clip} = \boldsymbol{T}_{Proj} \cdot \boldsymbol{P}_{Cam} \tag{3.1}$$

where $\boldsymbol{T}_{Proj}$ is a matrix of size 4×4, and $\boldsymbol{P}_{Clip}, \boldsymbol{P}_{Cam}$ are represented using homogeneous coordinates with the following component representation:

$$\boldsymbol{P}_{Clip} = \begin{bmatrix} x_{Clip} \\ y_{Clip} \\ z_{Clip} \\ w_{Clip} \end{bmatrix} \tag{3.2}$$

$$\boldsymbol{P}_{Cam} = \begin{bmatrix} x_{Cam} \\ y_{Cam} \\ z_{Cam} \\ w_{Cam} \end{bmatrix} \tag{3.3}$$

If we rewrite equation 3.1, we get:

$$\boldsymbol{P}_{Cam} = \boldsymbol{T}_{Proj}^{-1} \cdot \boldsymbol{P}_{Clip} \tag{3.4}$$

Since we can get $\boldsymbol{T}_{Proj}$ used during rendering from the *OpenSceneGraph* API, we just need $\boldsymbol{P}_{Clip}$ to calculate the vector $\boldsymbol{P}_{Cam}$. We define the content of the *OpenGL* depth buffer as a matrix $\boldsymbol{D} \in \mathbb{R}^{h \times w}$, where w denotes the width and h denotes the height of the rendered depth image. Considering the definition of the *OpenGL* clipping space, we can define $\boldsymbol{P}_{Clip}$ corresponding to one depth buffer entry $\boldsymbol{D}(i,j)$, where i denotes the row and j the column of the value within the matrix $\boldsymbol{D}$ as:

$$\boldsymbol{P}_{Clip}(i,j) = \begin{bmatrix} x_{Clip}(i,j) \\ y_{Clip}(i,j) \\ z_{Clip}(i,j) \\ w_{Clip}(i,j) \end{bmatrix} = \begin{bmatrix} \frac{(2.0 \cdot (j-0.5)}{w} - 1 \\ -(\frac{(2.0 \cdot (i-0.5)}{h} - 1) \\ 2 \cdot \boldsymbol{D}(i,j) - 1 \\ 1 \end{bmatrix} \tag{3.5}$$

It should be mentioned that the values $\boldsymbol{D}(i,j)$ obtained from the *OpenGL* depth buffer are values in the range $[0,1]$, and the clipping space is in the range $[-1,1]$. To understand the calculation of $x_{Clip}(i,j)$ and $y_{Clip}(i,j)$, one must consider the image coordinates illustrated in figure 3.4. It can be seen that, for instance, the center of the pixel containing $\boldsymbol{D}(1,1)$ is located on coordinate $(0.5, 0.5)$. We consider the depth values to correspond to the pixel centers, which leads to the calculation of $\boldsymbol{P}_{Clip}(i,j)$ from equation 3.5.

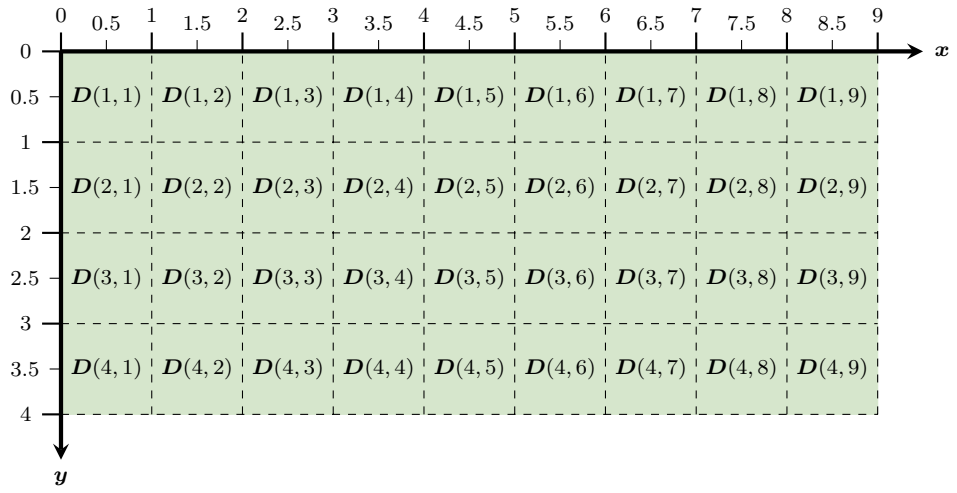

Figure 3.4: Illustration of image pixels and the image coordinate system. This figure should give an understanding of the image coordinates for pixel centers.

Using this, we are able to calculate a point $\boldsymbol{P}_{Cam}$ for each pixel $\boldsymbol{D}(i,j)$ of the *OpenGL* depth buffer by evaluating equation 3.4:

$$\boldsymbol{P}_{Cam}(i,j) = \begin{bmatrix} x_{Cam}(i,j) \\ y_{Cam}(i,j) \\ z_{Cam}(i,j) \\ w_{Cam}(i,j) \end{bmatrix} = \boldsymbol{T}_{Proj}^{-1} \cdot \boldsymbol{P}_{Clip}(i,j) \tag{3.6}$$

We have chosen to export our human body models from *MakeHuman* scaled, so that the unit of the coordinates is meter. This fact makes it possible for us to calculate the depth value for each single pixel of the rendered depth image $depth(i,j)$ by evaluating the following relation:

$$depth(i,j) = \frac{z_{Cam}(i,j)}{w_{Cam}(i,j)} \tag{3.7}$$

Using the calculated values $depth(i,j)$, we can save a depth image. This depth image is stored in a 16 bit single channel *PNG* file, where we store the depth as integer values representing millimeters of depth. We have chosen millimeters, since it is a common way depth is delivered by depth sensors like the *Microsoft Kinect* [24].

Figure 3.5 shows examples of the created training images. The upper images show textured RGB renderings of the models, and the lower images show the corresponding depth images as grayscale images.

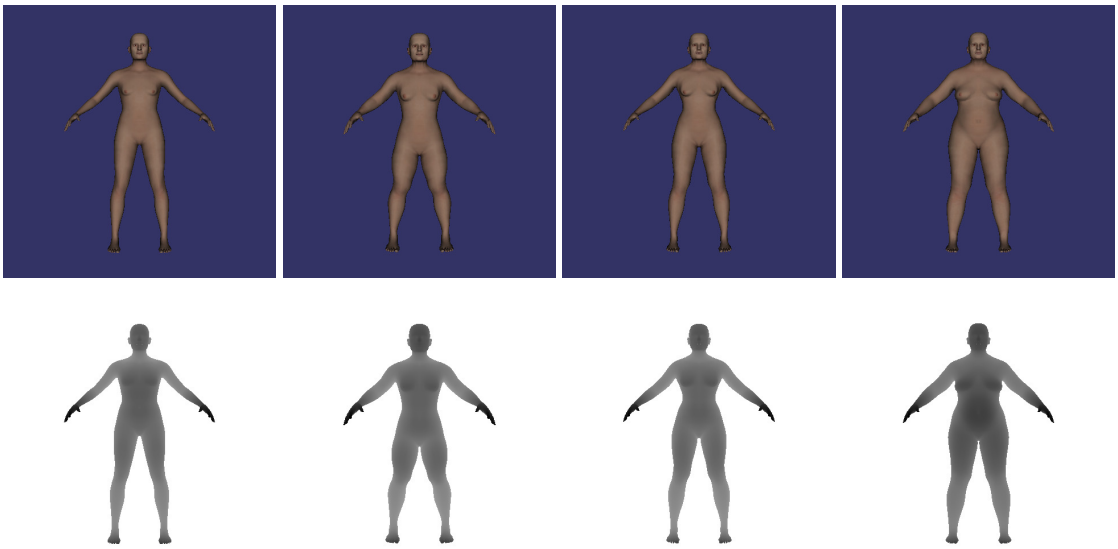

Figure 3.5: Depth and RGB Renderings of four different 3D body models. The upper row shows RGB- and the lower row depth-renderings, where vertically aligned images correspond to the same 3D body models

3.3 Neural Network Topology

A very crucial part in training machine learning systems is the mathematical model you choose for training. In our measurement pipeline, we use five separate *CNNs*. The topology of these *CNNs* is illustrated in figure 3.6. Hence we use the same topology for all five *CNNs*, the output's fully connected layer size is defined as N. In this figure, we show three different kinds of layers.

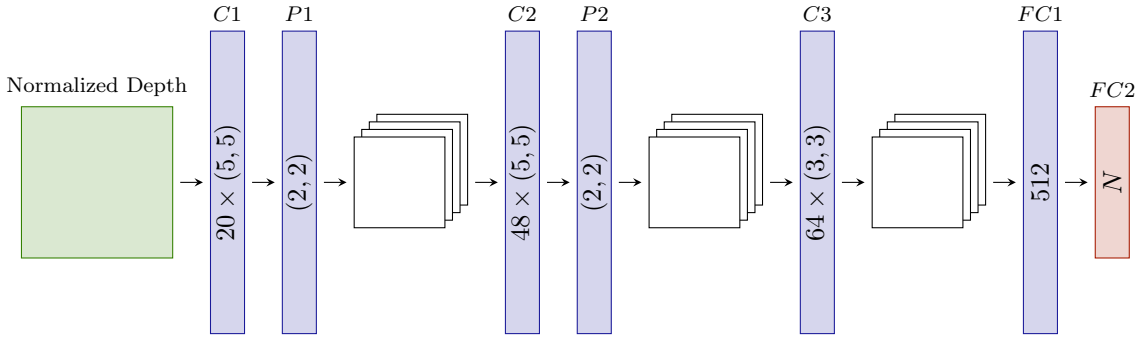

Figure 3.6: Illustration of the *CNN* topology we use in our measurement pipeline. For simplification reasons we did not include the regularizing dropout layers and the activation functions we use in this network.

The layers $C1$, $C2$, $C3$ denote convolutional layers, with their properties kernel size and

number of filters defined inside their rectangular illustration. $P1$, $P2$ denote the pooling layers with their pooling size defined in their rectangular illustration. Both pooling layers we use are MAX-Pooling layers. Finally, there are the layers $FC1$, $FC2$ which denote the fully connected layers with their neuron counts defined in their rectangular illustration. It should also be mentioned that every convolutional layer in our *CNNs* is followed by a Rectified Linear Unit (ReLU) as activation function.

