FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Interdisciplinary Project Report

Development of a system that allows for the semantic segmentation of a 3D model of a human body into its constituent parts

Neha Das

FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Interdisciplinary Project Report

Development of a system that allows for the semantic segmentation of a 3D model of a human body into its constituent parts

Author:	Neha Das
Supervisor:	Prof. Dr. Nassir Navab
Advisor:	Dr. Federico Tombari
Date:	July 06, 2018

Acknowledgments

I would like to thank Dr. Federico Tombari, Helisa Dhamo and Keisuke Tateno for advice and assistance during the course of this project.

Abstract

Development of systems that are equipped to provide a view into a patient's body form an essential part of the technological advancements needed to improve medical diagnosis and reduce invasive surgery. While volumetric imaging of a patient's anatomy via Computed Tomography or Magnetic Resonance Imaging have achieved this goal to an extent, there is a growing need for combining such medical data via Augmented Reality Systems and Head Mounted Displays with the real-time view of the patient's form. Applications of such an implementation include surgery planning and inter-operative guidance systems.

An essential component of an implementation of the aforementioned Medical Augmented Reality (MAR) System, would be a module that could segment patient's form (obtained as a 3D Model) into its constituent body parts and identify the body part that corresponds to the volumetric medical data so that the 2 models may be registered.

The implementation of this segmentation module is the main goal of this Inter-Disciplinary Project. This is accomplished via first segmentating the depth maps obtained in the process of creating the patient's 3D model using a Deep Convolutional Neural Network and then combining the labelled images via KinectFusion [20] to obtain a segmented 3D Model.

Contents

Outline for the report

1. Introduction

This chapter outlines the need for an MAR system that can provide a view into the human body. It also describes the proposed implementation for such a system and the contribution of this body of work towards the overall goal.

2. Related Works

In this chapter, some of the related techniques that could be utilized for segmentation of 3D scans are described.

3. Methodology

This chapter describes the methodology used in this body of work for segmentation of 3D scans. It discusses the theoretical background of these techniques, describes the actual architecture used and elaborates on some of the implementation details.

4. Dataset

This chapter discusses the creation and the features of the data set used in the methodology described in above section.

5. Experiments and Results

In this chapter, the results from the above implementation are described.

6. Future Work

Some ideas about future work in context of the solution presented in this report is laid out in this chapter.

7. Conclusion

Finally, this chapter summarizes the work done and presents the major conclusions that could be derived from it.

Appendix

The references and sources for the information that were presented in this report are shared in this section in addition to the links to the source code

1. Introduction

Development of systems that are equipped to provide a view into a patient's body form an essential part of the technological advancements needed to improve medical diagnosis and reduce invasive surgery. While volumetric imaging of a patient's anatomy via Computed Tomography or Magnetic Resonance Imaging have achieved this goal to an extent, there is a growing need for combining such medical data via Augmented Reality Systems and Head Mounted Displays with the real-time view of the patient's form. Applications of such an implementation include surgery planning and inter-operative guidance systems.

The result from this Inter-Disciplinary Project is an essential component of such an implementation. In the following subsections, we will briefly introduce the overall implementation and the contributions of this work.

1.1. Overall Implementation

The proposed implementation (Figure 1.1) has several major constituents:

- First, a 3D scan of the patient body is obtained via KinectFusion [20].
- At the same time, a texture mapping of the 3D scan is obtained.
- Simultaneously, each of the depth images acquired during the KinectFusion run are segmented using a Deep Convolutional Neural Network for semantic segmentation. These labelled images are then fed back to the KinectFusion algorithm so that they may be combined to form a segmented 3D Model.
- The body part corresponding to the volumetric medical data is then extracted using the segmentation labels and registered with the medical data, and finally the volumetric image is aligned with the patient's 3D model using the

transform parameters ($^{CT}T_{Kinect}$) estimated during the registration step.

1.2. Contributions

The major contribution of this work towards the overall implementation is the development of the segmentation module that can label the constituent parts in a 3D body model. This is performed via segmenting the depth images obtained during the KinectFusion run using a Deep Neural Network model for segmentation. Each labelled image is then fed back to a modified version of the KinectFusion algorithm and combined to form the segmented 3D model of the body.

This work thus constitutes of the following:

- Investigation of fast deep neural networks for semantic segmentation.
- Based on the investigation from the first step, development of a semantic segmentation architecture with deep convolutional and deconvolutional layers.
- Investigation of existing datasets/ creation of a dataset for training the above model.
- Implementation of the inference pipeline in C++ for integration with the overall implementation.
- Modification of the KinectFusion algorithm implentation [13] to suit the requirements and integration of the segmentation inference module with it.

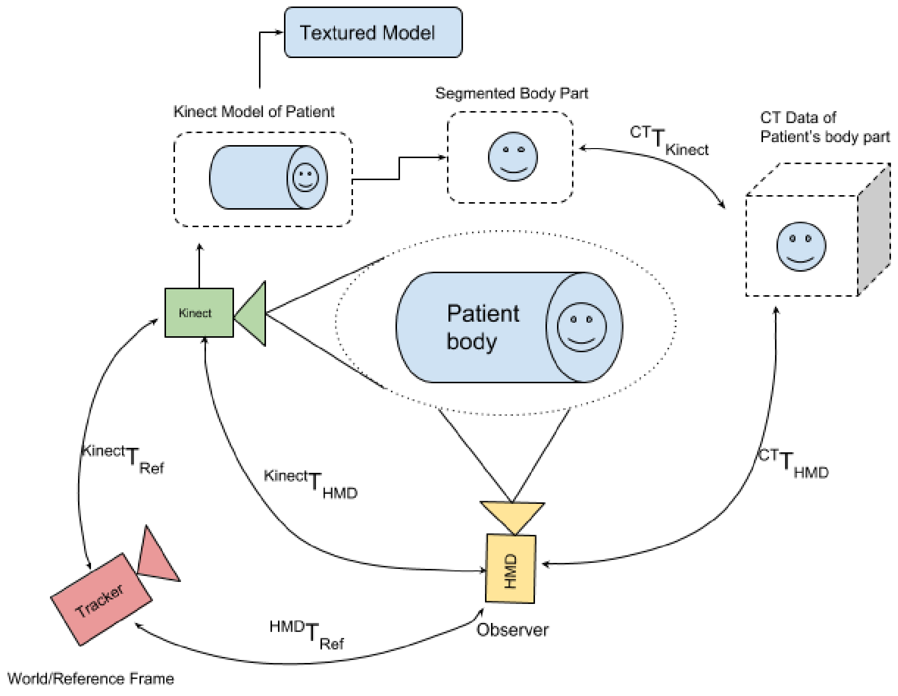


Figure 1.1.: Proposed System - It consists of four parts. First, a 3D scan of the patient is obtained via Kinect Fusion. Then, the 3D scan is segmented into it's constituent body parts and the point cloud data for the part that is of interest is extracted. Simultaneously the texture mapping for the 3D scan is obtained. The final step is proper augmentation of this scan data on to the patient's body from a user's view who will be wearing a HMD. To do so, the transformation between the Kinect scan and the medical 3D scan is obtained via registering both the point clouds with each other. For proper augmentation of medical data, a transformation $^{CT}T_{HMD}$ that can transform points from CT frame of reference to the HMD frame of reference i.e viewer's reference frame is needed. This is done using the transformations $^{CT}T_{Kinect}$, $^{Kinect}T_{HMD}$. Image from [14]

2. Related Works

Previous works have approached the problem of segmenting 3D scans of humans into constituent parts from many different directions.

