UHASSELT
KNOWLEDGE IN ACTION

Maastricht University

Faculteit Wetenschappen
School voor Informatietechnologie
master in de informatica

Masterthesis

Genereren van een 3D-omgeving vanuit een afbeelding

Koen Laermans
Scriptie ingediend tot het behalen van de graad van master in de informatica

PROMOTOR :
dr. Nick MICHIELS

BEGELEIDER :
De heer Steven MOONEN

2021
2022

De transnationale Universiteit Limburg is een uniek samenwerkingsverband van twee universiteiten in twee landen: de Universiteit Hasselt en Maastricht University.

UHASSELT
KNOWLEDGE IN ACTION

www.uhasselt.be
Universiteit Hasselt
Campus Hasselt:
Martelarenlaan 42 | 3500 Hasselt
Campus Diepenbeek:
Agoralaan Gebouw D | 3590 Diepenbeek

Maastricht University

Faculteit Wetenschappen
School voor Informatietechnologie

master in de informatica

Masterthesis

Genereren van een 3D-omgeving vanuit een afbeelding

Koen Laermans

Scriptie ingediend tot het behalen van de graad van master in de informatica

PROMOTOR :

dr. Nick MICHIELS

BEGELEIDER :

De heer Steven MOONEN

Universiteit Hasselt

Masterproef voorgedragen tot het behalen van de graad van master in de informatica

3D scene reconstruction from a 2D image

Auteur:

Koen Laermans

Promotor:

Dr. Nick Michiels

Begeleider(s):

Dr. Jeroen Put
Dr. Steven Moonen

Academiejaar 2021-2022

1 Abstract

Convolutional Neural Networks (CNN) have great potential in the field of image processing to learn many different things about an image. Some examples are learning how certain objects look like, learning the pose of a person or object and learning the depth map of an image. This research aims to explore each of these areas, provide a prototype that implements 3D scene reconstruction from a 2D image and determine it's potential in real-world scenario's.

Based on many existing architectures, a pipeline is constructed that converts an image to a 3D scene in multiple steps. The application allows users to view the obtained 3D scene and swap furniture models for new models of different styles. This research can serve as a source of information and inspiration for other researchers that focus on 3D scene reconstruction.

Contents

2 Introduction

Virtual versions of real world scenes are often needed in games, virtual reality experiences and robotics. They can be designed with 3D modelling tools, which is time-consuming and expensive. A different approach to designing these 3D models and scenes is by generating them with generative models. The benefits of this approach is that they require less skill and are less time-consuming. There are also many other applications for virtual scenes. One application domain is computer vision and robotics. Computer vision often needs lots of data to train models while publicly available 3D scene databases are scarce. Also robots require virtual scenes to learn autonomous navigation in the real world Ritchie, Wang, and an Lin (2018).
Another application is architectural design, which will be the main focus of this research. Some people want to redesign their living space. They want to try different style furniture to see if a furniture piece actually fits together with the other furniture pieces in the room or they want to rearrange present furniture pieces. They can do this by creating a virtual scene with the many available online tools and manually place different pieces into the scene. It can however be more efficient to start with your current living space and gradually make changes to that scene. For this to work, there has to be a way to quickly get a virtual version of your current scene. One way to achieve this would be to take a picture of a room and automatically convert this to a virtual 3D version. In the field of computer vision, this technique is called 3D scene reconstruction, which will be one of the main topics of this thesis Stefan, Pablo, and Vittorio (2020).
There are many ways to reconstruct a virtual scene. Some methods use a single image. This can be a depth image as used in Song et al. (2017), or it can be a regular image as used in Xiang, Schmidt, Narayanan, and Fox (2017). Other methods use multiple images, as in Denninger and Triebel (2020a) and Seitz and Dyer (1997) among others. Here, the focus will be on a single RGB image that you can easily capture with a smartphone or camera. This will make the application more user friendly.
In this thesis, a pipeline of supervised classification and regression machine learning models is proposed which will take an RGB image of an indoor room as input and output a 3D scene representation. This 3D scene can further be edited by swapping furniture pieces with other pieces. This way, people can view the visual impact of changing their furniture. An example of how this pipeline will work is depicted in Fig 1.

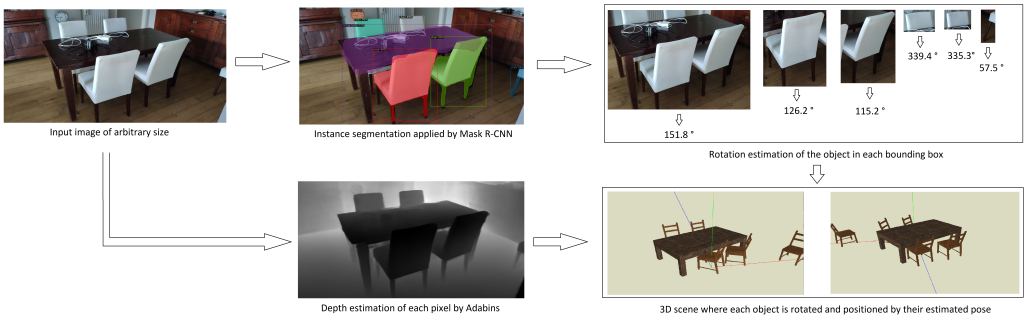

Figure (1) An overview of the process pipeline

The main objective of this research is the following:

To enable a user to view parts of their living room in an application and make modifications to it.

This main objective consists of the following sub-objectives:

Construct a model that can accurately place 3D models in a virtual scene as they are located in a 2D image.

Construct a model that can accurately rotate 3D models in a virtual scene as they are rotated in a 2D image.

Build a user interface that allows the user to easily interact with the virtual scene and make modifications to it.

There are a couple of challenges associated with these objectives. One is the ability to accurately predict the rotation of a furniture piece given only a small cropped out part of an image while parts of the object might be occluded by other objects. Another difficulty is the ability to predict the depth of an object in an image, since images are in 2 dimensions and do not contain any information about the 3rd dimension (the depth).