Figure 3.7 shows how we combined five separate *CNNs*. There are several reasons why we have split up our measurement pipeline in more than one *CNN*. One of our main reasons for this decision was the lack of insight when using a single *CNN* for the entire measurement process. Due to our approach we were able to evaluate the measurement error more easily. We are able to separately obtain the error in body region detection, as well as the measurement error for each of our 4 measurement *CNN*.

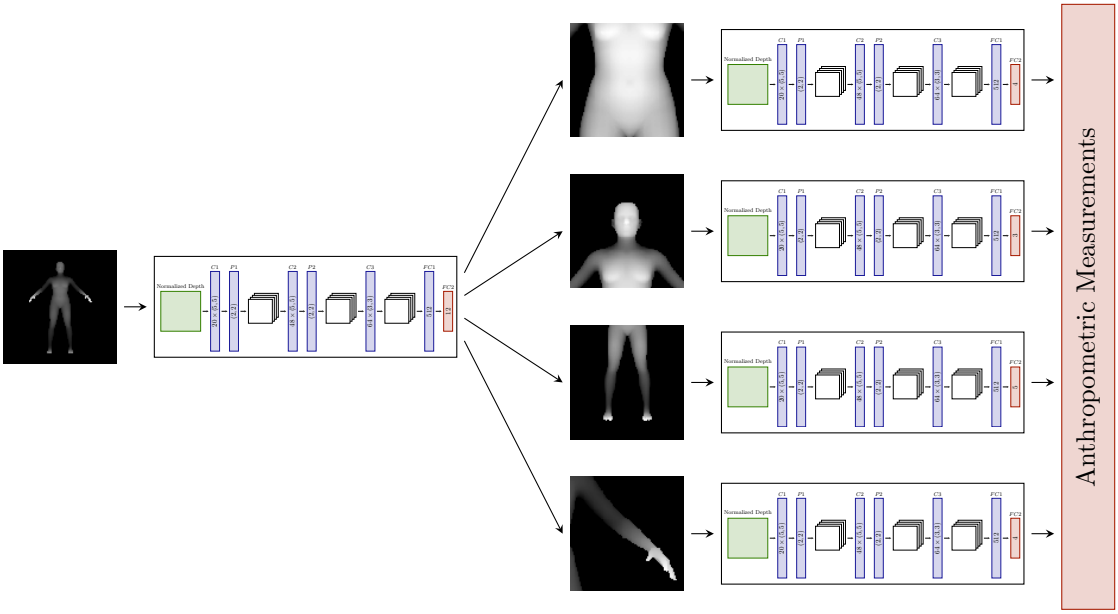

Figure 3.7: Illustration of our measurement pipeline topology. This figure shows how we connected the *CNNs*, but does not take the steps of data normalization into account. It shows all five *CNNs* we use, including samples of their input images.

Additionally, for the single *CNN* approach, we probably would have needed to train a network having much larger input size. The reason for this comes from the spatial resolution of the depth images fed into the *CNN*. Since we built a measurement pipeline aiming to get accurate measurements, we have to consider this fact. Our approach enables it to use *CNNs* with rather small input sizes, without the need to downsample the full-body depth images too much when extracting the anthropometric measurements. This is achieved thanks to the cascade of *CNNs* we use. The first network just tries to recognize the important regions for the measurement *CNNs* using a small representation of the full-body depth image, but the measurement *CNNs* then use the regions to extract their

input images directly from the full resolution body image. Therefore, we do not need to downscale the information of the full-body depth image too much for our measurement *CNNs*.

3.3.1 Regularization

For preventing overfitting to our training data-set, we did use the two common regularization approaches dropout and weight decay. Since our network topology is based on a network that was published by *Balakrishnan Prabhu*[1] and submitted to the *Kagge Facial Keypoint Detection Challenge* [11], we were able to benefit from their regularization approach. Therefore, we use the same parameters they did.

Specifically, this means that there is dropout enabled for the layers $P1$, $P2$, $C3$ and $FC1$ with the dropout probabilities $p(P1) = 0.1$, $p(P2) = 0.3$, $p(C3) = 0.5$ and $p(FC1) = 0.5$, where p denotes the probability a neuron in a layer gets dropped. Additionally, we use weight decay with $\lambda = 0.0005$.

3.4 Body Region Detection

Since our approach relies on sub-images of the full-body depth images, we need to define the selection of the sub-regions. In the first place, we needed to find sub-regions that work for extracting related anthropometric measurements. It was crucial to find a definition of these region in a way that we can automatically label them for all training samples. This enables us to fully automatically calculate training and test-data for our pipeline.

The automatic labeling of the depth image sub-regions is based on vertices of the human body mesh, projected into the depth image coordinate system. Since we just use synthetic data created using *MakeHuman*, we have a mesh with equal topology for each different human body shape. This enabled it for us to label landmarks of the human body in the depth images and select image-regions depending on the positions of the landmarks. Figure 3.8 shows a rendered depth image with highlighted landmark vertices of the human body mesh.

[1] http://corpocrat.com/2015/02/24/facial-keypoints-extraction-using-deep-learning-with-caffe/

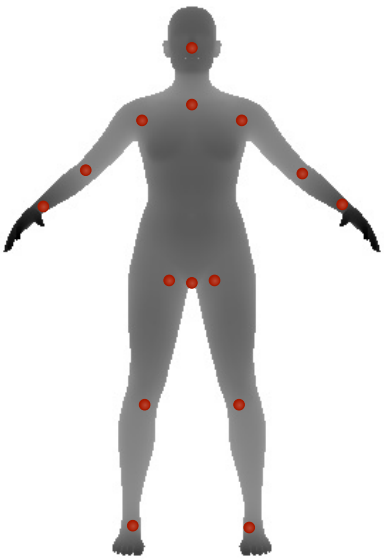

Figure 3.8: Landmarks of the human body mesh highlighted in the rendered depth image. These Landmarks are used to calculate the sub-regions used to train the body region detection covolutional neural network. The resulting regions are also used to extract the sub-regions for creating training samples needed for the measurement *CNNs*.

3.5 Data Normalization

A crucial part of training neural networks is the normalization of the image data fed into the network, as well as the normalization of the output data. Choosing a good normalization strategy can significantly affect the error of the network in a positive way. The main goal for normalization is to transform the data in a way that removes variance from the data we don't want the neural network to learn. The challenge is not to remove variance that would include information necessary to the machine learning problem.

Most *CNN* systems perform classification instead of regression. As a result, they employ different normalization methods. Even the regression systems available have a different type of normalization, because they rely on a feature extraction scheme that does not strongly rely on absolute values of the image pixels. For our problem, the absolute values are information that must be taken into account to extract accurate measurements. Therefore, we developed a normalization strategy suitable for our measurement pipeline.

3.5.1 Depth Image

The reason why we have to take the absolute values into account is because adding any value to the entire depth image will significantly distort the point-cloud represented by

the depth image. Even if the relative depth remains the same, the dimensions of the point-cloud are stretched/compressed parallel to the image plane. For example, mean subtraction of different body shapes could lead to the same result.

3.5.1.1 Depth Offset

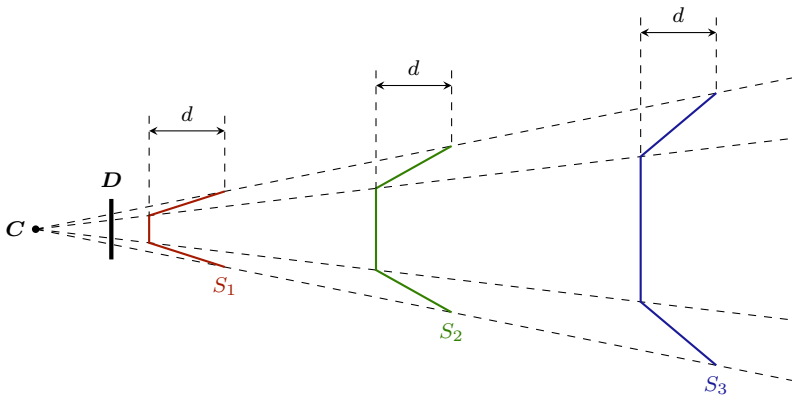

Figure 3.9: Illustration of different geometries resulting in the same depth images, just differing by a global depth offset. Note that these geometries are not just scaled versions of each other, but they are also distorted.

Figure 3.9 illustrates the mentioned problem. $S_{1,2,3}$ denote three different shapes. When projecting these shapes to the camera center $\boldsymbol{C}$ we get three depth images only differing by an absolute offset for the entire image. The value d denotes the depth range of the shapes $S_{1,2,3}$, which remains constant when adding an offset to the depth image. It can be seen that, when just adding a constant value to the entire depth image without scaling the depth, it leads to different stretched/compressed geometries represented by the transformed depth images.

To tackle this problem, one could consider normalizing the output of the network accordingly. Since we want to obtain metric measurements, where we do not know location and shape of the measurement paths for a given depth image, we are not able to determine the transformation implicitly applied to the output.

We do not want the variance in our depth images originating from absolute depth values to be much larger than the variance of the relative depth the human body covers in the image. To achieve this, we have to scale the depth values when adding an offset to the depth image. Figure 3.10 shows an illustration of three different shapes $S_{1,2,3}$, just differing by scale without any other distortions.

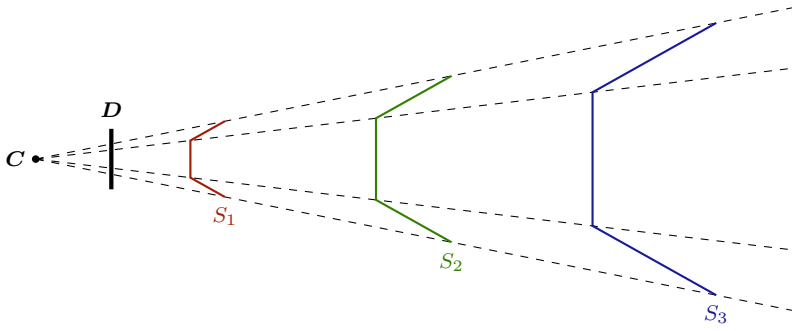

Figure 3.10: Illustration of different geometries just differing in position and scale. These geometries lead to the same depth images just differing by a global scale factor.