Many techniques rely on estimates of human body proportions. For instance, [28] introduces a method to segment 3D point clouds based on an amalgamation of heuristic approaches, each of which determine a key location in the human body. A similar approach is followed by [3], where 15 key locations such as head, torso and so on are estimated using PCA (Principal Component Analysis; [23], [15]) of the 3D point cloud and heuristics that describe the position of other body parts relative to the head and torso. However, hand-crafted guesstimates such as those used in these techniques often yield sub-optimal results since they may not be as generalized with respect to human pose or body shape.

In contrast, there exist supervised learning techniques like SCAPE ([2]), that result in creation of a deformable model for 3D scans by training its parameters on a dataset of registered 3D scans with multipe shapes and poses. This model can then be fit to any incoming 3D scan during inference time, even if it is a partial scan. While this method may not yield the exact segmentation map of the point cloud, obtaining a segmentation shouldn't be hard if the corresponding template has been segmented. However, these methods often need some kind of manual initialization where a sparse number of markers have to be identified on the input scan prior to the registration process.

Accurate Non-rigid registration techniques may also yield a similar solution. They can be used to register an input 3D scan with a template. Recent advances in the correspondence matching problem via deep learning [27] have increased the accuracy of this technique.

The method proposed in this body of work uses a 3D reconstruction technique such as KinectFusion and in addition to fusing the depth maps for the human body from different viewpoints, also segments the depth images to create segmentation maps and fuses them. For the reconstruction of the segmented 3D Model to be real-time, the segmentation output for the depth images have to be fast and accurate.

A number of methods employ supervised learning techniques for this purpose. The pioneering paper by Shotton et al. [26] proposed the use of Deep Random Forests to perform a per pixel classification of the input depth map. Other techniques ([22], [25]) have improved upon these results by using Deep Convolutional Neural Networks and Deconvolutional Layers for semantic segmentation of body parts from depth images.

This project will use a U-Net [24] like architecture for this purpose. U-Net has been known to have a small inference time and yield accurate results for semantic segmentation tasks from RGB images. To reduce the inference time, we also investigate the effect on compute time on substitution of the convolutional units in the encoder by depthwise separable convolutions, which can theoretically reduce the computation time by 8-9 times.

3. Methodology

As indicated in the previous chapters, the adopted methodology for segmentation of 3D scans follows two steps:

- Segmentation of the incoming depth maps from a depth camera.
- Fusion of the segmented depth maps in real time to create a segmented 3D scan.

In this chapter, we will explore the background and technical details of these two steps.

3.1. Depth Map Segmentation

Semantic segmentation is an image processing task which assigns a segmentation label to each pixel in an image. This task has wide ranging applications across many industries such as automotive, manufacturing and healthcare.

Before the advent of deep learning, automatic segmentation was usually performed by computer vision based methods such as region-growing/merging techniques ([1], [6]), implementations for solving Mumford Shah [19] functionals like Chan-Vese Level Set Method [7], Diffusion Snakes [12] and others. In the class of Machine Learning methods, Random Forest Classifiers [26] were popular.

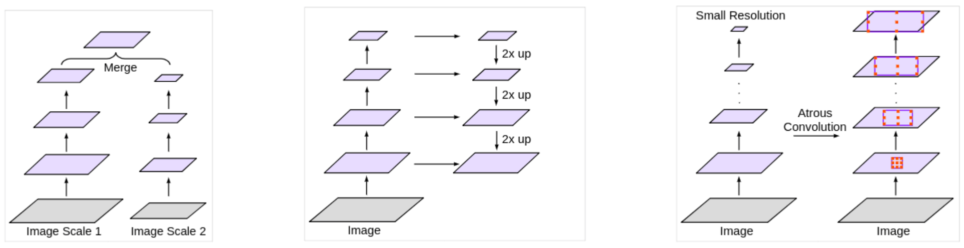
There exist several architectural choices for building a deep learning model for semantic segmentation [9]:

- **Image Pyramid**
  In these models, the input is scaled to multiple resolutions and Deep Convolutional Neiral Networks are applied to extract features ranging from aggregate

to detailed and merge them eventually to get a contextual module from which the segmentation map is derived. However, these models are not quite scalable, as they become huge for deep networks.

- **Encoder-Decoder Models**
  These models have a cascade of convolutional layers with decreasing output resolutions. Typically, strides larger than one are used intermittently in the cascade to decrease the resolution by a large factor and aggregate information. The convolutional cascade is followed by a decoder structure composed of de-convolutional or transposed convolutional layers that recover spatial information again. Skip connections may also be added between the encoder and decoder units in order to assist the decoder units in learning the spatial information lost through the pooling operations. Fully Convolutional Networks [18] presented the first architecture in this category. U-Net [24] is another popular choice from this category and will be the choice of architecture in this project.

- **Dilated/Atrous Convolutions**
  While pooling/multi-strided layers help aggregate information across the spatial dimensions of the image, they hamper the semantic segmentation task since they lose spatial details with each pooling/multi-strided step. Atrous/Dilated Convolutions instead opt for a dilated kernel and preserve the density of the feature layer while adopting a wider field of view for the kernels used. This helps prevent the loss of spatial details as the architecture flow approaches the output. Architectures proposed by [30] and [8] fall under this class.



(a) (b) (c)

Figure 3.1.: (a) Image Pyramid (b) Encoder-Decoder Architecture (c) Atrous Convolutions. All images are taken from [9]

While Dilated convolutions may seem like the best choice for segmentation tasks, an architecture utilizing them such as [8] are not without certain drawbacks [17]. The

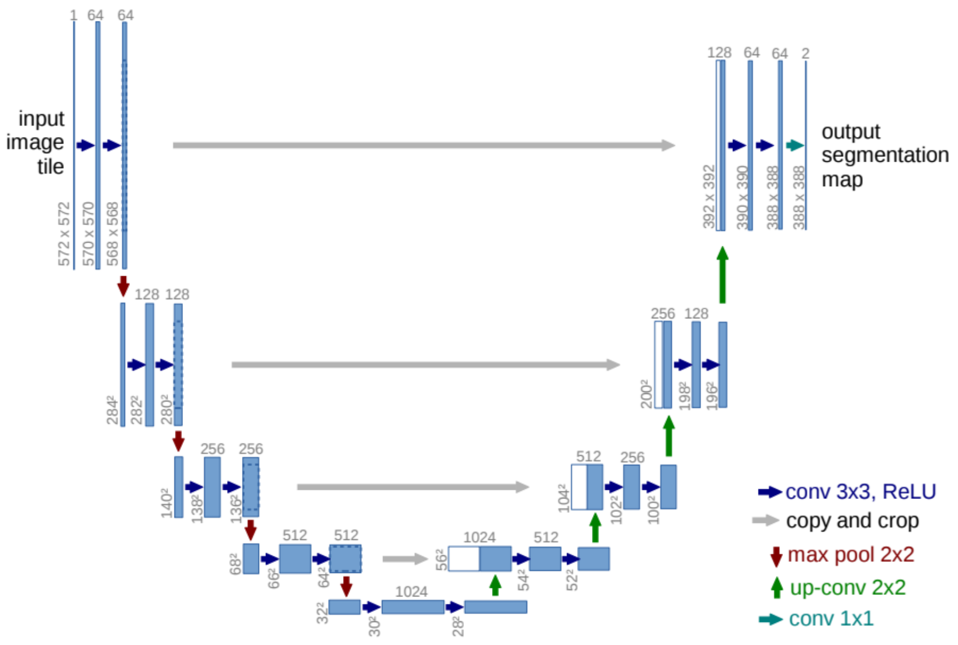


Figure 3.2.: U-net architecture [24] (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations

first is that they have a large memory footprint and have to be computed at each step for large resolution maps (since we do away with most pooling layers) and secondly, the architectures utilizing these constructs often have to output segmentation maps that are 1/8th the size of the original input image.

