REVISITING PATCHNET: STUDY OF ALTERNATIVE ARCHITECTURES

DEEP LEARNING FOR VISUAL RECOGNITION - PROJECT REPORT

Krisztián Bokor
Master's Student of Computer Science
Aarhus University
Aarhus, Denmark
202202855@post.au.dk

Ginés Carreto Picón
Master's Student of Computer Science
Aarhus University
Aarhus, Denmark
202202876@post.au.dk

Marc Johler
Master's Student of Data Engineering and Analytics
Technical University of Munich
Munich, Germany
marc.johler@tum.de

December 9, 2022

ABSTRACT

3D reconstruction problems from 2D representations have always been challenging and computationally demanding problems. There exist visualizations that embed 3D information onto a 2D representation in order to facilitate further the reconstruction, and also to help in other tasks such as image segmentation, augmented reality or robotics. Deformable objects add an additional layer of difficulty onto all of this. In this project we revisit *PatchNet* - a neural network which process image crops in order to achieve better results for all of this tasks - and also we propose modifications on its architecture and patching methods in order to make a study of their performance.

1 Introduction

In the last decade, deep neural network architectures have proven their great effectiveness for computer vision tasks. They use self-learned features from a set of examples in order to generalize over unseen situations that have a relation with the training examples. The motivation for this approach was to develop a system that performs as well as the human perception. Neural networks attempt to mimic human cognition in making sense of two-dimensional data, (amongst other representations of information) and the first major breakthrough in this initiative was the outstanding result of the *AlexNet* [1] architecture on the classification of the large-scale *ImageNet* [2] *dataset*. Since the achievement of this milestone, there have been numerous attempts at tackling various computer vision tasks with deep convolutional neural networks (CNNs).

One specific such task is the challenge of accurately approximating the 3D characteristics of a surface or object based on single or multiple two-dimensional renderings of said ground truth surface/object. The major challenges of this task are in essence on the perspective and limited information available of an object (for example, it may obstruct several portions of the object from the viewpoint). In the case of humans, we reconstruct the obstructed regions from our experience in observing similar objects. Similarly, it has been discovered by Tatarchenko et al. [3] that neural networks also tend to reason based on the classification of the object instead of visual information in order to recreate the scenes in total 3D reconstruction.

Secondly, since the 3D reconstruction models are given image examples with RGB values, lighting also plays a major role on the pixel intensities. A positive characteristic of the 3D reconstruction initiative is that given a *dataset* of meshes,

point clouds or other representation of spatial information, one is able to create a quasi-infinite *dataset* of renderings, from the virtually innumerable amount of viewing angles, lighting conditions etc.

As extensively elaborated in the review of Khan et al. [4], there are numerous methods of representing a 3D shape in a digital form. One of the most renowned of these is storing an object as a *mesh of vertices*, which are then connected in order to create triangular or polygonal faces. The ubiquitous OpenGL API [5] uses this representation. However, approximating such a complex data structure based on the features of an RGB image is a difficult task. Instead, one might look at simpler yet expressive ways of describing three-dimensional information. Such a representation is e.g. a depth map, which aims to store the z value (detph, distance from the viewpoint) of the rendering image, and the normal map expressing the orientation of an object in a given pixel.

After due consideration and the shortcomings of optimistic efforts in tackling the problem of 3D mesh reconstruction, we have decided to focus on solving the task of regressing these simpler representations from a single perspective encoded in an RGB image. Our aim was to implement such a neural network, better understand its results, and attempt to improve its performance via adapting both its input data preprocessing and its architecture.

In Chapter 2, related projects are discussed and compared to our proposal. After that, Chapter 3 details the different approximations tried, in terms of *datasets*, network architectures and related concepts. The results are displayed in Chapter 4 and compared to other existing approaches. At the end, in Chapter 5 the results and methodology are discussed, and also ideas for a continuation of the project.

The whole code has been implemented in Python using the Tensorflow API [6] and is publicly available on Github (`https://github.com/Atamarado/DLVR_3DReconstruction`).

2 Related work

An overview of essential contributions in the domain from recent years is given by Khan et al. in their previously mentioned review [4]. Apart from reviewing different network architectures like *Pixel2Mesh* [7] or *VANet* [8], Khan et al. publication also offers an overview of different representations of 3D shapes, frequently used data sets and evaluation metrics that are common for 3D reconstruction algorithms.

According to this review, compared to the complexity of reconstructing a vertex mesh from a 2D rendering, for which good results have been achieved by Wang et al. [7] using a graph convolutional network (GCN), regression with *depth* and *normal maps* as ground truth values can be solved with considerably less complexity. As it has been proven by Kendall et al. [9] amongst others, creating a model in which the encoded features get used by multiple decoders to solve different, yet related problems, the encoded features are going to become more general, thus mitigating overfitting. This notion has been applied to solve the two previously mentioned regression tasks simultaneously: Bednarik et al. [10] have performed the regression of the depth map, the normal map and the triangulated mesh of a two-dimensional image, using one singular decoder in order to share features. They have found that this architecture is *ill-suited* for the mesh regression however, thus prompting further work based on theirs to omit this task.