According to this illustration, we want to normalize our depth images in a way that geometry just differing by scale and appearing in the same image position results in the same depth images. The resulting scale must be taken into account when normalizing the output data of the *CNN*. To achieve this, we define an arbitrary reference depth value $d_{ref} = 1000mm$. We will show that the value of d_{ref} just results in the same scaling for all our normalized images. To remove the variance resulting from absolute depth values, we offset the depth image, so that the maximum depth value d_{max} of the geometry is moved to d_{ref}. It should be mentioned that the value d_{max} originates from the human body captured by the depth image rather than the background. We can achieve this only since we rely on segmented depth images in this stage of the pipeline. According to the similar triangles that can be obtained in figure 3.10, we can calculate the scale s_{offset} resulting from the depth offset as follows:

$$s_{offset} = \frac{d_{ref}}{d_{max}} \tag{3.8}$$

This scale leads to the following computation rule for a normalized depth value d' originating from a depth pixel value d:

$$d' = s_{offset} \cdot d \tag{3.9}$$

Since we want to use the resulting depth images to measure metric lengths, we need to make sure to avoid losing important information because of normalization. In the following, we show that the scaling of the depth values just scales any distance measure in the geometry by the factor s_{offset}. Consider a 3D-point $\boldsymbol{P}_i$ and its corresponding normalized point $\boldsymbol{P}'_i$. According to the pinhole camera model, we define this points as follows:

$$\boldsymbol{P}_i = \begin{bmatrix} x_i \cdot \frac{d_i}{f} \\ y_i \cdot \frac{d_i}{f} \\ d_i \end{bmatrix} \tag{3.10}$$

$$\boldsymbol{P}'_i = \begin{bmatrix} x_i \cdot \frac{d'_i}{f} \\ y_i \cdot \frac{d'_i}{f} \\ d'_i \end{bmatrix} = \begin{bmatrix} s_{offset} \cdot x_i \cdot \frac{d_i}{f} \\ s_{offset} \cdot y_i \cdot \frac{d_i}{f} \\ s_{offset} \cdot d_i \end{bmatrix} = s_{offset} \cdot \begin{bmatrix} x_i \cdot \frac{d_i}{f} \\ y_i \cdot \frac{d_i}{f} \\ d_i \end{bmatrix} = s_{offset} \cdot \boldsymbol{P}_i \tag{3.11}$$

where $(x_i, y_i)^T$ denotes the position of the points projection on the image-plane. f denotes the focal length of the depth camera. One can see that, regardless of these parameters, it holds that $\boldsymbol{P}'_i$ is $\boldsymbol{P}_i$ scaled by s_{offset}. Considering $\boldsymbol{P}_{1,2}$ to be two arbitrary points we can obtain the relation of their distances as follows:

$$\left|\left|\boldsymbol{P}'_1 - \boldsymbol{P}'_2\right|\right| = ||s_{offset} \cdot (\boldsymbol{P}_1 - \boldsymbol{P}_2)|| = s_{offset} \cdot ||\boldsymbol{P}_1 - \boldsymbol{P}_2|| \tag{3.12}$$

Since any curve in space can be approximated arbitrarily accurately by a connected set of points $\boldsymbol{P}_{1\ldots n}$, this shows that any length measure in the original geometry will be scaled by s_{offset} in the normalized geometry. Using this transformation, we achieved a normalization of the depth images that does not loose information, but helps focus our machine learning pipeline focus on body shape, rather than the absolute depth or just the body outline. This is a very important step in our pipeline, especially for guessing circumferences of human body parts. Imagining the bust circumference, the shape of the upper body front visible in the depth image is crucial for guessing if the circumference will be large or small. When having too large variance in the depth image due to the body outline, it would make it much harder to train the neural network seeing the differences in body shape.

3.5.1.2 Image Scaling

Since *CNNs* are trained with defined input and output size, we need to somehow scale or crop our input images to fit the input size of the network. The networks we use rely on a single channel 96×96 pixel image as input. To transform our depth images to the desired size, we select square regions inside the depth images and scale their size. The selection for the measurement regions is described in section 3.4. Since we already select square regions from the depth image, we just need uniform scaling along the x- and y-axis of the depth image regions. Again, we have to consider, that we do not want to loose information necessary to our measurement problem.

The size of the square depth image regions for anthropometric measurements is depending on the results of the body region detection. According to this, we need different scales for each depth image region to get the 96×96 pixel input resolution. Again we get different distortions originating from this transformation, where we have no possibility to predict the effect on the length measurements of the distorted geometries. To compensate for the distortion, we have chosen to scale the depth according to the scale factor of the

depth image dimensions. We can define the scale factor s_{scale} as follows:

$$s_{scale} = \frac{96}{w_{region}} \tag{3.13}$$

where w_{region} denotes the width of the square region in the original depth image. Since the region is square, width and height are identical.

The reason for this decision is originated from the simplification that the geometry is only located in a plane parallel to the depth image plane. Scaling in x- and y-dimension of the image leads to a scaling of the x- and y-axis of the geometry in the scene. When scaling the depth using the same factor, we would achieve the same uniform scaling along all three axis, described in section 3.5.1.1. Since our geometry is in a depth range larger than zero, i.e., it is not on a plane parallel to the depth image plane, this is only an approximate solution. The distortion resulting from the scaling is rather small, as long as the distance of the captured geometry is large compared to the depth range it covers. For our system, the human body is located at least two meters away from the camera, and the depth it covers is usually less than half a meter. One can consider the scale as a change in focal length of the camera, which is different for each single depth image, since the scale can be different.

The interpolation method we use for scaling our depth image regions to 96×96 pixels is nearest neighbor interpolation, to guarantee that no depth values are generated that were not captured.

3.5.2 Output Normalization

There exist several different approaches to normalize the values a *CNN* should produce as output. Our normalization strategy is based on the scaling factors s_{offset} resulting from the depth offset described in section 3.5.1.1 and the scaling s_{scale} of the image size described in section 3.5.1.2. Additionally, we subtracted the mean output value of the training data to get an average output of 0 for our *CNNs*. Since the variance of the different learned *CNNs* outputs differs, we normalize the outputs variance additionally.

Starting from a vector $\boldsymbol{m}_{metric}$, which denotes the anthropometric measurements, a *CNN* should learn. Each entry of the vector denotes a different anthropometric measurement in centimeter. We can compute a scaled vector $\boldsymbol{m}_{scaled}$ that compensates for the transformations of the corresponding depth image region:

$$\boldsymbol{m}_{scaled} = s_{offset} \cdot s_{scale} \cdot \boldsymbol{m}_{metric} \tag{3.14}$$

Using this transformation, we calculate the mean value of $\boldsymbol{m}_{scaled}$ over all training samples. We define this mean value as $\boldsymbol{m}_{mean}$. According to these definitions, we can calculate the

standard deviation $\sigma_{\boldsymbol{m}}$ as follows:

$$\sigma_{\boldsymbol{m}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} \left|\left|\boldsymbol{m}^{i}_{scaled} - \boldsymbol{m}_{mean}\right|\right|^{2}} \tag{3.15}$$

We use the standard deviation $\sigma_{\boldsymbol{m}}$ and the mean scaled output $\boldsymbol{m}_{mean}$ to calculate the final normalized output $\boldsymbol{m}'$ for training our *CNNs*:

$$\boldsymbol{m}' = \frac{1}{\sigma_{\boldsymbol{m}}} \cdot (\boldsymbol{m}_{scaled} - \boldsymbol{m}_{mean}) \tag{3.16}$$

By inserting the definition of equation 3.14, we get the formula to calculate $\boldsymbol{m}'$ we need for training data creation:

$$\boldsymbol{m}' = \frac{1}{\sigma_{\boldsymbol{m}}} \cdot (s_{offset} \cdot s_{scale} \cdot \boldsymbol{m}_{metric} - \boldsymbol{m}_{mean}) \tag{3.17}$$

Rewriting this equation brings us the computation rule for the anthropometric measurements $\boldsymbol{m}_{metric}$ in centimeter:

$$\boldsymbol{m}_{metric} = \frac{\boldsymbol{m}' \cdot \sigma_{\boldsymbol{m}} + \boldsymbol{m}_{mean}}{s_{offset} \cdot s_{scale}} \tag{3.18}$$

It should be mentioned that the scale factors s_{offset}, s_{scale} are different for each depth image sample. In contrast to this, $\sigma_{\boldsymbol{m}}$ as well as $\boldsymbol{m}_{mean}$ are constant among all samples evaluated on a specific measurement *CNN*.

3.6 Anthropometric Measurements

To do the anthropometric measurements on the human body depth image, we rely on four sub-region images of the full-body depth image. Each single sub-image is fed into a separate *CNN* to obtain the normalized anthropometric measurements available for the specific body region. The training data for anthropometric measurements originates from definitions of the *MakeHuman* [3] application. We were able to automatically obtain these measurement for each body model we created and store them into text-files. It is possible to get measurement values using the scripting engine of *MakeHuman*. The measurements we use in our measurement system are stated in table 3.1.

Anthropometric Measurement	*MakeHuman* Identifier
Upperarm Circumference	`measure/measure-upperarm-circ-decr|incr`
Upperarm Length	`measure/measure-upperarm-length-decr|incr`
Lowerarm Length	`measure/measure-lowerarm-length-decr|incr`
Wrist Circumference	`measure/measure-wrist-circ-decr|incr`
Upper Leg Length	`measure/measure-upperleg-height-decr|incr`
Thigh Circumference	`measure/measure-thigh-circ-decr|incr`
Knee Circumference	`measure/measure-knee-circ-decr|incr`
Lower Leg Length	`measure/measure-lowerleg-height-decr|incr`
Calf Circumference	`measure/measure-calf-circ-decr|incr`
Bust Circumference	`measure/measure-bust-circ-decr|incr`
Underbust Circumference	`measure/measure-underbust-circ-decr|incr`
Waist Circumference	`measure/measure-waist-circ-decr|incr`
Hip Circumference	`measure/measure-hips-circ-decr|incr`
Neck Circumference	`measure/measure-neck-circ-decr|incr`
Shoulder Distance	`measure/measure-shoulder-dist-decr|incr`
Neck Height	`measure/measure-neck-height-decr|incr`

Table 3.1: A list of all anthropometric measurements we used in our system including their identifiers needed to obtain them from *MakeHuman*.

4 Evaluation

Contents

In this chapter, we present the evaluation of our measurement pipeline. In the beginning we evaluate the accuracy for all five Convolutional Neural Networks (CNNs) we trained. Additionally, we show details about the training process itself. All results presented in this section originate from our synthetically generated training, validation and test data-set. The training data-set consists of 30000 randomized human body depth images including their ground-truth data. The validation and test data-set consist of 1000 independently generated random samples each.