Recently, RefineNet [17], an encoder-decoder module has been introduced to counter the usual fallacies of encoder-decoder models through Refine-net modules (See cited paper). Being an encoder-decoder network, they also donot suffer from the limitations of dilated convolutions. However, as stated before, we will be adopting a U-Net architecture since our goal is to be fast as well accurate while performing our segmentation task and Refine-net modules add to the computational weight.

3.1.1. U-Net

U-Net [24] adapts the semantic segmentation architecture from the paper Fully Convolutional Networks [18] to the field of biomedical images. It has an encoder-decoder architecture, where the encoder contains a cascade of convolutional blocks. Each block contains multiple convolutional layers and operates on different resolutions of the image. A new resolution (reduced by a factor of two) is obtained by keeping the stride of the first convolutional layer of the block as two. The decoder consists of simiarly stacked deconvolutional blocks, where the first layer is transposed-convolutional layer with stride two and the next couple of layers in the block are convolutional layers. Each convolutional block has a corresponding deconvolutional block and the output from each convolutional block is concatenated with the output from the first layer of each deconvolutional block to form skip-connections. A layout of the architecture can be seen in Figure 3.2.

3.1.2. MobileNet and Depthwise Separable Convolution

MobileNets were introduced in [16] by Howard et al. from Google Inc as a solution to the need for smaller and faster convolutional networks that can run on a slower processing unit (such as a mobile phone) and yet yield comparable results. Their prime feature was the incorporation of depthwise separable convolutions that have lower latency and size than regular convolutional units while yielding comparable numbers for accuracy. Additionally, they also featured two hyperparameters that help compress the network size further in a flexible manner. These constructs are discussed briefly in the subsections below:

Depthwise Separable Convolution

Standard Convolution Operations (with a Kernel size of $D_K \times D_K \times M \times N$ and output size $D_F \times D_F \times N$) in MobileNet are replaced by a factorized operation called the Depthwise Separable Convolution. The input is first convolved depthwise with a kernel of dimensions $D_K \times D_K \times 1$ and followed by a pointwise convolution with a kernel of size $1 \times 1 \times N$. The kernels used are illustrated in Figure 3.3

This reduces the computation cost of the convolution operation from

$$D_K \cdot D_k \cdot M \cdot N \cdot D_F \cdot D_F \tag{3.1}$$

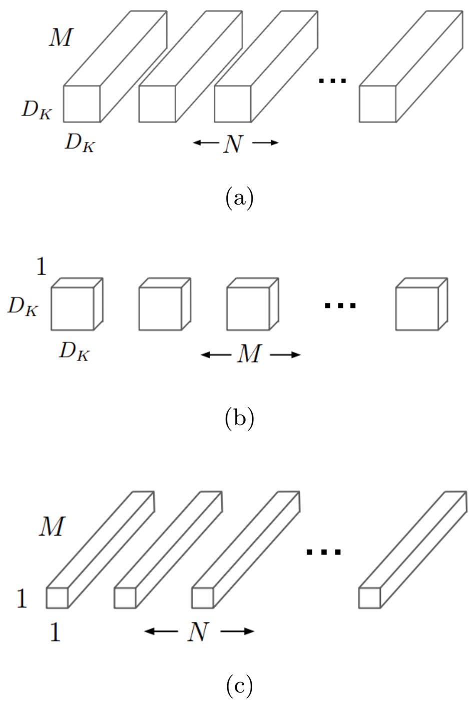


Figure 3.3.: Kernel representation for (a) Standard Convolution, (b) Depthwise Convolution Operation and (c) Pointwise Convolution Operation. (a) is replaced by (b) and (c) in Mobilenets. Figures from [16].

to

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F \tag{3.2}$$

The full operation set for regular convolutional layers vs depthwise-separable convolutional layers is illustrated in Figure 3.4

As evident from Figure 3.4, the depthwise separable convolutions in the Mobilenet architecture have twice the number of non-linear activations and batch-normalization operations as compared to their regular convolutional counter-part. In the experiments performed as a part of this work, it was observed that these additional batch-norm operations almost nullified the computational benefits reaped via the structure of depthwise separable convolutions. As a consequence, the interme-

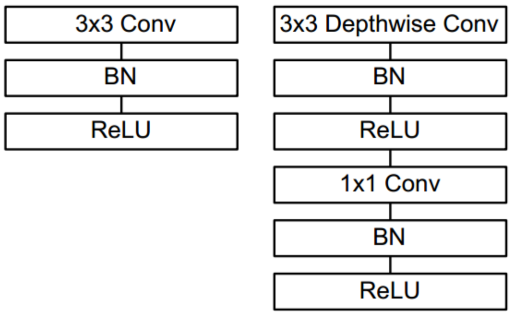


Figure 3.4.: Regular Convolutions vs Depthwise Separable Convolutions with BatchNorm and ReLu Activation. In MobileNets, the depthwise separable convolutions contain an intermediate non-linear activation and a batch normalization layer in addition to a similar unit at the end of the operation. Figure from [16]

diate layers were removed from our formulation of the full depthwise separable convolutional layer set. The paper on Xception ([10]) provides some empirical evidence to support the theory that removing these intermediate layers will not adverserly affect the performance of the network. Infact, it shows that the network converges faster and has more accuracy. The paper speculates that when the non-linearity is applied on an convolution of low depth (in this case, of size one), it may result in the loss of information which can lead to a decrease in accuracy.

In addition to the depthwise separable convolutions, the width multiplier and resolution multiplier hyperparameters can be adjusted in order to further compress the network at the expense of higher accuracy:

Width Multiplier

Width Multiplier (α) is a hyperparameter for reducing the number of the input and output filters, thus making them thinner. [16] shows empirically that a decrease in the value of α leads to a smooth drop in the accuracy of the model until α reaches 0.25 and the architecture is made too small.

Resolution Multiplier

Resolution Multiplier (ρ) is hyperparameter that allows one to change the output resolution by a factor of ρ^2. This parameter actually corresponds to the resolution of the input image. Thus it can be implicitly set by setting the resolution of the input image.

3.1.3. Transposed Convolution vs Upsampling with Convolution

The deconvolution operation in context of convolutional Neural Networks is not actually the deconvolution operation defined in a mathematical sense. In the context of images and CNNs, it refers to an operation that can retrieve the original image resolution - i.e, before a convolution operation was performed. Such an operation is needed in a U-Net based semantic segmentation network (see Figure 3.2), like the one proposed by this paper. This is usually done via Transposed Convolution Operation.

Transposed Convolution is a parametrized operation that can learn the correct upsampling of an image. In Tensorflow, it corresponds to the transpose or the gradient of a convolution operation with respect to its input[1]. The illustration in Figure 3.5 demonstrates the intuition behind this operation.