Our main source of inspiration was one of these further adaptations. Tsoli et al. [11] argued that 3D reconstruction based on a single image of a deformable surface is essentially impractical since it is extraordinarily difficult to compile a *dataset* of all possible deformations of a surface. However, the same deformation patterns can be observed invariant of their location on the surface. Using this notion of *spatial invariance*, they proposed a network that breaks up the input images into a set of *smaller sized overlapping patches*, and train the model based on this information, to then reassemble the predictions of both regression problems in an intelligent way. They managed to achieve better performance and a more generalized model this way. Another key advantage of this method compared to different contributions is that the input images can have a variable shape for a given patch size.

We aim to implement this network based on the description in the article. Inspired by the original publication, we investigate further the behaviour of this model, the distribution of the error of both regression tasks, and also their relative progress during training. Ultimately, we try to improve the model's accuracy by substituting its architecture with different approaches, such as applying *skip-connections* inspired by the *U-Net* [12] architecture between the encoder and both decoders, and replacing the standard back-end with an inception-based encoder [13]. The original version of *PatchNet* [11] is treated as the baseline model in this report, whilst the different adaptations are considered "potential improvements" of it.

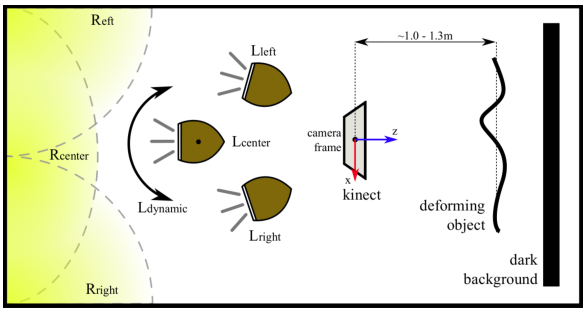


Figure 1: The surfaces have been lit with four different indirect sources of light, which is described in the file names of the data instances. Source: `https://www.epfl.ch/labs/cvlab/data/texless-defsurf-data/`

3 Methods

3.1 Used 3D shape representations

In this project we focus on the reconstruction of two types of 3D representations: The depth map and the normal map.

The depth map contains a single value for each pixel of a 2D image, which provides the distance from the camera to the corresponding point in space. They are also commonly referred to as *Z-buffers*, since considering the central axis of the view to be aligned with the viewpoint's Z axis, the depth values are the z values of the pixel's three-dimensional coordinates. Depth maps are considered as very sparse 3D geometry, since they only contain points directly in the line of sight of the camera. Therefore they cannot represent a complete 3D topology on their own but only the shape of the front face of the represented surface [4].

normal maps contain the coordinates of three-dimensional vectors perpendicular to the represented surface at a given pixel. This means that the 3D orientation of the surface is known at any given point in space that is visible from the camera [4]. These vectors are stored as three-dimensional tensors, similar to RGB images, with the distinction that they contain floating point vectors at each pixel, the norm of which are equal to 1 (*unit normals*). Thus, regression with the aim of approximating normal maps predicts three values per pixel.

Since both depth and normal maps are limited to one perspective and don't contain information about the hidden surfaces of the objects in the scene, they both represent so-called "partial surface models" [4].

3.2 Dataset

The *dataset* we have used was created by Bednarik et al. [10]. It is a freely available *dataset* consisting of 2D images of textureless, white deformable surfaces with different lighting conditions of objects from five different classes: *"cloth", "t-shirt", "sweater", "hoody"* and *"paper"*. The labels are respectively their corresponding depth and normal maps, a relatively few instances having ground-truth meshes as well. The main motivation behind the *dataset* creation was to compile a large scale *dataset* of deformable surfaces, putting emphasis on their lack of textures, which makes 3D reconstruction significantly more difficult. The images were taken with a Microsoft Kinect Xbox 360 camera, in order to retrieve accurate and aligned depth maps. The normal maps where then computed by differentiating the smoothed depth maps [10]. Altogether the *dataset* contains 26500 data instances; however not equally split between the classes.

The *dataset* thus contains a triple for each data instance. Firstly the RGB image of the deformed surface, which represents the raw features of our network. The images have been cropped to a size of 224×224, with a considerable amount of the image acting as *background*, a region to be ignored during training. The two ground truth values of the two regression problems are represented in corresponding depth maps and normal maps of each such image, having the same spatial dimensions. The depth maps are scalar values for each pixel, whilst the normal maps have three channels. A key component that the authors did not include directly as files was the so called *foreground map* of each instance. The losses shall only calculated on the foreground pixels of each image, and in order to distinguish these pixels the *dataset* creators used segmentation based on simple thresholding of the raw image and hole filling of the greatest component in order to create this binary mask. In the case of depth masks, the values at the background pixels are all set to zero, in order to represent inverse of infinity distance from the camera, which thus allowed us to also recreate these foreground masks in our dataloader based on this distinction.