More precisely, this means we evaluate the body region detection in the beginning, including its accuracy as well as its training process. Afterwards, the anthropometric measurements will be evaluated in the same manner, and, additionally, we compare our results to the measurement errors of manual measurements of anthropometric parameters [5]. In the last section, we discuss the limitations of our measurement pipeline.

4.1 Initialization of the CNNs

Initialization is a very critical point in *CNN*-based machine-learning systems. The initial properties of the *CNNs* can influence the progress in learning dramatically. A common way to initialize the properties would be to use the weights from another well-trained network, that is as similar as possible to your network. Of course similarity in this context

includes the topology as well as the application of the *CNN*. In our case, we did not find an existing trained network we could use, so we had to train our *CNNs* from scratch.

We were able to profit from the fact that all five *CNNs* we trained are similar in their topology, except for the output layer size. Even the input images are normalized in the same manner for all *CNNs*. To approve the concept before moving forward, we trained just a single network in the beginning. Since the most interesting problem to solve was the extraction of anthropometric measurements from a single depth image, we have chosen a measurement-*CNN* to start with. Considering our four different measurement-*CNNs*, we have chosen the *Upper Body Circumferences-CNN*. The reason for this is the fact that this network evaluates just circumferences of body parts, which is one of the crucial problems, since we cannot actually measure them because of the invisible back-side of the human body. Additionally, the evaluated measurements are pretty interesting in terms of tailoring clothes. All other *CNNs* were initialized with the weights of the *Upper Body Circumferences-CNN*.

To initialize the weigths of our *CNN*, we used a randomized initialization. Other initializations, like a constant value initialization did never seem to converge for our *CNN*. We used a *Xavier*-initialization pattern that is already implemented in the *Caffe Framework* [10].

4.2 Body Region Detection

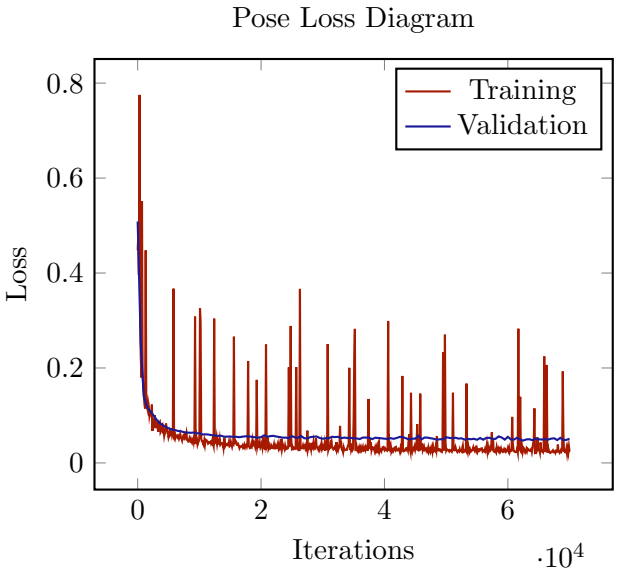

Figure 4.1: Diagram showing the loss of the *CNN*, trained for body region detection. The loss is displayed for the training as well as the validation data set. The peaks of the training loss originate from the batch learning approach of our *CNNs*. A single training iteration just uses a small subset of the training data-set to learn and compute the current loss value. Contrary the validation-loss is always computed using the entire validation data-set.

Our body region detection is a very crucial part in the whole pipeline. This is because the results define the input to our measurement-*CNNs*. If the regions we detect are inaccurate, this directly affects the results of the measurement. In section 4.4, we will evaluate the impact of the failure in body region detection on the results of the anthropometric measurements.

In Figure 4.1, it can be seen how the loss of the body region detection changes during the training iterations. It can be seen that the validation loss looks like it converged to a minimum. Regarding the peaks of the training loss, it should be mentioned that the batch size used in a single training epoch is much smaller than the entire training data-set. The validation loss is evaluated using the entire validation data-set, which explains why its graph is smoother than the train loss graph.

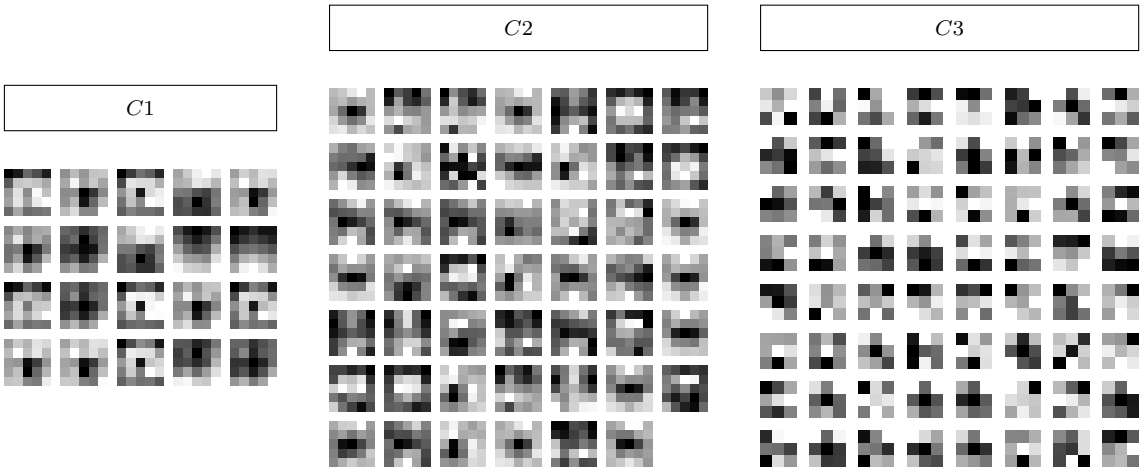

Figure 4.2: Visualization of the learned filters of the convolutional layers in the body region detection *CNN*.

Figure 4.2 shows the filter kernels of the body region detection *CNN*'s convolution layers. We normalized each filter kernel separately in a way that its pixel colors range from black to white. Due to this, it is not possible to see the relative importance of each filter kernel, but its structures are easier to obtain and compare. It can be seen that at least in the first convolutional layer of the *CNN*, there are some filter kernels of almost the same structure. We could have used a *CNN* using less filter kernels in the first convolutional layer achieving the same performance. It is also visible that some of the filters are rather complex in structure, or, in other words, contain rather high frequency parts. This can be a result of the *CNN* focusing on the high frequency depth discontinuities on the border of the human body in the depth images. This is a reasonable explanation, given that the selected body region boundaries correlate strongly with the body outline.

Table 4.1 shows accuracy related values of the body region detection *CNN*. All these values originate from evaluating the body region detection using our test data-set. The

unit of these values is defined in a percentage of the square images side length, which is fed into the body region detection *CNN*. There are 12 scalar outputs from the *CNN*. These are defined from four regions to detect, where each region is defined by 3 scalar values. These scalar values are the regions x- and y-position, as well as the side length of the corresponding square region. It can be seen that the overall performance looks promising, since the Mean Absolute Difference (MAD) is 0.17% at its maximum. Even the maximum error is just 3.35% of the square regions side length. Since our images are 640×480 pixels, and we just crop the center square with 480×480 pixels, the maximum pixel error is defined as $0.0335 \cdot 480 = 16.08$ pixels. Computing the *MAD* of 0.17% in pixels would lead to an error of $0.0017 \cdot 480 = 0.82$ pixels. This means the parameters of our detected body regions are on average more accurate than the size of one pixel.

	Max Abs. Diff [%]	Mean Abs. Diff [%]	Standard Deviation [%]
Arm X	0.32	0.07	0.09
Arm Y	0.37	0.09	0.11
Arm Size	0.87	0.16	0.20
Legs X	0.43	0.07	0.09
Legs Y	0.79	0.12	0.16
Legs Size	0.79	0.12	0.16
Upper Body Lengths X	0.53	0.07	0.10
Upper Body Lengths Y	1.13	0.15	0.19
Upper Body Lengths Size	0.95	0.13	0.17
Upper Body Circumferences X	1.53	0.11	0.15
Upper Body Circumferences Y	3.35	0.12	0.19
Upper Body Circumferences Size	3.33	0.17	0.26

Table 4.1: Accuracy related values for each measurement taken from the body region detection *CNN*. The presented values originate from feeding the *CNN* with the normalized square regions of the full depth images. All values represent percentages of the square region side length. There is an x and y position for each of the four regions that we detect, as well as the corresponding region size.

It should be mentioned that there is some room for improvement by trying variations of different definitions of the body regions we feed into the measurement-*CNNs*. This would mean that we would need to find new definitions for body regions relevant for corresponding anthropometric measurements. Using these definitions, we would need to train a *CNN* for each region we defined. According to this, it would require extensive work to just try a single combination of body regions for this measurement pipeline. To keep a reasonable scope for this work, we just implemented this single combination.

4.3 Anthropometric Measurements

This section gives an overview of the results of our anthropometric measurements of the human body. It includes information about the learning process as well as information for the results coming from the learned *CNNs*. The measurements of the learned *CNNs*, when fed with the synthetically defined body regions, will be compared to those of the same *CNNs*, when fed with the results of the body region detection *CNN*. Additionally, we present the learned filters of the convolutional layers, as well as the graph visualizing the progress in learning the parameters of our *CNNs*.

4.3.1 Upper Body Circumferences

As we already mentioned, this was the first *CNN* of our measurement pipeline we trained. This is maybe the most interesting one regarding the aim of this project, because it just extracts circumference measurements. This is a particularly interesting case, because we do not have enough information to do real measurement of these circumferences due to the invisible back-side on our depth images. This can be interpreted as a statistical problem, where we need to guess how the human body's back side is shaped. In existing approaches, this problem was attacked by learning statistical models of the human body. For our approach, we do not need to guess the geometry of the back side, because the *CNN* should learn the connection between the bodies front shape and its measurements directly.