It is often hard for people to visualise the changes to their indoor environment when buying a new furniture piece. This may lead to wrong choices that can not be reverted. Also choosing a color to paint the walls with can be a difficult

choice. These are the problems that can be solved by the proposed pipeline in combination with a User Interface. There already exist many methods to convert images to 3D scenes Denninger and Triebel (2020b); Seitz and Dyer (1997); Xiang et al. (2017), but they all have different purposes. By specifically aiming at the ability to edit indoor scenes, the accuracy of models can be improved by taking into account specific indoor scene and furniture characteristics like the fact that most furniture stands upright.

This pipeline will be evaluated by observing the accuracy of the individual machine learning models under different circumstances (under low lighting, with occlusions, ...). Also the pipeline as a whole will be evaluated by analysing multiple scene reconstructions with images from real scenario's. Lastly, the user interface will be evaluated based on how user friendly it is and which modifications to the scene it allows.

3 Related work

3.1 Scene reconstruction

There are multiple approaches to scene reconstruction that are useful for different purposes. The techniques we focus on here receive one or more images as input and output a 3D scene. The main challenge of the reconstruction is that the mapping from the 3D environment onto an image can not be inverted.

One approach to scene reconstruction requires a single image as input. An example of this is semantic scene completion from a single depth image Song et al. (2017). In this research, a semantic scene completion network (SSC-Net) is used to first calculate a voxel representation of the scene, and then predict an object label for each voxel that is calculated from the depth map. This prediction does not only apply to visible voxels, but also to non-visible voxels. This is done by inferring contextual information from voxels and their class labels. This results in an accurate representation of the input scene, but low quality models due to the size of the voxels with some errors. A result of this network is depicted in Fig 2.

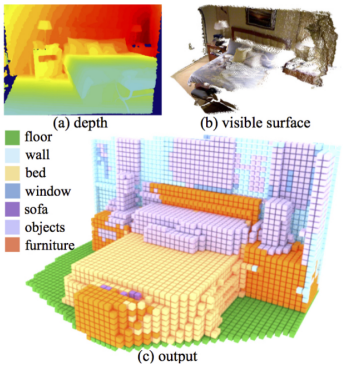

Figure (2) An example result of SSCNet

PoseCNN is another example of using a single RGB image to compute a 3D scene Xiang et al. (2017). PoseCNN first calculates an object class label for each pixel. Then it estimates the rotation of each object using a CNN. Lastly, it estimates the position of each object in 3D space and places a 3D model in a 3D scene with the estimates pose assigned to it. PoseCNN is further explained in chapter 3.3. The main difference with SSCNet is that

it outputs a scene with highly detailed 3D models, but requires a 3D model for each model present in the scene which is not feasible. An example result of this method is depicted in Fig 3. The estimated pose is indicated with a different color.

Figure (3) An example result of PoseCNN

Other approaches to scene reconstruction require multiple images as input. An example of this approach is the use of an RGB image and a normal map to calculate the corresponding 3D scene Denninger and Triebel (2020a). The reason for the need of a normal map is to construct planar surfaces more precisely due to the continuity information given by the normal map. Each pixel is projected into 3D to calculate which voxels are filled. Truncated Signed Distance Field (TSDF) are used to represent the high-resolution voxel output data.
Another example of multiple using input images is Photorealistic Scene Reconstruction by Voxel Coloring Seitz and Dyer (1997). Each image contributes to the coloring of a set of voxels. All sets of colored voxels construct the 3D scene.

3.2 Instance Segmentation

Instance segmentation is a combination of object detection and semantic segmentation. The goal of object detection is to locate individual objects with a bounding box, and classify them with a label. On the other hand, semantic segmentation classifies each pixel into a set of categories without distinguishing between multiple object instances. Instance segmentation is the detection of all objects in an image including the segmentation of each individual object He, Gkioxari, Dollár, and Girshick (2017). There are multiple ways to implement instance segmentation.

Object segmentation can precede object recognition. An example of this is called DeepMask Pinheiro, Collobert, and Dollar (2015). DeepMask is an

object proposal algorithm that aims to find different image regions which are likely to contain one or multiple objects. These image regions, called masks, have a score associated with them. This score estimates the likelihood that the region fully contains an object. The resulting regions are then passed to an object classifier. Other works that follow the same approach are Dai, He, Li, Ren, and Sun (2016) and Dai et al. (2016). However, this approach is slow and less accurate then the next approach.

A second approach is based on parallel segmentation and object classification. One framework that achieved great results with this approach is Mask R-CNN, a region based convolutional neural network framework that performs instance segmentation He et al. (2017). It is an extension of an existing architecture called Faster R-CNN, which performs object detection Ren, He, Girshick, and Sun (2015). It extends Faster R-CNN by adding a branch that calculates a segmentation mask for each predicted Region of Interest (ROI) in parallel to the object detection task.
The 2 stages of Faster R-CNN are a Region Proposal Network and a feature extractor. The first stage proposes bounding boxes for each candidate object. The second stage extracts features from these bounding boxes to predict an object class for each box. The first stage of Mask R-CNN is the same, but it adds another task to the second stage. In parallel to predicting the class labels, it also predicts a binary mask for each ROI. This parallelization ensures that there is little overhead and extra computing time involved in the extension to Faster R-CNN. 2 examples of instance segmentation performed by Mask R-CNN are shown in Fig 4.

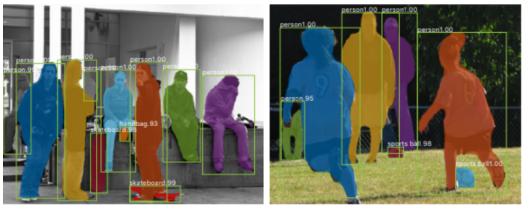

Figure (4) Examples of instance segmentation performed by Mask R-CNN

Fig 5 depicts an overview of the Mask R-CNN architecture. The left part performs ROI and object detection, while the FCN on the right performs semantic segmentation.
Mask R-CNN outperforms other state of the art instance segmentation architectures like FCIS.