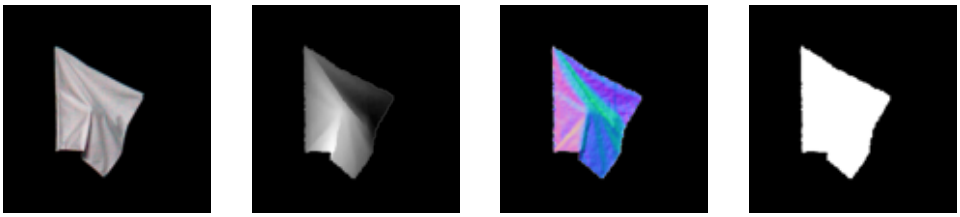


Figure 2: A data instance and corresponding depth map, normal map and foreground mask. The image corresponds to the most common (*cloth*) class.

A key thing to note, which was clarified to us by the original authors, is that the scalar, floating point depth values stored in the depth maps are in the scale of *meters*. This means that the values at each foreground pixel are equal to the actual distance of the surface at that pixel from the camera, expressed in meters.

3.3 Problem formulation

The objective is to reconstruct a textureless deformable surface's depth and normal map from an RGB-image, representing a single view on a three-dimensional deformable surface. Let I be the data of a single image with spatial size $H \times W$, represented as a tuple (R, M, D, N), where $R \in \mathbb{R}^{H \times W \times 3}$ is the RGB-encoding of the image. As mentioned in Subsection 3.2, in our case both H and W is equal to 224.

M corresponds to the image's foreground mask, which contains a value of one in each foreground pixel and zeros everywhere else. The set representation of all foreground pixels is denoted by F.

D and N correspond to relative depth respectively normal map of the image.

The relative depth map is computed by using the recorded depth map Q and subtracting its mean depth in the foreground pixels [11]:

$$D = Q - Z \tag{1}$$

where

$$Z = \frac{\sum_{f \in F} M_f \cdot Q_f}{\sum_{f \in F} M_f} \tag{2}$$

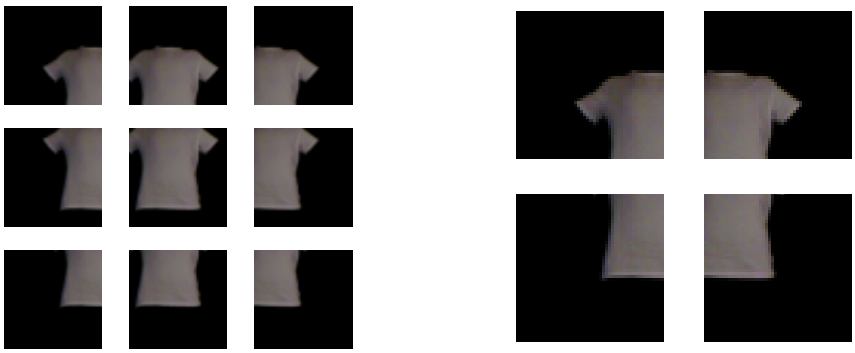
The reason for this is that as explained in Section 3.2, the depth values in Q are expressed in absolute distances, whilst in reality we would like our model to distinguish differences of depth. Another motivation for doing this is to zero-center our data, which is crucial for the success of deep learning via convolutions. The goal is then to predict the relative depth maps D and normal maps N from the RGB-encoding R of the images.

The approach proposed firstly splits the images up into patches $P^{s|s=1,\dots,S}$ of fixed and equal sizes, which form a tuple (R^s, M^s, D^s, N^s) [11] (Section 3.4). The RGB-encoding R^s of the patches are then used as features for training different architectures (Section 3.5) to predict the relative depth maps D^s and the normal maps N^s of the patches. To reduce the number of necessary *backpropagations*, we introduce batch processing in our implementation. (Section 3.6)

After successful training on the patch level, the output is used to predict the depth map patches and normal map patches of the whole images I. Those patches are then stitched back together to maps of the same dimension as the initial image by considering the position of each patch in that image. This is achieved by averaging over the overlap regions and in the case of depth maps additionally adding a certain translation offset for each patch [11]. The stitched maps are then smoothed in the areas around patch boundaries using bilateral filtering before they can be finally evaluated against their ground truth.

3.4 Split an image into patches

We implemented the original patching method from *PatchNet* [11] as well as our own custom method, which splits the image into the minimum necessary number of patches and ensures that the overlaps of neighboring patches are the same up to deviations of at most one pixel caused by non-divisibility. In Section 4.2 of [11] the authors claim that increasing the overlap is conceptually similar to increasing the amount of training data. With our approach, we want to evaluate, which method is more efficient in terms of the trade-off between performance and computational time.