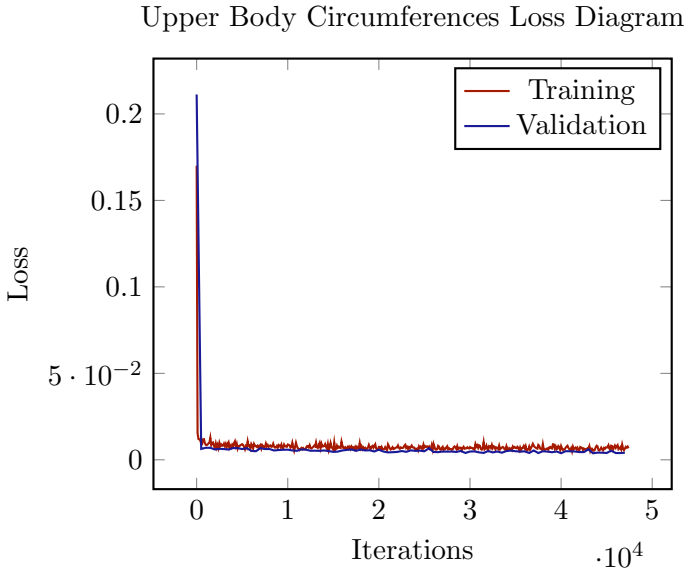

Figure 4.3: Diagram showing the loss of the *CNN* trained for upper body circumferences measuring. The loss is displayed for the training as well as the validation data set.

Figure 4.3 shows the loss during training including the loss of the validation set. In this

diagram, it can be observed that the loss immediately drops to the minimum it reaches during training. This is because we initialized it with the weights of another version of the same *CNN*, where we just used a slightly different normalization of the output values. Thanks to this fact, it learns very fast, since it just needs to adopt to the different output normalization.

In figure 4.4, we show the filters of the upper body circumference *CNN*'s convolutional filter kernels, like we did for the body region detection in figure 4.2. A major difference can be observed when looking at the filter kernels of the first convolutional layer $C1$. In comparison to those of the body region detection, these are much smoother, or, in other words, there are less high frequency components in it. This suggests that the net responds better to smooth changes in the depth images than it responds to sharp edges like the body outline. According to this, it is likely that our learned *CNN* really incorporates the shape of the upper body front to obtain the anthropometric measurements. This is what we wanted to achieve, since it can make huge differences for circumferences of the body, if the shape of the body differs, even if the outline seen from the front is almost the same. Just consider a pregnant woman, whose upper body outline seen from the front will not change so much, but the depth information of a captured depth image would.

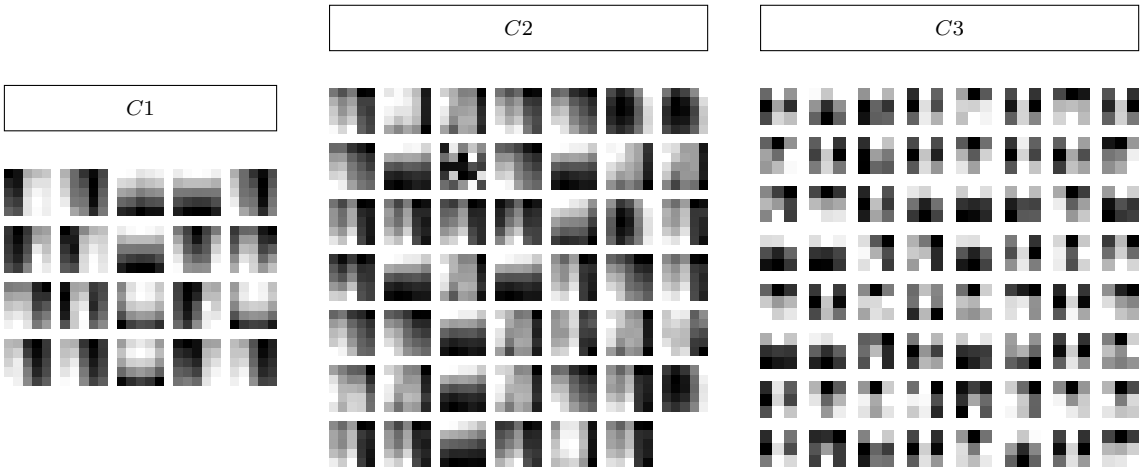

Figure 4.4: Visualization of the learned filters of the convolutional layers in the upper body circumferences measurement *CNN*.

In table 4.2, we summarized the accuracy statistics of the upper body circumference *CNN*. There are the *MAD*, the maximum absolute error as well as the standard deviation for each of the four measurement the *CNN* evaluates in the table. It can be seen that for all circumference measurements extracted using this *CNN*, the *MAD*, as well as the standard deviation, do not exceed the size of one centimeter. When looking at the maximum absolute differences, we can at least tell that the *CNN* does not create entirely unrealistic results for all test-data samples we evaluated.

To obtain the values shown in this table, we used the ground-truth data of our test data-set to crop the region of the full depth image we feed into the *CNN*. The reason for this is that we wanted to get an evaluation of the *CNN* without incorporating the failure of the body region detection *CNN* in the results. Of course, we also evaluated the results of the full pipeline. This evaluation is summarized in table 4.6 for all anthropometric measurements we extract from the depth images.

	Max Abs. Diff [cm]	Mean Abs. Diff [cm]	Standard Deviation [cm]
Bust Circumference	3.44	0.59	0.80
Underbust Circumference	4.54	0.56	0.76
Waist Circumference	3.30	0.56	0.75
Hip Circumference	2.79	0.49	0.64

Table 4.2: Accuracy related values for each measurement taken from the upper body circumference measurement *CNN*. The presented values originate from feeding the *CNN* with the synthetic body region data rather than the output of the body region *CNN*.

4.3.2 Arm Measures

In this section, we evaluate the *CNN* we trained for extracting anthropometric measurements of the arm. The arm measurement *CNN* is trained on the left arms of our training data-set models only. For our approach, this totally suffices, since the body models we train on are perfectly symmetrical. Even if we would have models where the measurements of the arms are not equal for both arms, we could use the same approach, since we would just need to horizontally flip the entire depth image and evaluate the body region detection, as well as the arm measurement *CNN* using the flipped image to obtain the measurements of the right arm. Then, we could extract our arm measurements for left and right arms separately. We did not evaluate the results of this procedure, since, for our symmetrical models, this would obviously lead to the exact same results.

Figure 4.5 shows a graph of the training and validation loss during training. Comparing this graph to the one of the upper body circumference *CNN* shown in figure 4.3, it can be observed that the training did take more iterations to approximately reach the minimum loss. This could be expected, since the arm measurement *CNN* was initialized with the trained parameters of the upper body circumference *CNN*. According to this, it needed to adopt its parameters in a way that suits for arm measurement. In contrast, the graph for upper body circumference measurements shows just the loss during learning a different output normalization.

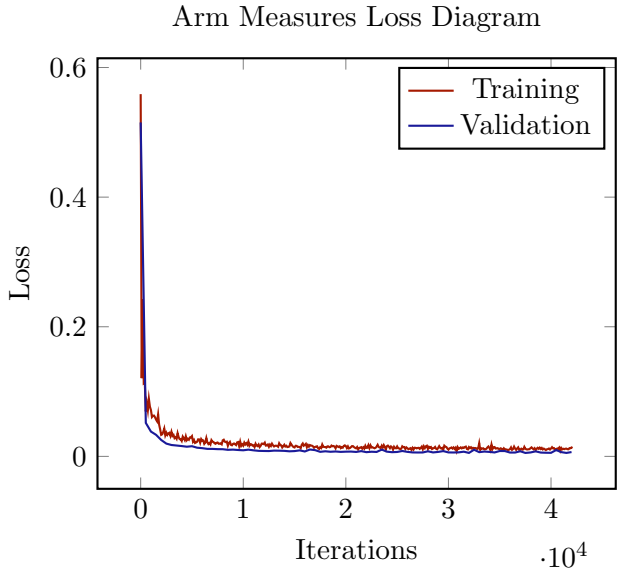

Figure 4.5: Diagram showing the loss of the *CNN* trained for extracting arm measures. The loss is displayed for the training as well as the validation data set.

When looking at the filter kernels of the convolutional layers of the arm measurement *CNN* in figure 4.6, it can be seen that many of them have some kind of diagonal pattern. The obvious explanation for this fact is that when looking at the depth-image sub-regions we feed into the network, the arm is also appearing in an approximately 45 degree angle from top left to bottom right. An example of a sub-region we feed into the *CNN* can be seen in figure 3.7. Since the arm's outline is not constant, it is reasonable that the *CNN* somehow needs to respond to the arm's structure to get accurate measurements.

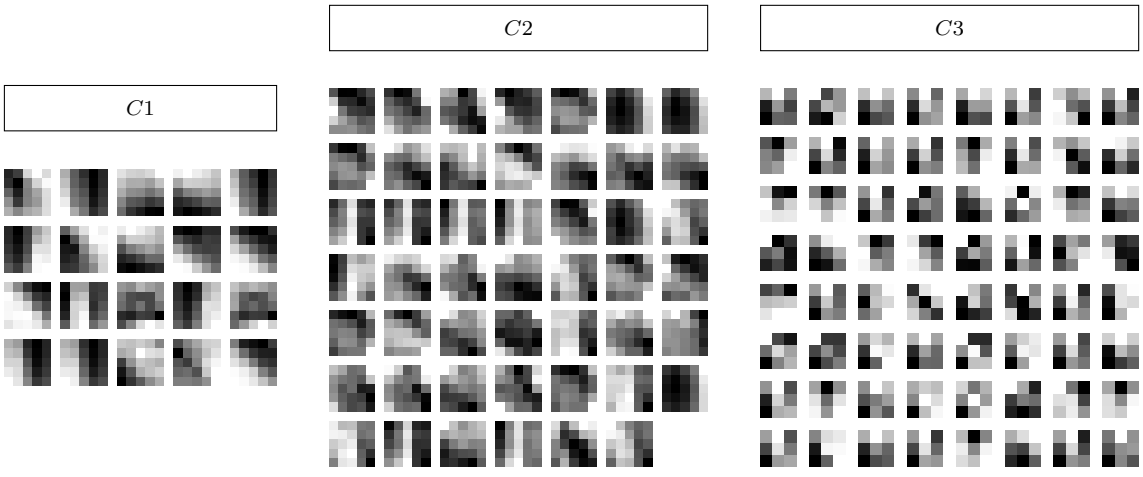

Figure 4.6: Visualization of the learned filters of the convolutional layers in the arm measurement *CNN*.

The accuracy of the arm measurements is summarized in table 4.3. Again, these values originate from feeding the *CNN* with the ground-truth arm regions, rather than the ones we detect with our body detection *CNN*. The table shows that, for arm measurements, we achieved that the *MAD* as well as the standard deviation for all separate arm measurements are below half a centimeter. Even the maximum errors do not exceed two centimeters. These results are promising since it is rather difficult to detect the elbow in the depth image, which is crucial for obtaining the upper arm and lower arm length.