For deconvolving to a resolution that is twice the size of the input, usually a transposed convolution with stride two is used. However, this may result in the appearance of checkerboard like artifacts if the kernel size is not divisible by the stride length [21]. These effects may be countered by either using a stride length that can fully divide the kernel length - for instance, with a stride length of one and an appropriate kernel size, any desired resolution can be achieved. However, using a stride length of one may not be feasible as it nearly doubles the number of network parameters. Another option is to use an upsampling filter (devoid of learnable weights) such as the bilinear interpolation filter or the nearest neighbor filter followed by convolutions of stride one. A comparison between the model outputs from the different constructs is shown in Figure 3.6.

We compare results and runtime for both networks with two-strided transposed convolutions and upsampling with convolutions.

[1]See tf.nn.conv2d_transpose: https://www.tensorflow.org/api$_docs/python/tf/nn/conv2d\_transpose$

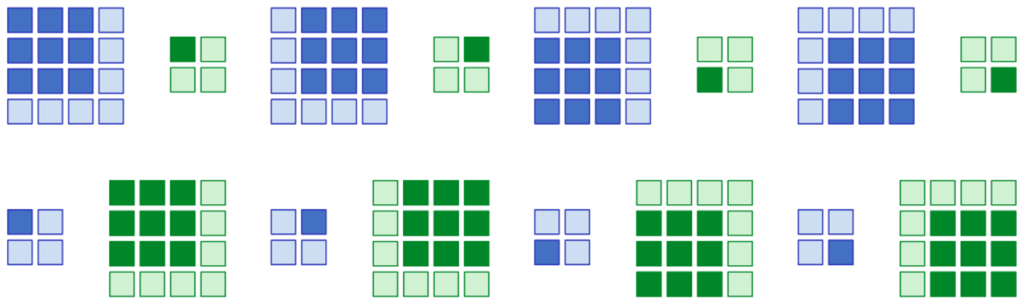


Figure 3.5.: The top row shows the effect of a convolution operator with a 3x3 kernel on an input of dimension 4x4. The bottom row shows the effect of a transposed convolution operator with a 3x3 kernel on an input of dimension 2x2. The blue grids represent the input, whereas the green grids represent the output. In Tensorflow, the transposed convolution operator exactly equivalent to the backpropagation operation on a convolutional layer such that the input to transposed convolution corresponds to the gradient of the output layer, and the output from the transposed convolution corresponds to gradient of the input layer of the convolution operation.

3.1.4. Proposed Architecture for Depth Segmentation

The task of depth segmentation of human bodies is evaluated on the following architectures in this body of work:

- **unet**
  A modified version of U-Net adapted to the input image resolution used in this project. This model has 4 convolutional and 4 deconvolutional blocks.

- **unet-mobile**
  This model is the same as unet except, instead of standard convolutional layers, separable convolutional layers have been used. The width multiplier and the resolution multipliers are set to 1.0 for the separable convolutions.

Additionally, we reduce the resolution of the images before we feed them into the network from 640x480 to 160x120. This is done to speed the network up. During prediction time, the nearest neighbor interpolater is used to upsample the output labels to 640x480.

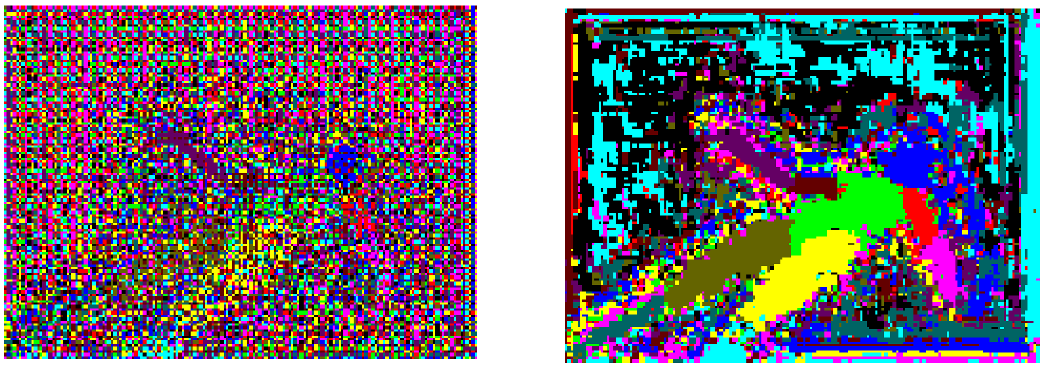

(a) (b)

Figure 3.6.: Predictions after training a model with (a) 2 strided transposed convolutions, and (b) upsampling + convolution, on a single image for 20 iterations

3.1.5. Implementation Details

The training and evaluation pipelines as well as the network model have been implemented in Python v3.5. Both *unet* and *unet-mobile* were implemented in Tensorflow v1.7 and utilize the Tensorflow Slim Framework, which encapsulates a lot of low level details while defining the layers and makes the implementation more readable and maintainable.

The data-pipeline makes use of queuing with multi-threading via Tensorflow Queuerunners to deliver batches of data to the training and evaluation pipelines. This allows for continuous retrieval of batches which results in a smooth operation.

Once the network is trained to convergence, the training graph is frozen along with the final weights to a Tensorflow Protocol Buffer format.

The final inference while retrieving depth maps from the Kinect sensor is performed via loading the frozen graph in the Tensorflow C++ API.

3.1.6. Training Details

Both models are trained via the Adam Optimizer with an initial learning rate of $1e-4$ and a decay rates of 0.9 and 0.999. Batches of 50 are used for training.

The batch normalization decay rate is kept at 0.9 and the dropout probability for retaining the node is kept at 0.999.

3.2. Fused 3D Segmented Model

The previous section describes the deep neural network that outputs a segmented label map for every depth map that is inputted. These label maps are used in conjunction with their corresponding depth maps to create a 3D model using KinectFusion [20], which is described below:

3.2.1. KinectFusion

KinectFusion is a realtime Simultaneous Localization and Mapping (SLAM) algorithm that can fuse the stream of depth maps obtained from a low-cost sensor such as Kinect into a 3D reconstruction of the scene which is updated using a Truncated Signed Distance Function (TSDF) representation and rendered as a mesh via raycasting. It comprises of the following components:

- **Surface Measurement**
  The incoming depth maps are processed via a bilateral filter to produce discontinuity preserving depth maps with reduced noise. These modified depth maps are then used to produce vertex maps by back projecting the pixels into the sensor frame of reference. Normal maps are also produced by taking the cross product of neighboring vertices in the vertex maps.

- **Pose estimation**
  With the processing of each depth map, the sensor pose relative to the global map frame is estimated. Here, global map refers to the fused TSDF representation maintained and modified with each cycle of the KinectFusion algorithm. Pose estimation is done via point-to-plane ICP (Iterative Closest Point) [4].

- **Surface Reconstruction**
  Via the signed distance function, a 3D space can be represented in terms of their closeness to the surface enclosed by it. The space inside the surface is increasingly negative as it moves away from the surface, while the space outside has increasing positive values. The function value for the surface points is 0.

TSDF, on the other hand, places such restrictions only on the space close to the surface (specified by a thresholding value). For points beyond this range, the TSDF value is truncated to a maximum μ or a minimum, $-\mu$. Using the sensor positions obtained from the pose estimation step and the filtered depth values, TSDF for each cycle is calculated and then fused with the global TSDF using a weighted average.