(a) Fixed overlaps (b) Variable overlaps

Figure 3: Comparison between the two splitting techniques, described in Subsection 3.4.

The authors have found through experimentation that the optimal patch size for the 224×224 images is 128×128. The overlap they described is half of each spatial dimension of the patch, meaning half of the width of each patch is shared with another patch, and the whole patch is redundant in the case that it has two neighbours on each side horizontally or vertically. This is why we coined the approach of instead splitting the image into the least number of patches given a fixed patching size, as described in the previous paragraph.

3.5 Network architecture

We implemented the network proposed in [11] as our baseline model. Due to the fact that skip-connections led to some remarkable improvements in different deep learning models in the literature and have some very beneficial theoretical properties [14, 15], we examine the performance improvement by adapting the network in a *U-Net*-like [12] fashion. A second variant that we present is the introduction of inceptions modules [13, 16] in the network architecture, which can potentially improve the performance of the encoder in an efficient way compared to regular convolutions.

3.6 Batch processing

The implementation of batch processing is not straightforward, since for validation we want to process the prediction of single patches and whole images at the same time (see Section 4.1). Therefore we use some sort of *pseudo-batch size*, which corresponds to a fixed number of images loaded in a training/validation iteration. These images will then be split into patches according to Section 3.4. This results in a batch with a variable number of patches, due to the fact that we allow images with different spatial sizes and therefore also a different number of patches. This batch will then be used in the training or validation iteration.

3.7 Loss

We implement our loss function according to Section 3.2 from [11], by considering the depth loss $\mathcal{L}_D$ and the normal loss $\mathcal{L}_N$ with equal weight. Without describing in too much detail, the depth loss is the average pixel-wise absolute loss between the ground truth and the predicted depth value. The normal loss on the other hand is a weighted combination of two different losses: first of which is a *linearized version of the cosine similarity loss*, denoted $\mathcal{L}_a$ as angular loss, which calculates the radian over π of the angle between the predicted and the actual normal vector value at each pixel. The angle enclosed by the two vectors is calculated by using the formula of the dot product of two vectors. Since we divide by π, the range of values for this loss is $[-1, 1]$. The second component of the loss is a unit length loss $\mathcal{L}_l$, which, similarly to weight decay, gives the backpropagation an incentive to reduce the predicted normal values to unit length. The formula is the squared error between the norm of the prediction minus 1.

For the computation of $\mathcal{L}_N$ we also set the ratio κ between $\mathcal{L}_a$ and $\mathcal{L}_l$ equal to 10 like proposed in [11].

Note that we omit to divide the loss through the total number of patches in a batch of samples. This ensures that every patch has the same weights for the gradients from a global perspective, independent of the order of loading and the number of other patches, that are contained in the same batch. Were we to divide the calculated loss for each image by the number of patches, considering that our model can process images of variable size as long as it is divided into

patches of a given size, the scale of the loss would be independent of the original image size. This means that the patches of a smaller image would have a larger impact on the gradient descent than a patch of a larger image.

3.8 Stitching and Refinement of the maps

To receive depth respectively normal map for the whole image we stitch the predicted depth and normal map patches together like proposed in [11]. We slightly adapted the way we compute the normal maps, since we are applying normalization two times: first, at each pixel location for each patch, so that all the normals of different patches will have the same weight in the overlap regions. Second, after the refinement step via bilateral filtering (Section 3.4 of [11]), because we want to counter the shrinking of lengths in the overlap region due to the averaging over normal maps. This is formally explained by the following corollary:

Corollary. *Given three-dimensional vectors v_i with $i = 1, \dots, n$ with $||v_i||_2 = 1$ for all i. Let $\bar{v}$ be the average over all v_i along the dimensions: $\frac{1}{n}\sum_i^n v_i$. Then the value of $||\bar{v}||_2$ is bound by 1 from above.*

Proof. Let v_i with $i = 1, 2$ be defined as above. Then for $\bar{v} = \frac{v_1+v_2}{2}$,

$$||\bar{v}||_2 = ||\frac{v_1 + v_2}{2}||_2 = \frac{1}{2}||v_1 + v_2||_2 \leq \frac{1}{2}(||v_1||_2 + ||v_2||_2) = 1 \tag{3}$$

by the use of the triangle inequality. Using induction we can extend this statement to an average of n different v_i, which finishes the proof. □

The stitching of depth maps is more complex, since in Section 3.3.1 of [11] they have noted that ideally the depth maps of neighbouring patches should be translated along the viewing direction of the camera. Essentially, all the patches of an image are translated (meaning brought closer or further away from the camera) in a way that the overlapping regions have values most similar. To constrain this to a single solution, the offset of the first patch is set to 0, and all the other patches are aligned accordingly. After this, the stitched depth map values are also smoothed out using bilateral filtering along the overlap regions.