	Max Abs. Diff [cm]	Mean Abs. Diff [cm]	Standard Deviation [cm]
Upperarm Circumference	1.57	0.28	0.37
Upperarm Length	1.39	0.19	0.26
Lowerarm Length	1.13	0.15	0.20
Wrist Circumference	0.82	0.16	0.20

Table 4.3: Accuracy related values for each measurement taken from the arm measurement *CNN*. The presented values originate from feeding the *CNN* with the synthetic body region data, rather than the output of the body region *CNN*.

4.3.3 Upper Body Lengths

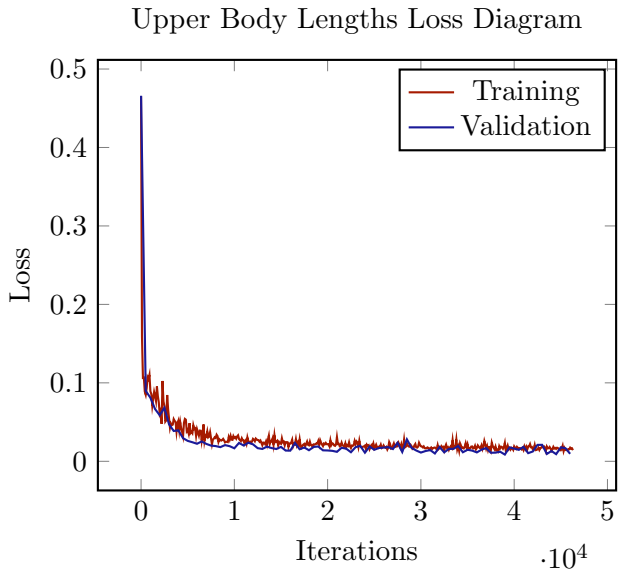

Figure 4.7: Diagram showing the loss of the *CNN* trained for upper body lengths measuring. The loss is displayed for the training as well as the validation data set.

In this section, we evaluate the upper body lengths *CNN*. The loss on the train and validation data-set is plotted in a graph shown in figure 4.7. It can be observed that it

took approximately a thousand iterations to reach the loss where the graph flattens out and the loss just improves marginally afterwards. Again, it can be seen that the *CNN* does not seem to overfit the provided training-data, since the validation data loss pretty much follows the curve of the training loss.

Figure 4.8 shows the filter kernels of the convolutional layers contained in the upper body lengths *CNN*. Interestingly, the structures seen in the filter kernels of the first convolutional layer $C1$ appear very different compared to all other three measurement *CNNs* we trained. There is none of the oriented gradient like kernels, like they appear multiple times in the other measurement *CNNs*.

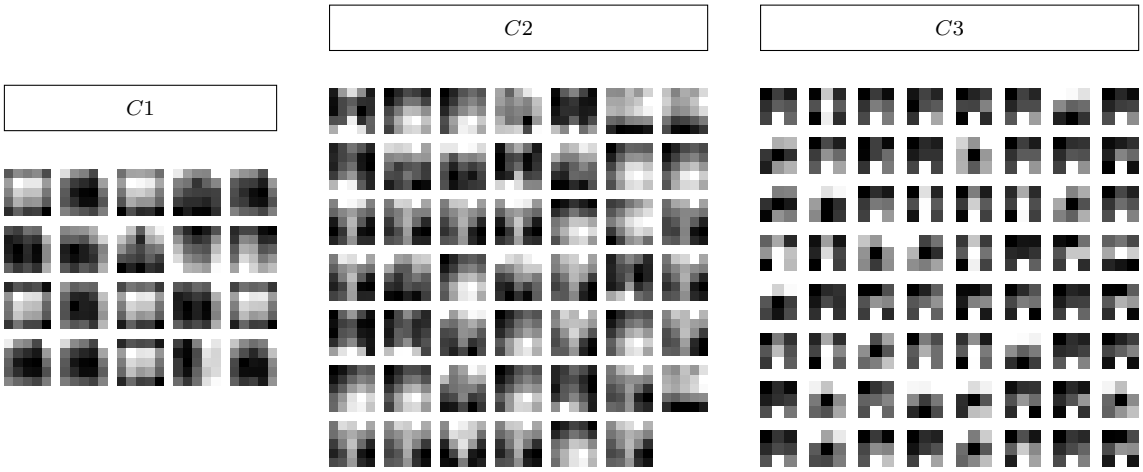

Figure 4.8: Visualization of the learned filters of the convolutional layers in the upper body length measurement *CNN*.

When looking at the accuracy statistics of the upper body lengths *CNN* in table 4.4, we can observe the errors we get on our validation data-set. It can be seen, that the *MAD* as well as the standard deviation do not exceed half a centimeter. When comparing the separate measurements extracted by the *CNN*, it can be seen that there is a significant difference in accuracy between the length measures and the circumference measure. Unfortunately, it is not easily possible to analyze the origin for this accuracy differences. We found two possible reasons for it. The first obvious reason could be, that it is more difficult for the *CNN* to obtain the neck circumference due to the missing information of the back-side of the neck, than to measure lengths using the front side only. Another possible reason comes from the observation of the non-gradient like filter kernels. Maybe the missing gradient-like filter kernels lead to less sensitivity on the necks curvature that could be crucial to obtain a more accurate neck circumference. Nevertheless, the measurement results are reasonably accurate for all three measurements.

	Max Abs. Diff $[cm]$	Mean Abs. Diff $[cm]$	Standard Deviation $[cm]$
Neck Circumference	2.53	0.41	0.49
Shoulder Distance	0.56	0.14	0.18
Neck Height	0.61	0.12	0.16

Table 4.4: Accuracy related values for each measurement taken from the upper body lengths measurement *CNN*. The presented values originate from feeding the *CNN* with the synthetic body region data, rather than the output of the body region *CNN*.

4.3.4 Leg Measures

In this section, we evaluate the *CNN* trained for extracting our five leg measurements. In figure 4.9, we show a graph of the loss on the training and validation data-set during training. The graph of the training data loss looks familiar when remembering those of our other measurement *CNNs*. The validation loss graph shows some interesting differences. In the beginning, it drops very fast and rises again to reach a local maximum at approximately 2000 iterations. Afterwards it continuously shrinks with progressing training iterations and seems to get a bit smoother. We assume that during the training process, the *CNN* gets a more generalized solution for the training data-set, which also brings benefits for the validation data-set.

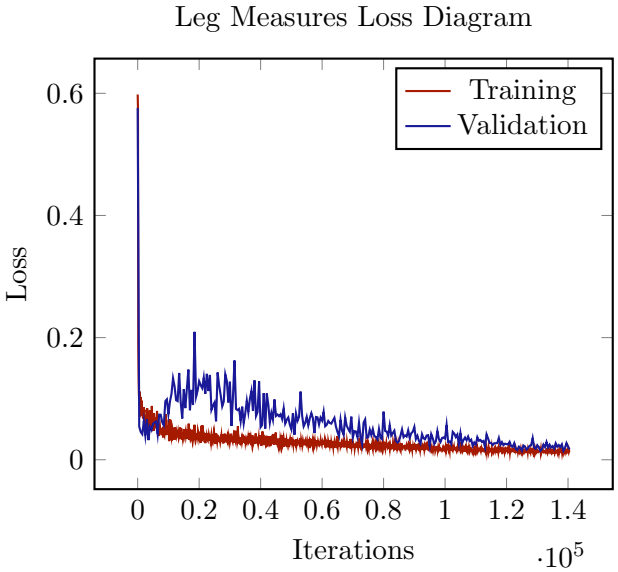

Figure 4.9: Diagram showing the loss of the *CNN* trained for extracting leg measures. The loss is displayed for the training as well as the validation data set.

When comparing the anatomy of the leg sub-region that we feed into this *CNN* to the one used for arm measurement, as it can be seen in figure 3.7, an important difference

is that the legs sub-region contains both legs, where the arm sub-region just contains a single arm. According to this, we are currently not able to obtain separate measurements for both individual legs, since our *CNN* is currently just trained to extract a single lower leg length, knee circumference etc. Since our training data is perfectly symmetric, we are not able to estimate the influence of asymmetric legs on our results.

In Figure 4.10, the filter kernels of the convolutional layers in our leg measurement *CNN* can be seen. When looking at the filter kernels in the first convolutional layer $C1$, it is obvious that the network learned to respond to vertical structures, which is what could be expected for the legs.

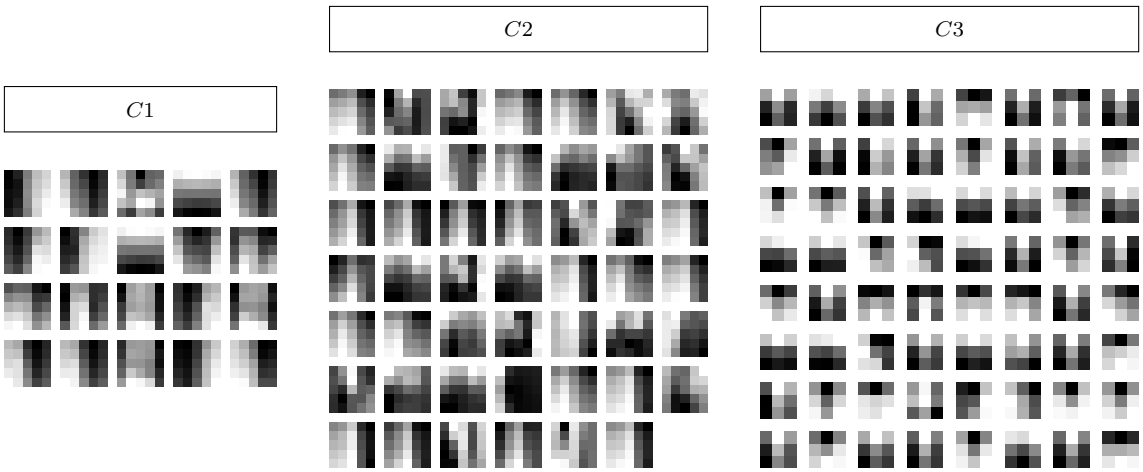

Figure 4.10: Visualization of the learned filters of the convolutional layers in the leg measurement *CNN*.