- **Surface Prediction**
  The dense representation of the global map is finally rendered via per pixel raycasting in the frame of reference provided by the current estimate of the sensor's position.

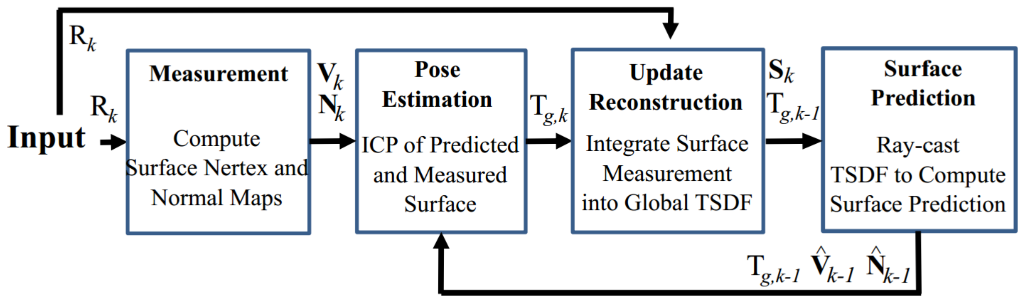

Input
R_k
R_k
Measurement
Compute Surface Nertex and Normal Maps
V_k N_k
Pose Estimation
ICP of Predicted and Measured Surface
T_g,k
Update Reconstruction
Integrate Surface Measurement into Global TSDF
S_k T_g,k-1
Surface Prediction
Ray-cast TSDF to Compute Surface Prediction
T_g,k-1 V̂_k-1 N̂_k-1

Figure 3.7.: Flowchart describing the KinectFusion algorithm. Figure taken from [20]

3.2.2. Combining Segmented Maps

As a part of the overall implementation, KinectFusion was implemented by Christian Diller [13]. This implementation also included the fusion of the RGB images obtained in conjunction with the depth images by kinect so that a colored 3D model could be obtained. The coloration for each 3D point in the global intrinsic representation (TSDF) is obtained via a weighted average of all the RGB values of the pixels that were back-projected to that 3D point.

A similar methodology is applied while combining the color-coded segmented depth maps - Each body part is represented by an RGB value. Each vertex of the combined 3D model is colored by a weighted average of the colors for corresponding pixels in each of the depth maps that influences that vertex. The influence of the background however (RGB value = 0,0,0) is not taken into account while calculating the average.

3.2.3. Extraction of the Body Part

Volumetric Medical Scans usually contain some meta-data that indicate the body part they belong to. This information can allow the system to automatically select a body part or a combination of body parts. The 3D point cloud of the body part is selected in the system by selecting all the vertices in the 3D model that correspond to a particular range of colors (specified in Chapter 5). A cleaner solution would have been to select a mode value for each of the vertices, but wasn't implementable due to GPU memory constraints.

4. Dataset

For the purpose of training and evaluating the models described in the previous chapter, we create a dataset of depth images of human figures alongside their segmentation labels. This was mainly motivated by the lack of publicly available datasets suited to our needs.

We use registered/aligned 3D scans derived from the CAESAR dataset provided by [29] for creating our data corpus. There are over 3000 scans in the dataset evenly divided into male and female subjects. For the purpose of this work, each of these 3D scans are factorized into 96 depth images taken from different angles, focal lengths and positions using the open source software Blender [5]. This ensures that the dataset is well augmented and the model is equipped to deal with human figures positioned randomly in the frame of camera and also any occluded body parts, if applicable. The pseudocode in Listing 4.1 describes the choice of camera angles, position and focal length for generation of these 96 depth images.

In addition to the depth images, RGB images representing the label maps for each depth image are also generated. We accomplish this via the following steps:

- **Color all the 3D meshes** In step 1, we paint the 3D meshes so that the RGB value for each vertex is representative of the body part it belongs to. For this we do the following:
  - Transform all the 3D meshes obtained from [29] from OBJ file format to PLY file format, since OBJ files cannot store vertex color information.
  - Pick a single 3D mesh and use the vertex paint tool to color all the vertices in accordance with the body parts they belong to in MeshLab [11].
  - Since all the meshes are registered or aligned, vertex colors are easily transferable from one mesh to corresponding vertices in other meshes.

- **Extract labelled images** RGB images with per pixel label are extracted from the colored 3D scans using the same combinations of camera positions, angles and focal lengths as for the depth images via blender [5]

The above steps are illustrated in the following figure:

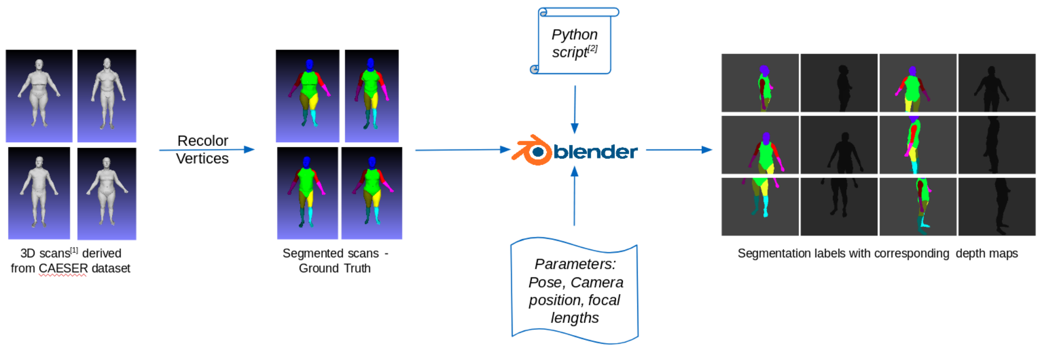

3D scans[1] derived from CAESER dataset
Recolor Vertices
Segmented scans - Ground Truth
Python script[2]
blender
Parameters: Pose, Camera position, focal lengths
Segmentation labels with corresponding depth maps

Figure 4.1.: Caption

In the current implementation, a total of 11 segmentation classes are defined. These along with their RGB representatives for labels are as follows:

1. Background - (0,0,0)
2. Torso - (0,255,0)
3. Head - (0,0,255)
4. Upper Left Arm - (255,0,0)
5. Upper Right Arm - (100,0,0)
6. Lower Left Arm - (255,0,255)
7. Lower Right Arm - (100,0,100)
8. Upper Left Leg - (255,255,0)
9. Upper Right Leg - (100,100,0)
10. Lower Left Leg - (0,255,255)
11. Lower Right Leg - (0,100,100)

Some examples of the depth and labelled images can be seen in the following figure:

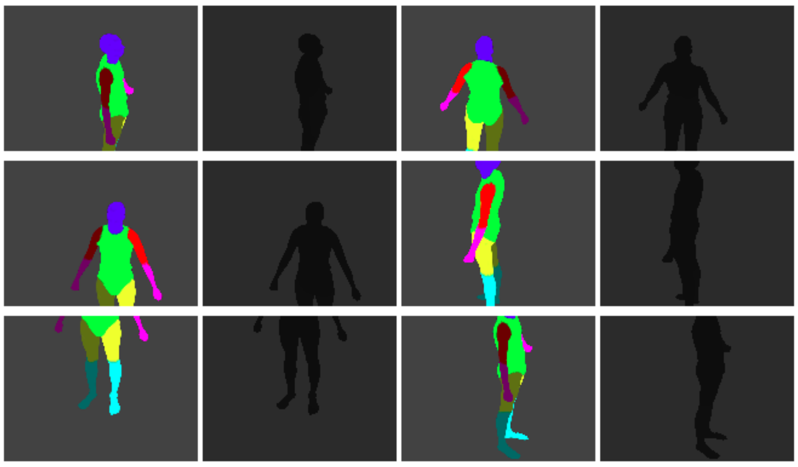


Figure 4.2.: Label Maps and corresponding Depth Maps

```
# number of camera angles around pos_empty
NUMBER_OF_VIEWS = 4
# the offset for first camera angle depth rendering for a ply
OFFSET_DEGREE_FROM_START = 1
# scale of the 3D scan. We scale down the 3D scan in the blender
# environment to keep the depth values close to the real world
OBJECT_SCALE = 0.1
# determines the range of the random values from which pos_empty,
# and camera position are picked
RANDOM_SHIFT_MULTIPLIER = 0.1
# pos_empty is th 3D position towards
# which the camera is directed
# pos_empty x positon : fixed position
# and min & max for random choices
EMPTY_MIN_X = -0.1
EMPTY_MAX_X = 0.1
# pos_empty y positon: fixed position
# and min & max for random choices
EMPTY_MIN_Y = -0.05
EMPTY_MAX_Y = 0.05
# pos_empty z positon: fixed position
# and min & max for random choices
EMPTY_MIN_Z = -0.1
EMPTY_MAX_Z = 0.05
# camera position: fixed position and
# min & max for random choices
CAMERA_ROTATION_AXIS = 'Z'
COORD_FIX = 0.7
```

```
COORD_MIN = 0.5
COORD_MAX = 1.0

# File containing the 3d scan in .ply format
fileName = '3d_Scan.ply'

# the set of camera positions to iterate over. 1 fixed camera
# position is chosen along with a random position from a
# uniform distribution in the range [COORD_MIN, COORD_MAX)
cam_pos = create_camera_positions(COORD_MIN, COORD_MAX, COORD_FIX)

# the set of pos_empty to iterate over.. 1 set of fixed
# x,y,z positions (central) for pos_empty is chosen along
# with 1 random position for x, 1 random position for y, and
# 2 random positions from z (upper and lower halves of body)
empty_pos = create_empty_positions(EMPTY_MIN_X, EMPTY_MAX_X,
                                   EMPTY_MIN_Y, EMPTY_MAX_Y,
                                   EMPTY_MIN_Z, EMPTY_MAX_Z)

for itr_pos in range(0, len(cam_pos)):

    # iterate over all the camera positions
    position = cam_pos[itr_pos]

    # Camera position is set to chosen position
    set_camera_position(position)
    focal_lengths = get_camera_focal_lengths(CAMERA_FOCAL_FILE)
    for itr_focals in range(0, len(focal_lengths)):

        # iterate over all the camera focal lengths
        focal = focal_lengths[itr_focals]

        # Camera focal length is set to chosen focal length
        set_camera_focal_length(focal)
        for itr_empty_pos in range(len(empty_pos)):

            # iterate over all the camera focus points
            pos_empty = empty_pos[itr_empty_pos]
            if validate_file(fileName, VALID_FILE_EXTENSION):
                clear_scene()
                file_path = os.path.join(INPUT_DATA_DIR, fileName)
                import_file(file_path)
                index = (itr_pos * len(focal_lengths) + itr_focals) *
    len(empty_pos) + itr_empty_pos
                outFileBase = 'human_' + str(itr_file)

                # render depth map for "NUMBER_OF_VIEWS" equidistant
                # angles, starting with "offset_start_degree", with
                # the camera directed at position "pos_empty"
                rotate_empty_and_render(fileName, outFileBase, index *
    NUMBER_OF_VIEWS, offset_degree_start, pos_empty)
    itr_focals_start = 0
itr_pos_start = 0

# the starting offset for camera angles is different for each 3D scan
# in order to cover all 360 degrees
offset_degree_start = offset_degree_start + OFFSET_DEGREE_FROM_START
```

Listing 4.1: Pseudo-code for generating depth images from 3D scan

5. Experiments and Results

5.1. Depth Image Segmentation

We train both the *unet* and *unet-mobile* model ensembles on 23040 depth-label pairs derived from a little less than 300 different scans provided by [29] for 100 epochs with a batch size of 50. Training loss graphs can be seen in Figure 5.1

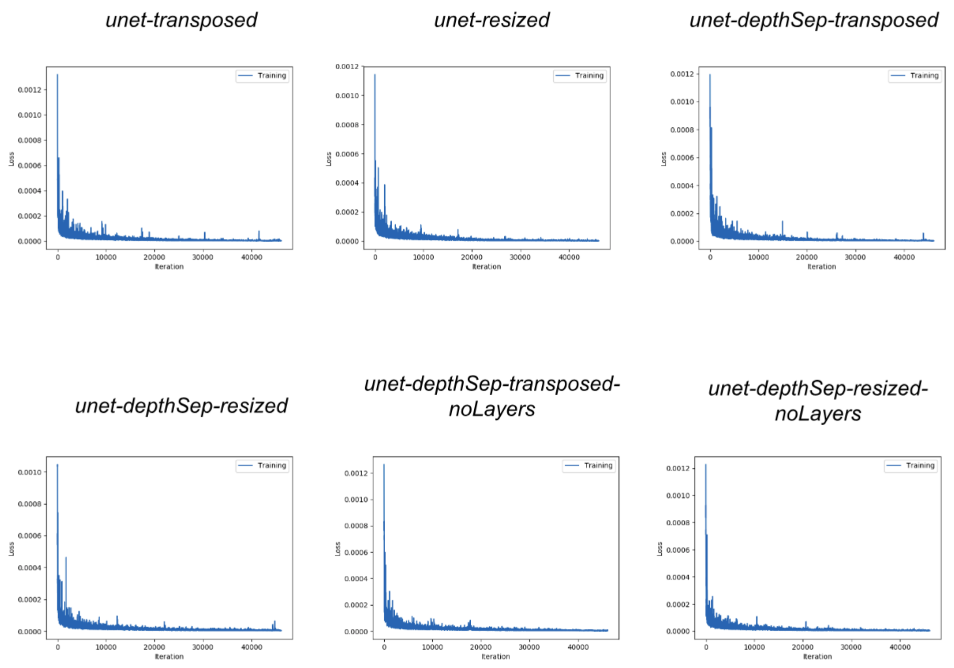


Figure 5.1.: Graph for Training Loss

The models trained are listed in the Table 5.1

Models	Convolution type	Upsampling	IL_{bool}
unet-transposed	Regular	Transposed	N.A
unet-resized	Regular	Resized	N.A
unet-depthSep-transposed	Depthwise Separable	Transposed	Present
unet-depthSep-resized	Depthwise Separable	Resized	Present
unet-depthSep-transposed-noLayers	Depthwise Separable	Transposed	Absent
unet-depthSep-resized-noLayers	Depthwise Separable	Resized	Absent

Table 5.1.: Models for semantic segmentation. IL_{bool} indicates the presence or absence of the normalization and activation layers between depthwise and pointwise convolution operations.