4 Results

4.1 Overview

Evaluation *dataset*

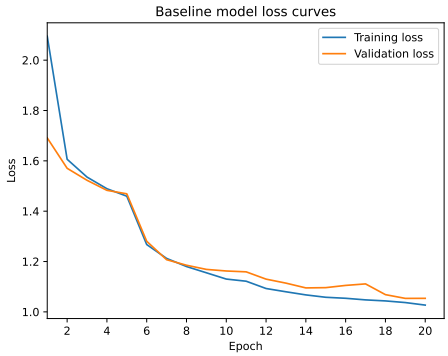
In order to be able to compare our results to those of *PatchNet* [11], we decided to use the same *dataset*, which was initially introduced in [10]. We use 20 percent of the data for testing and split the remaining data $80/20$ into training and validation data. That way it is ensured that the model evaluation is made on unseen data to avoid rewarding memorization.

Evaluation metrics

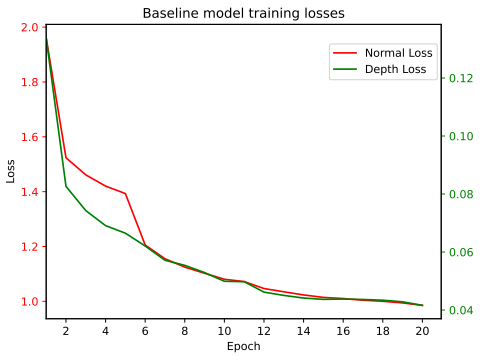
We evaluated our work based on the combination of the values values of the angular error E_A and the depth-based error from depth generation via patch depth maps stitching E_D^t as described in section 4.1 of [11]. As a basis for analysis, we have also plotted the progression of the two separate regression losses, in order to compare their scale and overall training performance.

The original approximation [11] used only the angular loss and the depth-based error as evaluation metrics, expressing the former in matter of degrees, and the latter in the scale of *millimeters*. The angular representation is straight-forward, since the loss itself is a ratio of the radian, thus the angle can be easily calculated. The *millimeter* length was unusual however, but considering that the depth values in the data are expressed in meters, the loss can be easily converted to the required scale.

In addition to default validation with the patches of the images in the validation set, we additionally perform validation based on the error of the reconstruction with respect to the full image. Tsoli et al. [11] focused on the loss metrics of the patches, whilst we investigated the performance of the regression of the whole image posterior to the stitching procedure. This way we can observe if the network's isolated performance as well as its ability to generate patches which are useful for the image-level reconstruction improve in the same manner.



(a) Training and validation loss progression of the baseline model over 20 epochs.



(b) Training loss of the baseline model broken into the two separate loss components over 20 epochs.

Figure 4: Various loss curves of the training of the baseline model, described in Subsection 4.2.

Hyperparameters of Optimizer and Bilateral Filtering

For all the experiments we use Adam optimization with a fixed learning rate of 0.001 and set β_1 to 0.9, β_2 to 0.999 and ϵ to 10^{-7}. The bilateral filtering is used with the same kernel size and variances as described in Section 4.1 of [11].

4.2 Baseline model

In the following we compare the baseline method [11] to our custom methods by applying different adaptations and investigate the influence on the performance of prediction.

To obtain a baseline result for our research, we considered the recreated architecture described in Section 3.5 as the baseline network. Worth to note that this also implies that the patching method used was the one described by the authors, with fixed-sized overlaps yielding more patches than theoretically necessary. With the prescribed overlap size, their method breaks each image up into $3 \cdot 3 = 9$ patches.

The batch size we used - which refers to the number of images loaded, as explained in Section 3.6 - is 32 images, which results in $32 \cdot 9 = 288$ patches at once, in the case of the fixed overlaps. This number of batches resulted in 529 training batches, and 133 validation batches. In the following, we will analyze in detail the results of training this architecture on the entire training data, as a matter of losses, and the ratio and distribution of said losses.

Loss decrease

The average running time of a single epoch upon the whole training *dataset* using *Google Colaboratory* was in the range of 16 to 20 minutes. The observed loss curves in Figure 4 display the expected behaviour of deep learning models, with a sudden decrease of the training loss in the first and second epoch. Due to the fact that we have reduced the dimensions of the inputs by breaking them up into patches, a strong hypothesis was that the smaller input domain would make it more likely that our model would suffer from overfitting.

In order to study the phenomenon, we ran a validation loss calculation over the validation set at each epoch. Our findings (also visible in Figure 4) were that the validation loss tends to decrease along with the training loss, thus removing the concern of the model memorizing the training data, despite having a big loss reduce during the first epochs.

Another key factor in our training that we decided to investigate is the balance of the two losses of the two regression problems: the depth and the normal loss. The depth loss is one order of magnitude smaller than its counterpart, which can be explained by the definition of either metric. Recall that the depth loss in Section 3.7 is the average absolute error between the predicted depth and the ground truth depth value at the respective pixel. In our data, these depth values range within $[1.0, 1.3]$, as it can be seen on Figure 1. Note that the depth values are stored as the distance of the object in the pixel from the camera viewpoint. This means a maximal error of 0.3 for the depth prediction of a pixel. This holds even given the fact that we zero center the data.