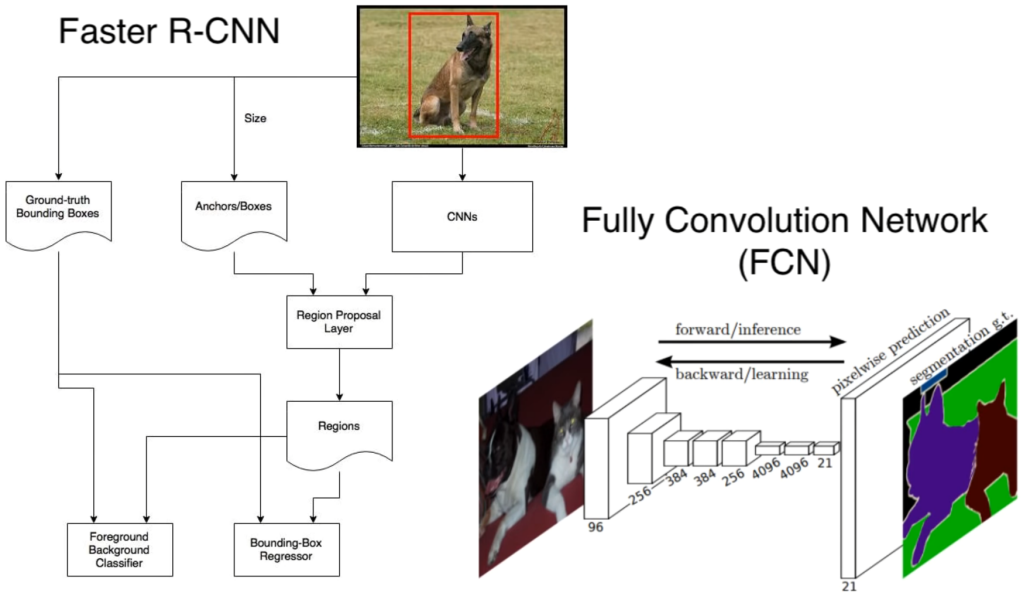

Figure (5) Overview of the Mask R-CNN architecture

Another approach combines object detection and segment proposal into one fully convolutional network, that predicts a likelihood for each pixel that it belongs to a certain object category. These are generally translation invariant, meaning that a pixel will receive the same score at any position in the image. To solve the problem of instance segmentation, a pixel's score should depend on the region around the pixel. Instance-aware semantic segmentation, dubbed FCIS, solves this issue Y. Li, Qi, Dai, Ji, and Wei (2016). First, a region proposal network (also fully connected), proposes multiple regions of interest (ROI). For each region, a pixel-wise score map is calculated. The score maps indicate whether a pixel belongs to an object bounding box or not and whether it is inside an object's instance mask or not. In general, these pixel-wise scores address object classes,boxes and masks. It is a fast approach, but exhibits systematic errors on overlapping instances. An overview of their architecture is depicted in Fig 6. Also some results of FCIS are shown in Fig 7. This also shows the artefacts of overlapping instances.

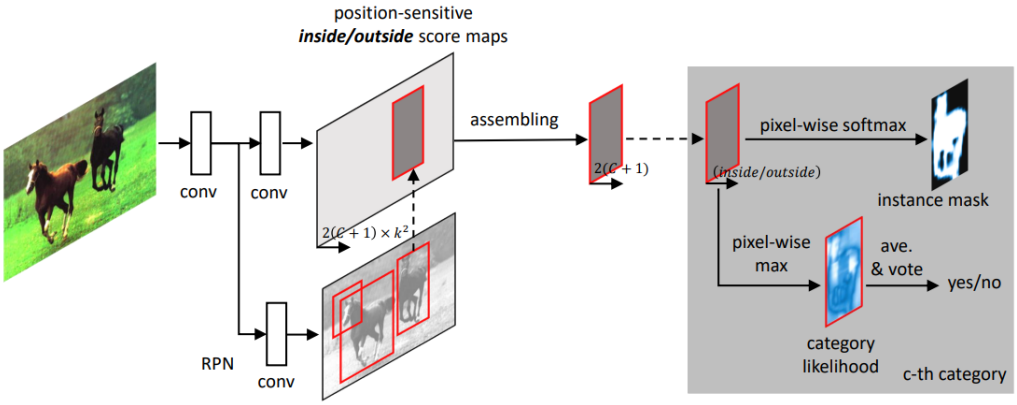

Figure (6) Overview of the FCIS architecture

Figure (7) Results of the FCIS architecture

3.3 Pose Estimation

Knowledge of the pose of an object is of importance to robotics, scene reconstruction etc. When robots want to grab an object, they have to know where that object is and how it is rotated in order to move there end effector in the correct pose. When a 3D scene is reconstructed from a 2D image, the objects have to be placed at a certain location with a certain rotation Gao, Lauri, Zhang, and Frintrop (2018).

One approach to object pose estimation is called PoseCNN Xiang et al. (2017). PoseCNN converts a 2D image into a 3D scene in a couple of steps. It first predicts object labels for each pixel. It then estimates for each detected object the 2D pixel that contains the center of this object. Next, the 3D location of each object is estimated by projecting the center pixel of

each object into 3D space. Lastly, the rotation of each object is estimated. For each object's bounding box, its convolutional features are regressed to a quaternion. Quaternions are 4D vectors used to represent rotations in 3D Goldman (2011). To predict an object label for each pixel, semantic labeling is used. To estimate the translation of an object, the center pixel of the object needs to be calculated first. This is done by a Hough voting mechanism. Each pixel votes for certain other pixels to be the center. The pixel with the highest score will be selected as the center and the pixels that voted on that center are considered inliers. The average depth of the inliers is used as the depth for the center pixel. The following equations are used to project pixels into 3D space: Depth is the depth of the pixel, x and y are the pixel

$$x' = (x - px) * depth/fx$$

$$y' = (y - py) * depth/fy$$

$$z' = depth$$

Figure (8) Equations (a) - (c) used to project a pixel coordinate to a 3D location

coordinates, px and py are the principal point coordinates and fx and fy are the focal lengths. The rotation estimation is done by a CNN that receives a RoI pooling map and outputs a quaternion. By combining the rotation and position, PoseCNN can position each object in a 3D scene.