These models are evaluated on a test set from the same data distribution. The accuracy metrics used are Intersection Over Union and per-pixel accuracy. Table 5.2 lists these evaluation metrics.

Models	Acc.	IOU	CPU Time$_1$	GPU Time$_1$	CPU Time$_2$	GPU Time$_2$	Params (mil)	FLOPS (bil)
unet-transposed	96.338	86.512	40.833	1.761	328.072	9.792	24.39	23.79
unet-resized	96.074	85.669	43.546	1.802	319.553	9.483	23.62	23.14
unet-depthSep-transposed	96.950	86.766	18.476	1.301	173.305	14.869	15.43	18.11
unet-depthSep-resized	96.960	87.369	20.131	1.298	182.461	10.7	15.01	16.97
unet-depthSep-transposed-noLayers	95.800	85.111	14.993	1.299	89.528	7.302	15.43	18.11
unet-depthSep-resized-noLayers	95.826	85.160	17.239	1.215	147.411	7.363	15.01	16.97

Table 5.2.: Evaluation metrics (Acc. - Accuracy, IOU - Intersection Over Union), Inference time (Time$_1$ the time taken for running a batch of 200 images in seconds and Time$_2$ is the time taken for running a batch of a single image over 200 iterations in milliseconds) , number of parameters (Params) in millions and number of FLOPS in billions.

As you can see from Table 5.2, architectures utilizing depthwise separable convolutions instead of regular convolutions incur far less processing time on CPU compared to the latter. While the same should be true for GPUs as well, the current implementations of depthwise separable convolutions do not fully utilize their processing power[2] and haven't been able to achieve a similar speedup in computation. In addition to a boost in performance, depthwise separable convolutions have far lower number of parameters than the regular ones, hence larger batches of data can be trained together if needed. These performance benefits occur even though the accuracy of the network remains, more or less, the same.

Figure 5.2 shows the prediction results for 2 images taken from the test partition of the synthetic dataset on which all models were trained.

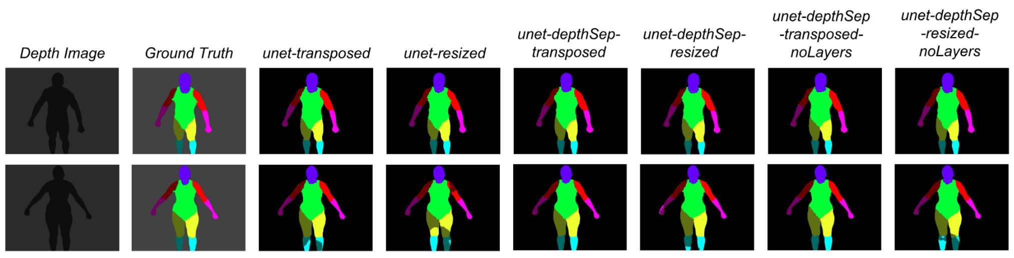


Figure 5.2.: Segmentation results for depth images from the test partition of the synthetic dataset on which all the models were trained on.

Figure 5.3 shows the prediction results for 4 images taken from a real dataset aquired using Kinect V1

Note from the prediction results shown in Figure 5.2 and 5.3, that while the networks perform reasonably well on depth images sampled from the dataset created artificially using scans from [29] and the blender - a part of which is used for training the networks - their performance worsens considerably on real depth data from Kinect V1 camera.

A reason for this may be that these images are sampled from different distributions. Moreover, the depth images captured by the Kinect device are not perfect. They often have holes and are noisy. A potential solution to this predicament could be hand-labelling some of the data acquired via Kinect and using them for training

[2]https://devtalk.nvidia.com/default/topic/1025870/jetson-tx2/depthwise-convolution-is-very-slow-using-tensorrt3-0/

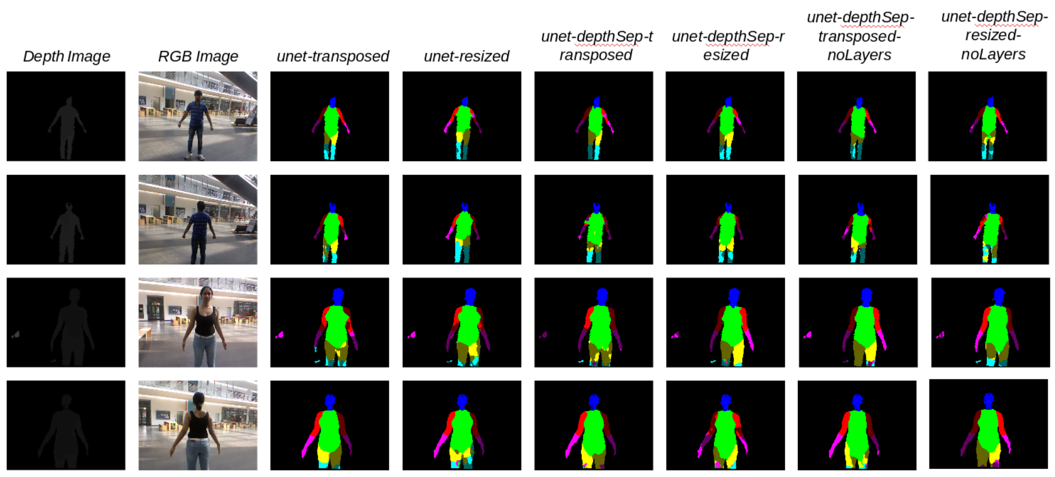


Figure 5.3.: Segmentation results for depth images taken by Kinect V1.

too.

Also, from the results, we can see that the checkerboard artifacts are not as visible as before training for networks with transposed convolutions. Resizing + convolution however, still has a benefit over transposed convolutions when you can replace the convolution operation with a depthwise separable convolution operation as it may result in the reduction of parameters (See Table 5.2).

5.2. Label Fusion and Body Part Extraction

Figure 5.4 shows the results of fusing the label maps onto the 3D model and extracting a body part using the vertex color information for two sets of depth maps acquired via Kinect. The point cloud for the requisite body part is acquired by collecting the vertices whose RGB value lies within a range determined by the color of the label (specified below):

1. Background - ([0,50),[0,50),[0,50))
2. Torso - ([0,50),[177,255],[0,50))
3. Head - ([0,50),[0,50),[177,255])
4. Upper Left Arm - ([177,255],[0,50),[0,50))
5. Upper Right Arm - ([50,177),[0,50),[0,50))

6. Lower Left Arm - ([177,255],[0,50),[177,255])
7. Lower Right Arm - ([50,177),[0,50),[50,177))
8. Upper Left Leg - ([177,255],[177,255],[0,50))
9. Upper Right Leg - ([50,177),[50,177),[0,50))
10. Lower Left Leg - ([0,50),[177,255],[177,255])
11. Lower Right Leg - ([0,50),[50,177),[50,177))

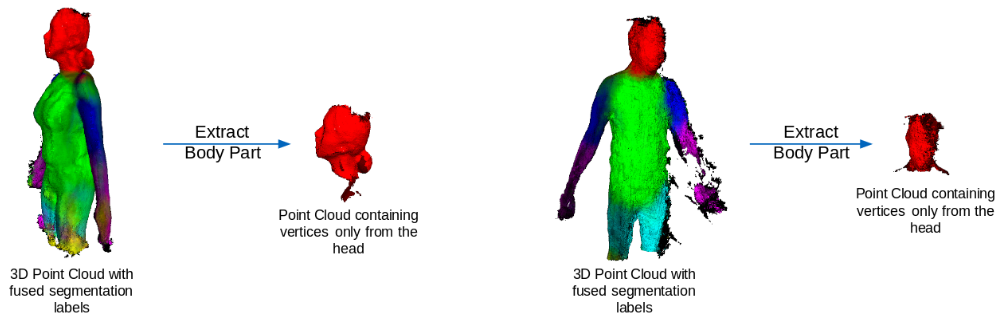

Extract Body Part
Point Cloud containing vertices only from the head
3D Point Cloud with fused segmentation labels
Extract Body Part
Point Cloud containing vertices only from the head
3D Point Cloud with fused segmentation labels

Figure 5.4.: Fusion and extraction results for depth images taken by Kinect V1.