The normal loss has two components: the angular and the length component. The angular loss is the radian of the angle between the prediction vector and the actual target vector over π. This means its value ranges from 0 to 1. This value is multiplied by the factor of $\kappa = 10$ however, which explain the magnitude difference between each loss.

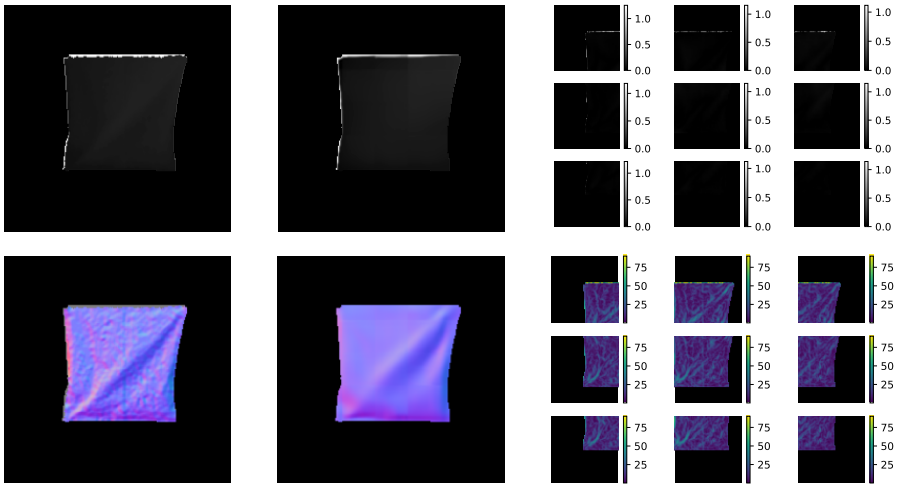


Figure 5: Heatmaps visualizing the distribution of each loss across the prediction. The first two figures in each row are the ground truth and predicted maps, respectively. The scale of the losses in the case of the depth map is meters, whilst the scale of the normal error is in degrees.

Prediction quality

In order to better understand the loss values and the challenges faced by the model, visualizing the distribution of the loss in the predictions was another initiative. Firstly, we plotted as a heatmap the concentration of the absolute difference of the depth map regression, displayed on Figure 5. A common limitation that we observed was that the error appeared to be disproportionately high at the edges of the foreground of the various classes. On the other hand, the regression of the pixels in the inner areas proved to be significantly more accurate.

Performing the same analysis on the normal maps we have learned that the loss is higher in areas where the surface changes orientation, essentially in the creases of the object. This can also be observed in Figure 5.

Comparison with the state-of-the-art

As a way of evaluating the quality of our implementation as compared to the results described by Tsoli et al. [11], we evaluated our trained model with some of the same error metrics as the original inventors of the architecture.

Firstly, as opposed to the loss used for the backpropagation of the depth estimation during training, for which the choice was the *Mean Absolute Error* (MAE), the authors decided to evaluate the model via using the *Mean Squared Error* (MSE). That is they calculated the pixel-wise average of the squared error between the predicted value and the ground truth. Taking this into account, our results compared with theirs on our reserved, unseen test set for whole image prediction are presented in Table 1.

	Tsoli et al.[11]	Ours
E_D^t	$15.12 \pm 4.73mm$	$24.76mm$
E_A	$14.72 \pm 3.39°$	$23.42°$

Table 1: Our test losses compared with the best losses achieved by Tsoli et al.[11] One key thing to note is that we took the best value of the authors' various error metrics, as comparison. As can be seen in their publication, in some cross validation scenarios their model actually performs worse.

4.3 Influence of size of the overlap region

We change the overlap from being half of the patch size (which is in our case $128/2 = 64$) to our flexible overlap approach described in 3.4. Note that this will result in a decrease in the number of patches per image from $3 \cdot 3 = 9$ to $2 \cdot 2 = 4$. We expect that this quasi-reduction of training data will lead to overall lower performance, but that the savings in computation time will outweigh this disadvantage. Therefore we also consider the training time per batch for comparing the two approaches.

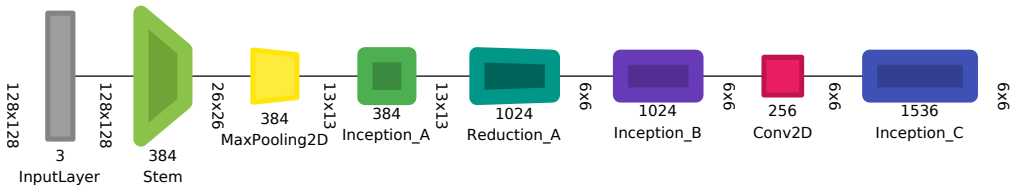


Figure 6: Architecture of the Inception-based encoder [17].