A different approach to pose estimation relies on correspondences between images and 3D models. One example of this approach is OnePose. Onepose relies on a video scan for each supported object type. A sparse pointcloud is then reconstructed from the video using Structure from Motion. When a new image is captured, image features are matched with features of the point cloud. From these correspondences, the object pose relative to the camera can be reconstructed. This method does not require a lot of training data for a new object category, but does not always succeed in matching 2D features with 3D features due to textureless objects and thus may not be able to detect the pose.

3.4 Transfer learning

When building a CNN classifier or regressor, a common technique is to make use of transfer learning Joshi and Guerzhoy (2017). Transfer learning is the transferring of knowledge from one domain to a similar domain. This helps to improve the performance of the target domain Zhuang et al. (2019) as in Rothe, Timofte, and Van Gool (2015) and Joshi and Guerzhoy (2017). A widely known network architecture in machine learning is called VGG16, a CNN architecture used for object detection Simonyan and Zisserman (2014). It has originally been trained on the ImageNet dataset which contains 1000 different object categories Deng et al. (2009) and achieved state-of-the-art results. By using the trained weights from the VGG16 network as a starting point for a similar domain, the new model can train faster and achieve better results because it already starts with knowledge about the input. When the weights are randomised at the start of the training process, convergence can be a lot slower.

3.5 Depth estimation

In scene reconstruction, some techniques require the depth of an image to be known. Because this is not always the case, the depth has to be estimated with depth estimation techniques. Some techniques will be discussed here.

The depth can be estimated from mono or stereo images. Most of the techniques that use stereo images are based on point-matching between the left and the right image. These points are typically matched based on handcrafted or learned features Vasiljevic et al. (2019). This technique is used in Smolyanskiy, Kamenev, and Birchfield (2018),Scharstein and Szeliski (2002), and Flynn, Neulander, Philbin, and Snavely (2016) among others. Next, a technique that uses a single image will be discussed. AdaBins (Adaptive Bins) addresses the estimation of a depth map given a single RGB image. A drawback of convolutional layers is that they, only after the tensors reach a low spatial resolution, process global information. AdaBins fixes this issue by refining the output with a learned post-processing building block that processes information at the highest resolution. Firstly, it predicts a minimum and maximum depth for the scene and divides the depth range into adaptive bins. The amount and width of bins is calculated based on the input image. Secondly, it calculates the final depth values as linear combina-

tions of the bin centers. This approach has some advantages. A classification approach instead of a regression approach improves performance Fu, Gong, Wang, Batmanghelich, and Tao (2018). Also, the linear combinations result in smooth transitions between pixels in contrary to assigning a single bin to each pixel. An overview of AdaBins' architecture is depicted in Fig 9.

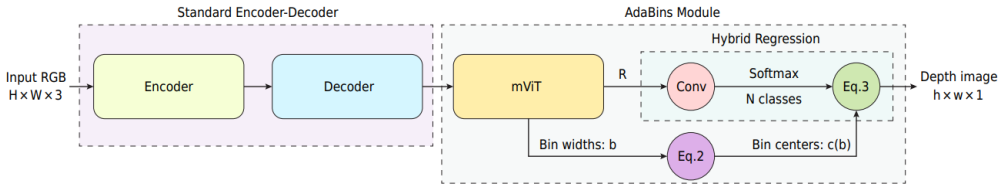

Figure (9) Architecture overview of depth estimation by AdaBins

It consists of 2 main components: an encoder-decoder and the AdaBins module. The encoder is built on a pre-trained encoder called EfficientNet B5. The decoder is a standard feature upsampling decoder. Mini-ViT is a vision transformer that estimates sub-intervals, called bins, that are more likely to occur within the depth range. The output embeddings from this transformer are convolved with decoded features to obtain range attention maps, indicated by R in the overview. Softmax is applied to these range attention maps. This leads to a value for each pixel. The final depth values are calculated from the linear combination of the pixel values and the depth-bin-centers Bhat, Alhashim, and Wonka (2021). Some results of AdaBins are shown in Fig 7. A follow up on this technique is BinsFormer Z. Li, Wang, Liu, and Jiang (2022).

4 Process Pipeline

The process of converting an RGB image to a 3D scene consists of combining several models that each solve a part of the problem. An overview of the process pipeline is depicted in Fig 10. Firstly, instance segmentation is applied to the image. This results in a mask and box for each detected object in the image. Secondly, the resulting boxes are provided as input to the rotation estimation model, which estimates the orientation of each detected object. Thirdly, the 2D location of each object in the image is converted to a 3D location by projecting a pixel that represents the object to 3D space. The information from all 3 steps is combined to retrieve models from a database and position them correctly in a virtual 3D scene.