	Max Abs. Diff $[cm]$	Mean Abs. Diff $[cm]$	Standard Deviation $[cm]$
Upper Leg Length	1.67	0.29	0.37
Thigh Circumference	4.53	0.95	1.17
Knee Circumference	2.16	0.56	0.69
Lower Leg Length	1.30	0.23	0.31
Calf Circumference	1.97	0.60	0.72

Table 4.5: Accuracy related values for each measurement taken from the leg measurement *CNN*. The presented values originate from feeding the *CNN* with the synthetic body region data, rather than the output of the body region *CNN*.

Table 4.5 summarizes the accuracy of the leg measurement *CNN* using the *MAD*, the maximum absolute difference, as well as the standard deviation. These error measures where obtained by feeding the *CNN* with the ground-truth leg sub-regions, rather than

using the results of the body region detection *CNN*. It can be seen that the *MAD* as well as the standard deviation are below 1.2 centimeter for all leg measurements. When ignoring the thigh circumference, they even stay below one centimeter. It seems that, for this measurement *CNN*, the thigh circumference is the most challenging measurement.

4.4 Full Pipeline

In this section, we evaluate the end-to-end performance of the system. Table 4.6 shows the *MAD*, maximum absolute difference and the standard deviation of all anthropometric measurements our system obtains. These values result from comparing the ground-truth of our test-data with the output of our measurement pipeline. In contrast to the previously shown accuracy statistics tables, this table is a result of just feeding full-body depth-images into the pipeline and analyzing the measurement outputs. Interestingly, the results are almost the same as when just analyzing the results of the measurement *CNNs* using the ground-truth body regions. Some errors even decreased when evaluating the pipeline including the body region detection step. This lets us speculate that, even though we just used symmetric artificial training data, just with the ground-truth sub-regions for the measurement *CNN* training process, the measurement *CNNs* learned some kind of general solution. Obviously, for the worst case body region detection samples in the test data-set, the measurement *CNNs* did obtain approximately the same accuracy.

Additionally, we added the *MAD* obtained during an Anthropometric Survey (ANSUR) [5] of U.S. army soldiers. Unfortunately, they analyzed different anthropometric measurements, which means we can only compare a subset of measurements. However, we think the measurements we can compare give a good overview about our pipeline's measurement accuracy.

The *MAD* we compare with results from a survey where the anthropometric measurements of U.S. army soldiers were taken regularly. They were measured by different measuring persons trained on how to obtain these measurements. Therefore, we think there is no need to develop a system to be more accurate than this, since this should be the most accurate process for obtaining anthropometric measurements. The *MAD* evaluated in this *ANSUR*, is the average difference a single measuring person gets between two times measuring the same measurement of the same soldier. For our comparison, we use the average of these differences among different measurement persons, measuring several different soldiers each.

When comparing the *MAD* of the *ANSUR* with the *MAD* of our measurement pipeline, it can be seen that, for most anthropometric measurements, we are in a good range with our errors. Some of our mean errors are even lower than those of the *ANSUR*.

	Max Abs. Diff [cm]	MAD [cm]	Standard Deviation [cm]	ANSUR MAD [cm]
Upperarm Circumference	1.57	0.30	0.39	-
Upperarm Length	1.39	0.17	0.23	-
Lowerarm Length	0.93	0.16	0.22	-
Wrist Circumference	1.11	0.18	0.23	0.21
Upper Leg Length	1.82	0.31	0.40	-
Thigh Circumference	4.19	0.96	1.18	-
Knee Circumference	2.16	0.56	0.69	-
Lower Leg Length	1.42	0.23	0.31	0.20
Calf Circumference	2.10	0.61	0.73	0.13
Bust Circumference	4.15	0.63	0.84	1.14
Underbust Circumference	4.34	0.59	0.79	-
Waist Circumference	4.51	0.58	0.78	0.60
Hip Circumference	4.46	0.54	0.71	0.48
Neck Circumference	2.49	0.41	0.49	0.77
Shoulder Distance	0.54	0.14	0.18	0.51
Neck Height	0.65	0.12	0.16	-

Table 4.6: Accuracy related values for all extracted anthropometric measurements of our pipeline. The presented values originate from feeding the four measurement *CNNs* with the regions detected by our body region detection *CNN*. Additionally, we added the *MAD* of the *ANSUR* [5] of U.S. army soldiers. This *MAD* was obtained when taking an anthropometric measurement of a soldier twice by the same measuring person and comparing the outcomes.

4.4.1 Normalization

In this section, we evaluate the impact of the normalization strategy developed for the input and output of our measurement *CNNs* on the measurement accuracy. The details of our normalization strategy are described in section 3.5. To evaluate the normalization's impact on our measurement, we observe the measurement results of our system without the intelligence of our measurement *CNNs*. On the one hand, this gives an idea of how our normalization reduces the variance our measurement *CNNs* need to deal with, and, on the other hand, it gives an idea of how much intelligence the *CNNs* did achieve.

Since our normalization leads to a zero-mean output for our training data-set, we just replaced the output of our measurement *CNNs* by zeroes to do the evaluation in this section. Therefore, the results show how powerful the normalization of our *CNN* input and output values actually is. Table 4.7 shows the results of this experiment. Most of the maximum errors are way too large for a reasonable measurement system, but there are some measurements that are overall quite accurate. Obviously all non-circumference measurements can be predicted pretty good without the need of our measurement *CNNs*. This means that our body region detection in combination with our data normalization

strategy already represents a good measurement system for lengths.

The explanation we have for this interesting observation strongly relies on the basic idea of our data normalization strategy. All steps we do during normalization aim to get the absolute size information away from the measurement *CNNs*, so it can focus on the variance caused by the shape of the human body, rather than its size. Furthermore, this suggests that each length measurement is almost constant in length, related to its corresponding sub-region size. Otherwise, we would need more intelligence than our normalization to obtain accurate measurements of body lengths. For circumference measurements, we can see that the average error is in a reasonable range for some measurements, but the maximum error is obviously too large for all of the circumference measurements.

	Max Abs. Diff $[cm]$	MAD $[cm]$	Standard Deviation $[cm]$
Upperarm Circumference	15.94	3.35	4.19
Upperarm Length	3.28	0.47	0.60
Lowerarm Length	2.23	0.42	0.54
Wrist Circumference	8.37	1.78	2.15
Upper Leg Length	4.06	1.53	1.76
Thigh Circumference	25.65	5.00	6.46
Knee Circumference	9.15	2.56	3.15
Lower Leg Length	2.93	0.82	1.00
Calf Circumference	13.28	2.82	3.57
Bust Circumference	32.47	6.63	8.47
Underbust Circumference	35.10	6.81	8.62
Waist Circumference	49.02	7.34	10.16
Hip Circumference	35.43	6.35	8.47
Neck Circumference	12.81	2.23	2.94
Shoulder Distance	2.05	0.54	0.66
Neck Height	2.65	0.38	0.49

Table 4.7: Measurement accuracy statistics for data normalization evaluation. The measurement errors listed in this table are obtained when replacing the output of our measurement *CNNs* with zeroes only. This makes sense, since this is the mean output for our training data-set. Therefore, this table shows the power of our normalization strategy without the intelligence of our measurement *CNNs*.

Figure 4.11 visualizes the influence of our pipeline stages on the measurement accuracy we achieve. The bar graphs in this figure enable an easy observation of the relation between errors of different stages. The error measures we show in this figure are the maximum absolute errors in the dark colored bars, as well as the standard deviation of the errors in the light colored bars, based on our test data-set.

To get an understanding of how much variance is included in the test data samples, we show the errors originating from defining the mean test-data measurements as output

of the pipeline in the red bars. The green bars show the errors originating from defining the measurement *CNNs* outputs as zeroes and therefore evaluate the power of our body region detection paired with the data normalization strategy we developed. These values can also be obtained in table 4.7. Finally, the blue bars show the errors originating from the evaluation of our full measurement pipeline that can be found in table 4.6.

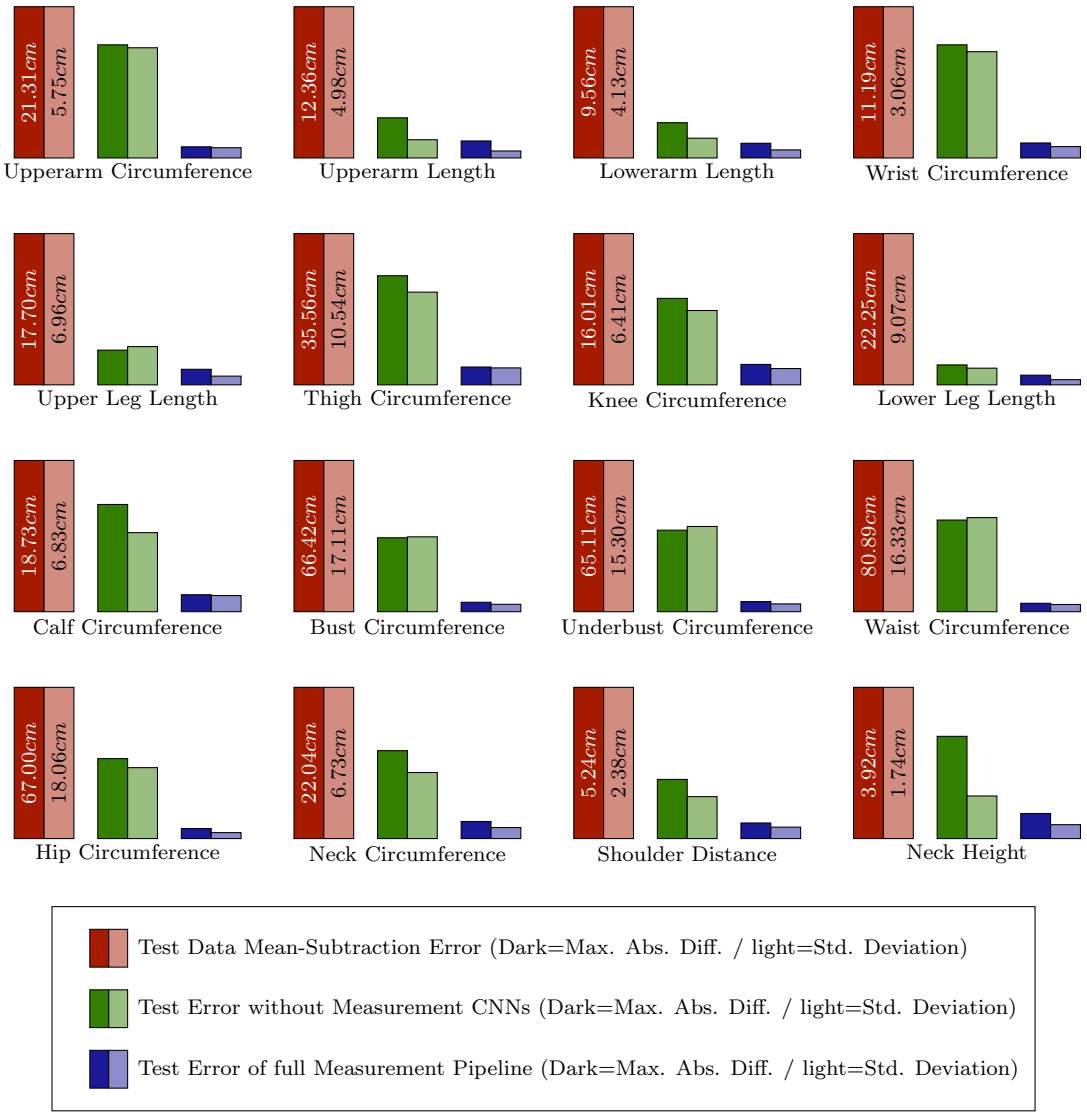

Figure 4.11: Maximum absolute differences and standard deviations of all anthropometric measurements of our pipeline shown for three cases. The red bars visualize errors resulting from just defining the mean measurement values as outputs. Green bars show the errors obtained when defining the measurement *CNNs* output zero but leaving normalization and body region detection active. Finally, the blue ones visualize the errors of our entire pipeline on the test data-set.