In Table 2, it can be seen that the overall error on the training set is considerably higher for the model with fixed overlaps than for the model with variable overlaps at the first epoch. However, with further epochs we do not see any significant improvement in accuracy. This can be explained by considering that the patches are the same size regardless of their overlaps, but less data makes it easier to fit the training set with a deep network such as this. However, with progression, this initial lead seems to become comparatively negligible.

The average time per batch prediction displayed in Table 3 is approximately proportional to the number of patches in both the case of fixed and variable overlap sizes. The same phenomenon is observed in each architecture. As an additional metric we provide the *Efficiency*, which we define as the inverse of the product of the running time per batch and the final validation loss after 10 epochs from Table 2. We observe that by considering running time and loss in this way, our variable-sized overlap approach improves the efficiency for all three network architectures at least slightly. In real-life applications, the user must of course decide individually with which importance they will consider running time and accuracy each, rather than relying on our metric. Whilst these results are expected, the metrics in Table 3 give a solid basis for further comparison in Section 5 between the efficiency and accuracy trade-off between each network.

4.4 Effect of adding Inceptions

One of the modifications proposed to the original *PatchNet* [11] has been the replacement of the Convolution layers in the encoder for different inception modules, proposed in [13]. These modules offer a computationally efficient way to extract features from imagery. An already implemented version of inception modules has been used [16]. The details of the implementation can be consulted at Figure 6. As such, our expectation was that this adaptation would result in faster prediction time, at the possible cost of loss increase.

Table 2 shows that adding the Inceptions to the network will significantly increase the overall error for the patching with fixed overlap size and also for the patching with variable overlap size. Furthermore, there have been no efficiency gains observed either, as can be seen in Table 3. The main discussion about these results is that there should be changes to take advantage of the inception modules in the decoders, in order to utilize the additional extracted features. By not having those features, the overfitting phenomenon may be occuring.

4.5 Effect of using a U-Net-like network architecture

In a second variant we adopt the network architecture of *PatchNet* by adding skip-connections between the layers of the encoder and the decoder with the same spatial dimensions. This method was proven by [12] to increase the performance of autoencoder networks such as ours. However, referring to Table 2, that comes at a cost of a parameter number increase of two orders of magnitude.

Model	№of parameters	Overlaps	Epoch 1		Epoch 5		Epoch 10	
			Depth	Normals	Depth	Normals	Depth	Normals
Baseline	768204	Fixed	$0,13387$	$1,96116$	$0,06650$	$1,39303$	$0,04991$	$1,08041$
		Variable	$0,10365$	$1,54785$	$0,06228$	$1,21277$	$0,04759$	$1,10290$
UNet	1989452	Fixed	$0,09441$	$2,08526$	$0,05236$	$1,26096$	$0,04059$	$0,95629$
		Variable	$0,09033$	$2,20493$	$0,05873$	$1,69562$	$0,05386$	$1,68133$
InceptionNet	10756132	Fixed	$0,11732$	$2,56237$	$0,09190$	$1,94582$	$0,07905$	$1,67416$
		Variable	$0,10224$	$2,50226$	$0,08105$	$1,67286$	$0,06649$	$1,67302$

Table 2: Validation loss achieved by each experiment as a factor of data preparation setting, encoder architecture and training length. The values represented here are the actual scalar value of the losses, without conversion into degrees or millimeters.

	Fixed		Variable	
	Running time	Efficiency	Running time	Efficiency
Baseline	$2,1300$	$0,41535$	$1,04800$	$0,82938$
UNet	$2,23900$	$0,44802$	$1,18700$	$0,48551$
InceptionNet	$2,40600$	$0,23706$	$1,23900$	$0,46398$

Table 3: Running time and efficiency of different architectures and patch variation. Running time represents the time taken to compute an entire batch of images (32), and efficiency is a own designed metric calculated as $E = 1/(\sum losses * runningtime)$

By taking a look at our loss table again, one can see that using the *U-Net*-like architecture excels out of our networks, since its error with fixed overlap patching is the smallest out of all the results. In comparison, it seems as if using a network with such skip-connections performs significantly worse with our variable-sized patching method.

5 Discussion

Our experiments' goal was to recreate the *PatchNet* [11] architecture to the best of our abilities, investigate its training progress, and attempt to aid its possible shortcomings with a different data preparation method, and a couple of renowned distinct encoder architectures.

We have observed that the baseline network based on the author's implementation is well suited for this multitask [9] regression. In the *dataset* creator's work [10], it is stated that the latent representation created by training the encoder of this architecture with the depth estimation and normal approximation was ill-suited for the regression of 3D meshes. The performance of the other two tasks is notable, however.

The batch normalization layers of the architecture seemed to successfully prevent overfitting, as observed in the loss plots of Figure 4. The multitask nature of the training also seemed to aid the generalization of the network, and both regression tasks' losses decreased at a conveniently similar way.