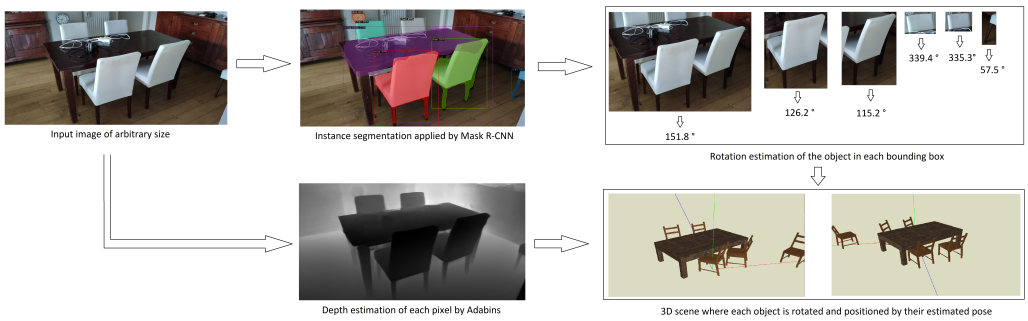

Figure (10) An overview of the process pipeline

4.1 Instance Segmentation

Mask R-CNN, explained in chapter 3.2, achieves state-of-the-art results and is easily extendable with custom datasets. There are already multiple implementations of the Mask R-CNN architecture that are publicly available. One of these implementations is implemented by Facebook's research team. It is called Detectron2[1], the successor of Detectron. It has already been pretrained on the COCO dataset[2] which contains 123,287 images with ground truth instance segmentations of 80 different object categories. Amongst these categories are chairs, sofas, tables etc. The extensibility is useful to make it learn instance segmentation on objects that are not part of the COCO

[1]Detectron2 Github Repository `github.com/facebookresearch/detectron2`
[2]COCO Dataset `cocodataset.org`

dataset. Detectron2 will be used in this research.
In Fig 11, the result of Mask R-CNN being applied to an indoor scene is shown. This shows that it is able to detect every distinct instance of each furniture object. The masks are also quite accurate, but exhibit some errors in rare occasions. The box surrounding each object is used to cut out an object and provide it as input to other models. For furniture types that usually do not have many occlusions, the mask is used instead of the whole content of the box. The reason for this is that masks of partially occluded objects often exhibit errors. The masks are also used for choosing a location that represents the object as explained in 4.4.

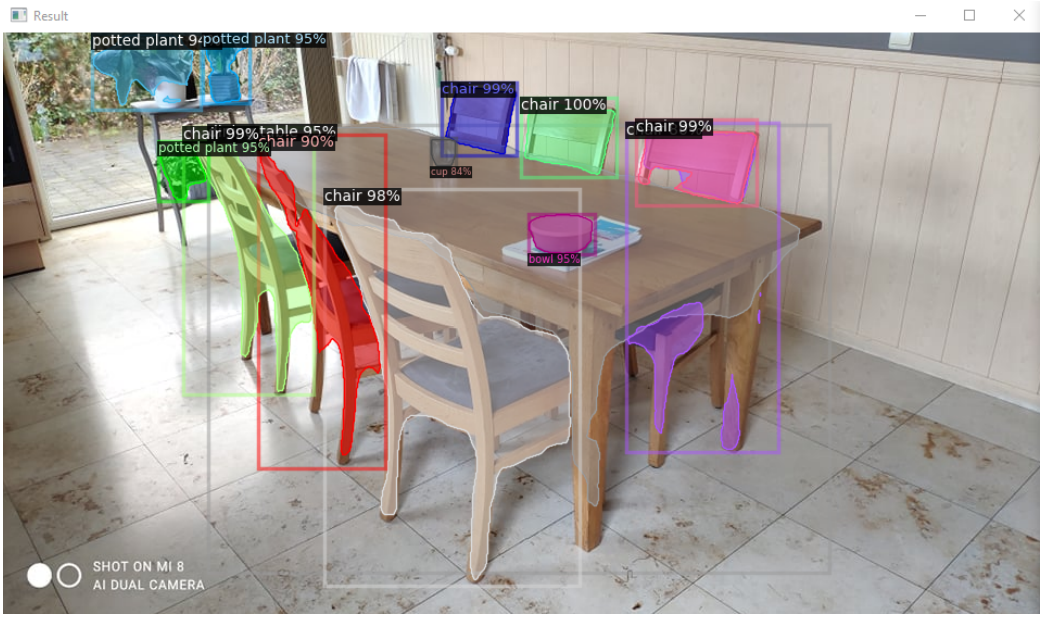

Figure (11) An example of Mask R-CNN being applied to an image of an indoor scene

4.2 Pose Estimation

A sub-problem of this process/pipeline is called pose estimation. The pose of an object is composed of a location and an orientation. There is also a dimension attached to the pose. If both location and orientation are 3D, then the pose of interest is a 6D (or DoF) pose. The pose describes how the object is transformed from a local coordinate system of the object itself,

to a reference coordinate system. An example reference coordinate system is that of a camera. If for example the 3D location is (1,2,0), then the object is translated 1 unit in the x direction and 2 units in the y direction starting from the camera position. In the context of my research, both the 3D location of furniture pieces and the rotation around the up axis are of interest. Other rotational axis will not apply to most furniture. The reason for this is that furniture generally stands upright. Four values of interest means that this part can be called 4D pose estimation. The approach here is similar to PoseCNN, explained in chapter 3.3, but is slightly modified to fit the objectives of this thesis. A CNN that is trained in a supervised way to estimate the rotation of a furniture piece image is used to estimate the 1D rotation. To estimate the 3D location, the CNN architecture proposed by AdaBins is used in combination with the equations used in Fig 8. Both are explained in the following sections.

4.3 Rotation estimation

The input images for this stage are bounding boxes provided by Mask R-CNN. Given an input image and a bounding box, this stage tries to predict the rotation of the furniture piece in the image bounding box. Because only the rotation around the up axis is important for furniture placement, that value will be the output of this stage. Because chairs are one of the more difficult furniture pieces to predict properties of, the architecture is based on predicting the rotation of chairs but also works on other furniture types. Chairs in images are often occluded by other objects (most likely by a table). That is why the model must be able to predict a chair's rotation while only given parts of the chair as input. Other furniture pieces like wardrobes or tables suffer less from problems like this one. Wardrobes are a lot bigger so occlusions will most likely only occlude a small part of its full shape. Tables exhibit another feature that makes it a lot easier to predict their rotation. They are most likely symmetrical, meaning that angle predictions only have to cover the first 180 degrees.
Because we already know the furniture type of each mask (given by MASK R-CNN), we can train separate weights for each type of furniture. That way, the accuracy will be higher for each type. The downside of this approach is that a new set of weights has to be loaded into memory for each supported furniture type.
Transfer learning is also applied to the chair's (and other furniture types)

rotation estimation. By using the weights of the VGG-16 network that has been pre-trained on the ImageNet dataset, the model already knows what the features of a chair look like. That means it does not have to relearn those features all over again. It just needs to modify the weights of the final layers to adapt to the new outputs. Transfer learning greatly improved the accuracy and training speed of the rotation estimation models because it obtained a high accuracy much sooner. To have an image dataset with ground truth rotation data that the model can use to learn the rotations a dataset has to be constructed. The dataset generation process is explained in chapter 5.