6. Future Work

6.1. Domain Adaptation

As can be seen from results in Figure 5.3, while the prediction of segmentation labels on the data from Kinect is not random, it is still not as good as it is on the synthetically created dataset. An attempt to correct this behavior will possibly require the investigation and implementation of domain adaptation techniques. One possible solution to this is augmentation of the training dataset with hand-labelled depth images acquired using a Kinect sensor.

6.2. A more sophisticated label fusion technique

Averaging over the RGB values of the labels applicable for each vertex in the 3D model is a rather crude method and forces us to manually choose a range of values that define a label for a vertex. This is a rather difficult task since there is a high chance that these ranges overlap. A better method may be to take the mode of the labels applicable to each vertex in the 3D model. However, as pointed out before, limited GPU memory hurdles the implementation of this technique. A possible solution could be down-sampling the vertices and color them using modes and then apply a nearest neighbor policy. The uncertainty with which a segmentation label is predicted (by using a softmax layer over the logits in the network) could also be taken into account for better results.

7. Conclusion

In this body of work, we proposed a system for segmenting 3D Human Models into constituent body parts. Our proposed system consisted of two major components - a Deep Neural Network based on U-Net with depthwise separable convolutions for segmenting 2D depth maps and a modified implementation of KinectFusion[20] for fusing the predicted segmentation label maps into a segmented 3D Model. Due to the lack of suitable datasets for training our models, we created our own using 3D scans derived from the CAESER dataset by [29] and the Blender Software [5]. We compared different segmentation models in this work and presented empirical results and qualitative discussion showing the advantages of U-Nets with depthwise separable convolutions over regular U-Nets, especially in terms of latency - for nearly the same accuracy, our experiments showed a decrease in computational latency by nearly 25% on GPU and by nearly 47% on CPU for U-Nets with depthwise seperable convolutions over regular convolutions. We further showcased the results from fusing the predicted labels to create a segmented 3D model from a dataset of real depth images (collected using Kinect) and showed how to extract a body part from it.

A. Additional Information

A.1. Source Code

The source code of this project can be accessed from the following repositories :

- Data collection and Training pipeline for segmentation - https://gitlab.lrz.de/ga83tuc/segmentation/tree/master
- Segmentation pipeline c++ - https://gitlab.lrz.de/ga87yiq/mart/tree/master/segmentation

A.2. Docker Image

A docker image with all the installed dependencies for inference (not training) is available here :

- https://hub.docker.com/r/neha191091/idp-environments/

Bibliography

[1] Rolf Adams and Leanne Bischof. Seeded region growing. *IEEE Transactions on pattern analysis and machine intelligence*, 16(6):641–647, 1994.

[2] Dragomir Anguelov, Praveen Srinivasan, Daphne Koller, Sebastian Thrun, Jim Rodgers, and James Davis. Scape: shape completion and animation of people. In *ACM transactions on graphics (TOG)*, volume 24, pages 408–416. ACM, 2005.

[3] Jilliam María Díaz Barros, Frederic Garcia, and Désiré Sidibé. Real-time human pose estimation from body-scanned point clouds. In *VISAPP (1)*, pages 553–560, 2015.

[4] Paul J.; N.D. McKay Besl. A method for registration of 3-d shapes". ieee trans. on pattern analysis and machine intelligence. los alamitos, ca, usa: Ieee computer society. 14 (2): 239–256. doi:10.1109/34.121791, 1992. URL: `https://ieeexplore.ieee.org/document/121791/`.

[5] Blender. Open-source 3d computer graphics software. URL: `https://www.blender.org/`.

[6] Claude R Brice and Claude L Fennema. Scene analysis using regions. *Artificial intelligence*, 1(3-4):205–226, 1970.

[7] Tony F Chan and Luminita A Vese. Image segmentation using level sets and the piecewise-constant mumford-shah model. In *Tech. Rep. 0014, Computational Applied Math Group*. Citeseer, 2000.

[8] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *arXiv preprint arXiv:1606.00915*, 2016.

[9] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[10] François Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint*, pages 1610–02357, 2017.

[11] Visual Computing Lab ISTI CNR. Meshlab. http://meshlab.sourceforge.net/.

[12] Daniel Cremers, Florian Tischhäuser, Joachim Weickert, and Christoph Schnörr. Diffusion snakes: Introducing statistical shape knowledge into the mumford-shah functional. *International journal of computer vision*, 50(3):295–313, 2002.

[13] Christian Diller. Mart: kinect_fusion. `https://gitlab.lrz.de/ga87yiq/mart`, 2017-2018.

[14] Sumit Dugar. Development of a system that allows registration of segmented point cloud to patient ct data and provide augmentations. 2018.

[15] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

[16] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[17] Guosheng Lin, Anton Milan, Chunhua Shen, and Ian Reid. Refinenet: Multi-path refinement networks for high-resolution semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[18] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[19] David Mumford and Jayant Shah. Optimal approximations by piecewise smooth functions and associated variational problems. *Communications on pure and applied mathematics*, 42(5):577–685, 1989.

[20] Richard A. Newcombe. Kinectfusion: Real-time dense surface mapping

and tracking, 2011. URL: `https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/ismar2011.pdf`.

[21] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016. URL: `http://distill.pub/2016/deconv-checkerboard`, `doi:10.23915/distill.00003`.

[22] Gabriel L Oliveira, Abhinav Valada, Claas Bollen, Wolfram Burgard, and Thomas Brox. Deep learning for human part discovery in images. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 1634–1641. IEEE, 2016.

[23] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[24] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[25] Alireza Shafaei and James J Little. Real-time human motion capture with multiple depth cameras. In *Computer and Robot Vision (CRV), 2016 13th Conference on*, pages 24–31. IEEE, 2016.

[26] Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, and Andrew Blake. Real-time human pose recognition in parts from single depth images. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1297–1304. Ieee, 2011.

[27] Lingyu Wei, Qixing Huang, Duygu Ceylan, Etienne Vouga, and Hao Li. Dense human body correspondences using convolutional networks. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, pages 1544–1553. IEEE, 2016.

[28] Zhiqiang Wen, Yi Yan, and Haiyan Cui. Study on segmentation of 3d human body based on point cloud data. In *Intelligent System Design and Engineering Application (ISDEA), 2012 Second International Conference on*, pages 657–660. IEEE, 2012.

[29] Yipin Yang, Yao Yu, Yu Zhou, Sidan Du, James Davis, and Ruigang Yang. Semantic parametric reshaping of human body models. In *3D Vision (3DV), 2014 2nd International Conference on*, volume 2, pages 41–48. IEEE, 2014.

[30] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.