Our novel approach of breaking the pictures up into the least possible amount of patches did result in faster training, with comparable average losses per patch. As noted in the analysis of the results, the variable patches also proved to yield smaller training error, yet this gap decreased with training time. However, we have found a way to reduce the training time with similar validation losses.

The substitution of the encoder architecture yielded intriguing results. Firstly, the skip connections inspired by the *U-Net* [12] architecture managed to improve the accuracy of our model. The *PatchNet* model is another successful application of such connections.

Replacing traditional convolutional layers with inception modules was not beneficial to the performance of our implementation.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[2] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[3] Maxim Tatarchenko, Stephan R. Richter, René Ranftl, Zhuwen Li, Vladlen Koltun, and Thomas Brox. What do single-view 3d reconstruction networks learn? *CoRR*, abs/1905.03678, 2019.

[4] Muhammad Saif Ullah Khan, Alain Pagani, Marcus Liwicki, Didier Stricker, and Muhammad Zeshan Afzal. Three-dimensional reconstruction from a single rgb image using deep learning: A review. *Journal of Imaging*, 8(9), 2022.

[5] Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2*. Addison-Wesley Longman Publishing Co., Inc., USA, 3rd edition, 1999.

[6] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael

Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[7] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. 2018.

[8] Yi Yuan, Jilin Tang, and Zhengxia Zou. Vanet: a view attention guided network for 3d reconstruction from single and multi-view images. In *2021 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6, 2021.

[9] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. *CoRR*, abs/1705.07115, 2017.

[10] Jan Bednarik, Pascal Fua, and Mathieu Salzmann. Learning to reconstruct texture-less deformable surfaces from a single view. In *2018 International Conference on 3D Vision (3DV)*, pages 606–615, 2018.

[11] Aggeliki Tsoli, Antonis Argyros, et al. Patch-based reconstruction of a textureless deformable 3d surface from a single rgb image. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[12] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[13] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning, 2016.

[14] Michal Drozdzal, Eugene Vorontsov, Gabriel Chartrand, Samuel Kadoury, and Chris Pal. The importance of skip connections in biomedical image segmentation. In *Deep learning and data labeling for medical applications*, pages 179–187. Springer, 2016.

[15] A Emin Orhan and Xaq Pitkow. Skip connections eliminate singularities. *arXiv preprint arXiv:1701.09175*, 2017.

[16] Ginés Carreto Picón, Maria Francesca Roig-Maimó, Miquel Mascaró Oliver, Esperança Amengual Alcover, and Ramon Mas-Sansó. Do machines better understand synthetic facial expressions than people? In *Proceedings of the XXII International Conference on Human Computer Interaction*, Interacción '22, New York, NY, USA, 2022. Association for Computing Machinery.

[17] Alex Bäuerle, Christian van Onzenoodt, and Timo Ropinski. Net2vis – a visual grammar for automatically generating publication-tailored cnn architecture visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 27(6):2980–2991, 2021.

Appendix

We decided to provide the following information separated from the rest of this report since it is not directly related to the outcome of our project and would break the red line in its presentation.

Change of Project Scope

This section is directed to our professor Henrik Pedersen and all the teaching assistants from the course "Deep Learning for Visual Recognition" of the autumn semester 2022.

As you may have noticed we changed the scope of our project twice. First from making adaptions to *VANet* - another 3D reconstruction network - and then from using *Pixel2Mesh* in combination with the patch-functionality of *PatchNet*. While the first change of scope was necessary because we underestimated the enormous effort the implementation of the respective network requires, we had to cancel our second approach due to dependency issues of the publicly available version of *Pixel2Mesh*, which seem not to be maintained in any way anymore. Even though we made a great effort trying to adapt the existing code, our efforts eventually failed and we couldn't achieve compatibility between all required packages for the implementation.

Reflection on the project

Even though we couldn't achieve the goals we set for ourselves in the very beginning we can take a lot out of the project. One thing is that - like Henrik used to say - in the face of the very limited time we should "start with the skateboard, and not with the car". This was essentially our first big mistake: we had a lot of visions and ideas but we failed to limit ourselves to a reasonable basic approach that is extendable to our more ambitious concept at a later point.

The second thing we learned is that we should never rely at first glance on third-party open-source code from GitHub to work. Even though it is amazing that we have access to the work of many contributors worldwide, the fact that maintaining implementations is not financially rewarded most of the time can lead to cases where those implementations are full of outdated software components and dependency conflicts. Dealing with those kinds of issues was not meant to be part of the project and cost us a lot of time, which we could have needed in the later stages. We learned that we should not depend on something to work, and that we should always be prepared to implement everything we need from scratch in the very worst case.

We are most likely not the first to face issues like that, but maybe our case can be helpful as an example to point out the mentioned key issues, and can therefore reduce the risk for future project teams to end up in the same struggle.