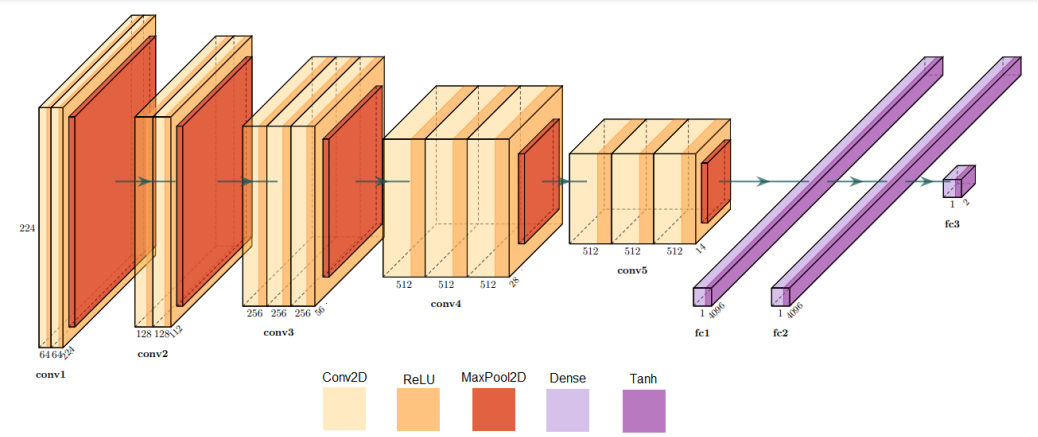

Figure (12) Rotation estimation CNN architecture

4.3.1 Model architecture

An overview of the model architecture used for rotation estimation is depicted in Fig 12. It is a variation of VGG-16, a state-of-the-art CNN architecture used for object detection Simonyan and Zisserman (2014). The accuracy of this model is explained in chapter 6.4. The dataset generation process will be explained in chapter 5.

It requires a 224x224 image as input and outputs 2 values: a sin and a cos. Loss functions like mse (mean squared error) can not handle the cyclic nature of the angle if the angle would be represented by a single value. They will output a high loss value when the predicted angle is 355 degrees and the ground truth angle is 5 degrees, while the actual rotation difference is very small (10 degrees). To cope with this cyclic nature, the 2 cyclic functions

sin and cos are used. Both can be used separately to predict angles ranging from 0 to 180. When combining both sin and cos, angles ranging from 0 to 360 can be used. As depicted in Fig 13, both combined form unique combinations on the interval [0,360] or [0,2*π]. In contrary to representing angles

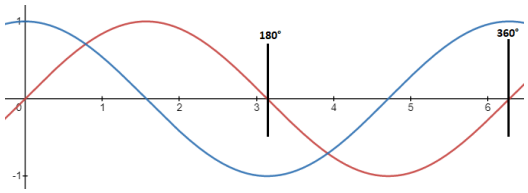

Figure (13) Sin (blue), Cos (red)

by a single value ranging from 0 to 360, cos and sin combinations after 360 degrees repeat. This makes the angle representation appropriate for a loss function and for training in general. After a prediction, we can retrieve the predicted angle (θ) by the following formula: θ = arctan2(predicted sin, predicted cos). This output is later used to rotate the 3D model that represents the furniture piece provided as input to this architecture.

4.4 Location estimation

The goal of this stage is to place each detected furniture piece in the correct location in a virtual 3D scene. This is done by calculating the depth of each pixel and the direction in which to project that pixel.
To calculate the depth of each pixel, the AdaBins architecture, explained in chapter 3.5, is used. This model achieves state-of-the-art results.
The direction estimation of a pixel is based on a couple of intrinsic camera parameters. Cameras have extrinsic and intrinsic parameters. Extrinsic parameters describe the pose of the camera. This consists of a location and a rotation with respect to the world frame. These are not available to us when a random image is selected by a user. Intrinsic parameters describe many features of the camera while the image was taken. Examples of intrinsic parameters are the camera's focal length, whether the flash was on or not etc. These can be roughly estimated from an image, or extracted from the image metadata. Intrinsic parameters also allow a mapping between pixel coordinates and camera coordinates Komorowski and Rokita (2012). The parameters of interest for the location estimation of objects in an image are the focal length and principal point. Once these parameter have been obtained,

they can be used to project pixels to a 3D location. Given a pixel coordinate, its depth, the focal length and the principal point, the 3D location can be calculated using the equations from Fig 8.

Once an object has been detected in an image and a depth map is estimated from that image, a pixel that represents the object's location has to be chosen and the corresponding depth value has to be used to place a corresponding 3D model in the scene. Some parts of objects might also be occluded by other objects. A chair will often be partially occluded by a table. This might lead to the center object pixel not being part of the detected object, but rather a part of the object that occludes the detected object. An example is depicted in Fig 14, where the center of the detected chair is part of the table. If the

Figure (14) Center of chair occluded by table

center pixel would be chosen here, the depth would match the depth of the table instead of the depth of the chair. That would lead to bad positioning of the chair. To combat this issue, the mask outputted by Mask R-CNN is used. First, the average pixel position of the mask is calculated. Next, the pixel in the original image that is part of the mask and is closest to the average position is chosen. This results in a pixel close to the center of the object.
Now that we have obtained a pixel that can be converted to a 3D location, the 3D model has to be placed at that position. Ideally, not the center of the 3D model would be placed at that position, but rather the part of the model that corresponds to the chosen pixel in the image. For example, if the

center pixel in the back of the chair is chosen, then the center vertex in the back of the chair model should be placed at that position. Another easier method is placing the center of the 3D model at that position. This will still achieve accurate results, but they will almost always be a tiny bit different from the "perfect" result. Another way that achieves slightly better results is explained next.