To understand the meaning of the bars height in figure 4.11, one must consider the linear factor between the mean subtraction error shown with the red bars and the error

represented by the other bars. More precisely, this means that for a single measurement we define the height of the dark colored bars proportional to its corresponding error measures. The same is true for the light colored bars. Therefore, this figure makes it easy to understand how each step reduces the error factor.

This bar diagram is a good visualization of the fact that the normalization already does a good job predicting most length measurements, but definitely not for circumference measurements. Additionally, the significant error reduction for circumference measurements when using the measurement *CNNs* shows that these *CNNs* are effectively capturing the body shape of the individual parts.

4.5 Limitations

In this section, we summarize the limitations of the measurement pipeline we developed. Most of them were already mentioned in the corresponding sections.

First of all, the most obvious limitation is that for now we did not make experiments using real-world data. For doing this, we would need ground-truth data for actual humans, consisting of depth images and the anthropometric measurements of them, conforming to the ones we obtain using *MakeHuman* [3]. Therefore, even if we had a small set of real-world samples, it would be difficult to evaluate the measurement accuracy. To have a chance to do this, we would need to do an *ANSUR* like in [5] and additionally capture depth images for all persons we do measurements on. We would need an extensive amount of these samples to train the *CNNs* of our measurement pipeline. This obviously is not practicable for the scope of a master thesis.

Since we neither use training- nor test-data from actual depth sensors, we currently do not provide solutions for problems with actual cameras. These problems would, for instance, be lens distortion, sensor noise and undefined depth pixel values. To make our approach usable for cameras with different intrinsic parameters, we would also need to do further research. The reason for this is that, currently, we do not consider the actual intrinsic parameters of the camera. We just consider them to be constant among all training and test samples.

When using actual depth sensors, we would also be facing the problem of segmenting the human body in the images to make our normalization strategy applicable. Additionally, we currently just trained and tested our pipeline with human models residing in the same pose. Therefore, we cannot expect our pipeline to be able to deal with significant pose variations. It is likely that we would need to train our pipeline with different poses, to have a chance to get accurate measurements in different poses. On the other hand, it should be possible to guide a person that wants to get measured which pose it should reside in.

5 Conclusion

Contents

5.1 Summary

During this project we developed an anthropometric measurement system, using a single depth-image of the human body's front, based on state-of-the-art technologies. We developed a toolbox for automatically creating training data consisting of depth-images of anatomically correct human bodies, including a set of anthropometric measurements for each body. This process is described in section 3.2.

Using this data creation toolbox, we were able to render enough depth-images of human bodies including their anthropometric measurements to train and test the Convolutional Neural Networks (CNNs) included in our measurement pipeline. The pipeline we developed can be described as a cascade of five separate *CNNs*. One *CNN* detects important body regions and four measurements *CNNs* get these regions as input to obtain the anthropometric measurements. The details about the measurement pipeline architecture can be found in section 3.1.

An important aspect to achieve good results using our approach is the data normalization strategy. This strategy is described in section 3.5. Using this normalization strategy, we were able to reduce the influence of the absolute body size in the data fed into our *CNNs*. Therefore, our measurement *CNNs* were able to focus on the shape rather than the absolute size of the human that is captured, which made it possible to achieve good results with all our measurements.

The evaluation of our results shows that our measurement results are comparable to those of an Anthropometric Survey (ANSUR) of U.S. army soldiers. In this *ANSUR*,

they evaluated the accuracy of trained personnel obtaining anthropometric measurements of soldiers. This comparison is contained in table 4.6. Additionally, we evaluated and presented the results of the single steps, like body region detection, normalization and measurement in this thesis.

To achieve these results, we based our work on existing research in the fields of camera geometry, human body anthropometry and machine learning. We were able to develop a pipeline, that computes accurate measurements from a single depth-image. Additionally our approach is computationally inexpensive when evaluating a single depth-image. A fast evaluation on a mobile device would definitely be possible.

5.2 Outlook

Since this pipeline is just based on synthetically generated data, there are many points that can be extended. When using real-world cameras, one would need to provide solutions for the common issues with depth-sensor images. These common problems would, for instance, be the noise of the sensor data, undefined pixel depths and incorrect depth values. In computer vision research, several projects have already explored these kinds of problems. Another challenge when using real-world cameras would be the fact that our pipeline strongly relies on constant intrinsic parameters of the camera.

For making it possible to train, or at least evaluate, this measurement pipeline with real-world data, many other challenges would arise. Probably the largest challenge would be to get anthropometric measurements of the captured humans, that are consistent with the ones used for training in terms of how and where they are measured.

Another obvious extension for our approach would be to support different poses, rather than relying on the person residing in our predefined pose. In order to achieve this, one would at first need training data in different poses. Additionally, it would likely be necessary to have a more flexible way of the body region detection that, for instance, allows regions to not just have pixel-axis parallel outlines.

A

List of Acronyms

ADALINE	Adaptive Linear Neuron
ANSUR	Anthropometric Survey
CNN	Convolutional Neural Network
MAD	Mean Absolute Difference
ReLU	Rectified Linear Unit
SSD	Sum of Squared Differences

Bibliography

[1] Allen, B., Curless, B., and Popović, Z. (2003). The space of human body shapes: reconstruction and parameterization from range scans. In *ACM transactions on graphics (TOG)*, volume 22, pages 587–594. ACM. (page 17)

[2] Anguelov, D., Srinivasan, P., Koller, D., Thrun, S., Rodgers, J., and Davis, J. (2005). Scape: shape completion and animation of people. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 408–416. ACM. (page 17)

[3] Bastioni, M., Re, S., and Misra, S. (2008). Ideas and methods for modeling 3d human figures: the principal algorithms used by makehuman and their implementation in a new approach to parametric modeling. In *Proceedings of the 1st Bangalore Annual Compute Conference*, page 10. ACM. (page 3, 21, 33, 51)

[4] Goodfellow, I., Courville, A., and Bengio, Y. (2015). Deep learning. book in preparation for mit press. (page 7)

[5] Gordon, C. C., Churchill, T., Clauser, C. E., Bradtmiller, B., and McConville, J. T. (1989). Anthropometric survey of us army personnel: methods and summary statistics 1988. Technical report, DTIC Document. (page 21, 35, 47, 48, 51)

[6] Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., and Wang, G. (2015). Recent advances in convolutional neural networks. *arXiv preprint arXiv:1512.07108*. (page 9)

[7] Hartley, R. and Zisserman, A. (2003). *Multiple view geometry in computer vision*. Cambridge university press. (page 5)

[8] Hirshberg, D. A., Loper, M., Rachlin, E., and Black, M. J. (2012). Coregistration: Simultaneous alignment and modeling of articulated 3d shape. In *European Conference on Computer Vision*, pages 242–255. Springer. (page 17)

[9] Hubel, D. H. and Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154. (page 7)

[10] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*. (page 10, 14, 36)

[11] Longpre, S. and Sohmshetty, A. (2016). Facial keypoint detection. (page 15, 27)

[12] McHenry, K. and Bajcsy, P. (2008). An overview of 3d data content, file formats and viewers. *National Center for Supercomputing Applications*, 1205. (page 22)

[13] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814. (page 9)

[14] Nickolls, J., Buck, I., Garland, M., and Skadron, K. (2008). Scalable parallel programming with cuda. *Queue*, 6(2):40–53. (page 14)

[15] Rojas, R. (2013). *Neural networks: a systematic introduction*. Springer Science & Business Media. (page 7)

[16] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1. (page 11)

[17] Shotton, J., Sharp, T., Kipman, A., Fitzgibbon, A., Finocchio, M., Blake, A., Cook, M., and Moore, R. (2013). Real-time human pose recognition in parts from single depth images. *Communications of the ACM*, 56(1):116–124. (page 17)

[18] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958. (page 14)

[19] Tsoli, A., Loper, M., and Black, M. J. (2014). Model-based anthropometry: Predicting measurements from 3d human scans in multiple poses. In *IEEE Winter Conference on Applications of Computer Vision*, pages 83–90. IEEE. (page 17)

[20] Vreeken, J. et al. (2002). Spiking neural networks, an introduction. *Institute for Information and Computing Sciences, Utrecht University Technical Report UU-CS-2003-008*. (page 8)

[21] Wang, R. and Qian, X. (2010). *OpenSceneGraph 3.0: Beginner's Guide*. Packt Publishing Ltd. (page 22)

[22] Weiss, A., Hirshberg, D., and Black, M. J. (2011). Home 3d body scans from noisy image and range data. In *2011 International Conference on Computer Vision*, pages 1951–1958. IEEE. (page 17)

[23] Widrow, B., Hoff, M. E., et al. (1960). Adaptive switching circuits. In *IRE WESCON convention record*, volume 4, pages 96–104. New York. (page 9)

[24] Zhang, Z. (2012). Microsoft kinect sensor and its effect. *IEEE multimedia*, 19(2):4–10. (page 6, 24)