We are trying to achieve the ideal placement, where the part of the 3D model, that corresponds to the chosen pixel in the 2D image, is placed at the location of the chosen pixel. This is depicted in Fig 15, where they should both be placed at the 3D location (4,3,8). This can be achieved by first load-

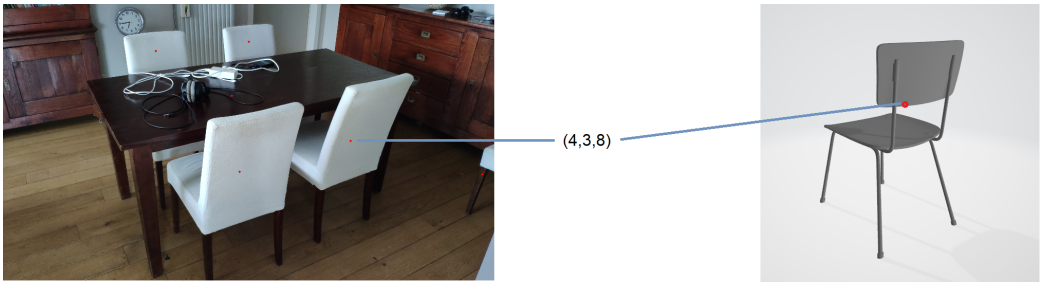

Figure (15) Ideal location correspondence

ing the 3D model into memory. Next, the model's vertices should be rotated to match the rotation of the detected object. Because rotation estimation precedes this step, the rotation is already available at this stage. Now a ray can be casted from the camera's location to the rotated 3D model to see which vertex the ray intersects with or which vertex is the closest to the ray if none intersect. An offset is added to the end location of the ray. This corresponds to the offset of the chosen pixel inside it's bounding box. After finding that point, we can place that point of the 3D model instead of the model center at the location that we have calculated earlier.
A more intuitive example is shown in Fig 16 and explained next. The right part of the figure shows the chair and it's estimated bounding box by Mask R-CNN. The red dot indicates the chosen pixel, which is located in the back of the chair. This is also 38 pixels to the right and 40 pixels up from the center of the bounding box. Given this data, we need to select a vertex of the chair model that corresponds to the pixel in the back of the chair. We do this by estimating the 3D location of the pixel and taking the inverse of this location to get the camera location compared to the vertex. We cast a ray

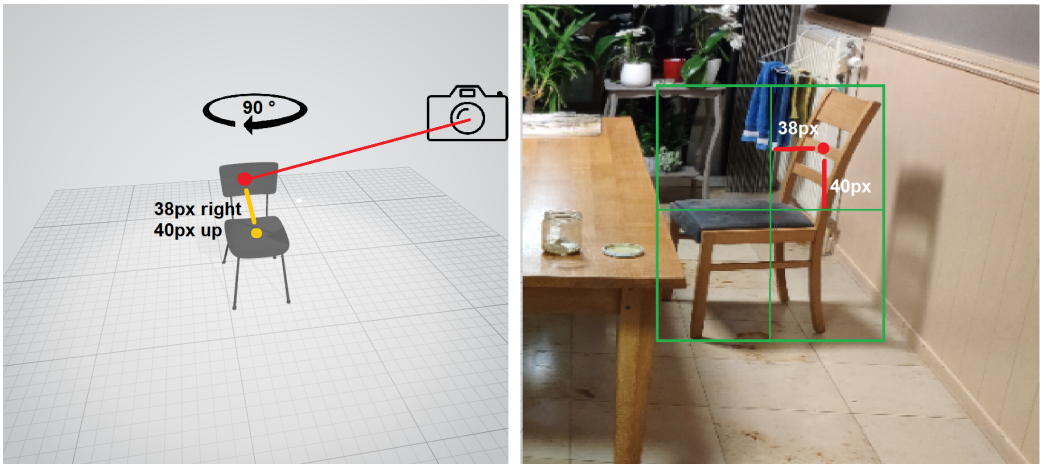

Figure (16) Ideal location visualisation

from this location to the chair model, which is rotated 90 degrees, because the chair is also rotated 90 degrees in the image. The end location of the ray is the center of the chair (indicated by the yellow dot) offsetted by 38 pixels to the right and 40 pixels up, because the red dot is also offsetted by this amount from the center.
In Fig 17 is shown how this technique achieves a better result compared to placing the center of the model at the calculated location. The green models

Figure (17) Effect of using the ray cast technique

are those placed by their center, while the others are the ones placed with the special technique. This shows that the special technique leads to a better result. For example, the red dot of the green table leads to the center of the 3D table being placed at the location of the red dot. Actually, the front of the table should be placed at that location. We achieve this result by moving

the table backwards. The same goes for the chairs: the red dot is at the back of the chairs, so the chairs should be moved to the front.

After having explained the pipeline in-depth, the dataset generation process will be explained next. This contains information about the dataset used to train the rotation estimation model. The results of each individual step and the pipeline as a whole will be discussed in chapters 6,7 and 8.

5 Synthetic Dataset Generation

There exist many variations of indoor scenes and furniture pieces have different poses in each scene. To be able to predict the rotation of objects in an image, much data is needed to cope with the many variations and furniture types. There are not many 2D image datasets publicly available that have been annotated with rotation data. Furthermore for this research, specific furniture types are needed (chairs, wardrobes, tables, etc.).
A solution to this problem is to generate synthetic datasets that match the requirements of this research. One commonly used way to generate image datasets is rendering images from 3D models. This way, the image can be annotated with pose data because the position and rotation is known while the image is being rendered. There exist many tools that are able to load 3D models and render images from these models. Some examples are Blender [3], Unreal Engine: [4] and Unity [5] among others. For this research, Blender will be used because it supports easy-to-use scripting with the python computer language. By using these render tools, the problem of finding datasets is shifted from finding annotated 2D image datasets to finding 3D model datasets. Because these do not have to be annotated, there are some publicly available.
The one that will be used for this research is called 3D-FUTURE [6]: 3D FUrniture shape with TextURE. This database contains around 17000 3D models with realistic textures, over 20000 photo-realistic synthetic images captures in 5000 diverse scenes and over 6000 scenes Fu et al. (2020). For this research, only the 3D models are used. The 3D models are annotated with a super-category, category, style, theme and material. An example annotation is:


```
{
    "model_id": "0d3e3b3c-3f1a-47ee-8566",
    "super-category": "Cabinet/Shelf/Desk",
    "category": "Coffee Table",
    "style": "Modern",
    "theme": "Texture Mark",
```


[3]Blender: blender.org
[4]Unreal Engine: unrealengine.com
[5]Unity: unity.com
[6]3D-FUTURE: https://tianchi.aliyun.com/specials/promotion/alibaba-3d-future

```
    "material": "Wood"
}
```

The super-categories are shown here:

```
{'id': 1, 'category': 'Cabinet/Shelf/Desk'},
{'id': 2, 'category': 'Bed'},
{'id': 3, 'category': 'Chair'},
{'id': 4, 'category': 'Table'},
{'id': 5, 'category': 'Sofa'},
{'id': 6, 'category': 'Pier/Stool'},
{'id': 7, 'category': 'Lighting'},
{'id': 8, 'category': 'Other'}
```

These are also the main categories that will be supported in this research. Because chairs suffer from many occlusions and have a more difficult shape than others, the model to predict the rotation of chairs will be trained first. The same model can then be applied to other furniture types as well.

5.1 Generation of the chair image dataset

As mentioned in the previous section, Blender will be used to generate chair images from 3D chair models. Blender scripting makes use of the python language and lets you edit all aspects of a 3D scene through code. After filtering out all the chairs from the 3D-FUTURE dataset, about 1228 models remain. The setup of the blender scene is depicted in Fig 18. A camera

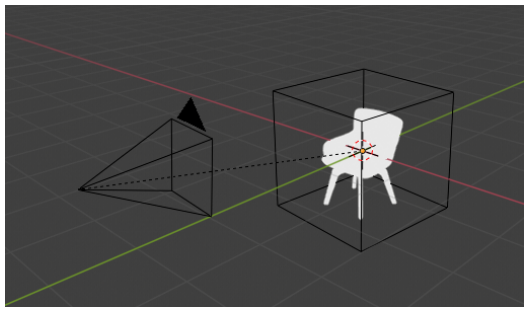

Figure (18) Blender setup

is attached to a box and each 3D model is added to the scene one by one.

After loading each chair, its texture is applied to it. For each chair the box is rotated from 0 to 360 degrees, causing the camera to rotate around the chair. This is the initial setup which has been expanded later on. At this stage, a decision has to be made about how many images to generate per model and thus how much to rotate the box each time. An interval of 5 degrees was chosen to generate a sufficient amount of data. This leads to 72 images for each chair starting from 0 and going up to 355. Each image's resolution is 256x256, which is about the input size of the rotation estimation model. In total, this results in a dataset consisting of 88416 images. The structure of the dataset is shown in Fig 19, where each folder contains the images corresponding to that rotation of the box/camera. This way, the dataset

Figure (19) Dataset setup

can easily be reviewed. The filename of each image contains the degree of the chair's rotation so the label of each image can be easily retrieved while reading the dataset in a program. A small subset of the dataset is shown in Fig 20. These image are stored under the folder named '125'. The rotation estimation results of a model applied to this dataset is described in chapter 6.1. The total time to generate this dataset was about 4 hours.

5.1.1 Dataset update: adding noise and occlusions

First of all, a background was added to each image. 20 different realistic backgrounds of indoor scenes were chosen. A random background was assigned to each image. Fig 21 depicts 6 images from this dataset. Secondly, instance segmentation was applied to each image and only the chair segmentations were cut out and re-scaled to a 256x256 image. The reason for keeping only the segmentations is that that would remove background clutter and thus improve the rotation predictions. This might seem contradictory to the addition of backgrounds. However, there are often holes in a chair (in the backrest or under the handrail) so the chair's segmentation will still contain the clutter from the added background images as a real (non-synthetic) image would. Lastly, multiple parts from each image were cut to take into account the many occlusions that chairs suffer from in images. This resulted

Figure (20) Snapshot of the chair dataset. Chairs are located in the folder named 125 because they are rotated 125 degrees.

in 4 images for each image. The original image, the image without the left part, the image without the right part and the image without the top part. The resulting dataset is shown in Fig 22. The rotation estimation results of a model applied to this dataset is described in chapter 6.3. The total time to generate this dataset was about 5 hours.

5.1.2 Dataset update: more realistic occlusions and lighting conditions

For this version of the dataset, some additions to blender's scene setup were made as shown in Fig 23. First of all, a spotlight and a sun were added. For each image, the light settings (brightness, color, location) were randomised to simulate many real world lighting conditions. Also a table has been added to the scene to simulate occlusions by a table. This way, no cuts have to be made later on as in the dataset explained in chapter 5.1.1. Because not all chairs are occluded by a table in real indoor environments, every chair is rendered in every rotation with and without the table. This doubles the size of the dataset, leading to a dataset of 162434 images.
Next, instance segmentation could be applied to each image. However, this

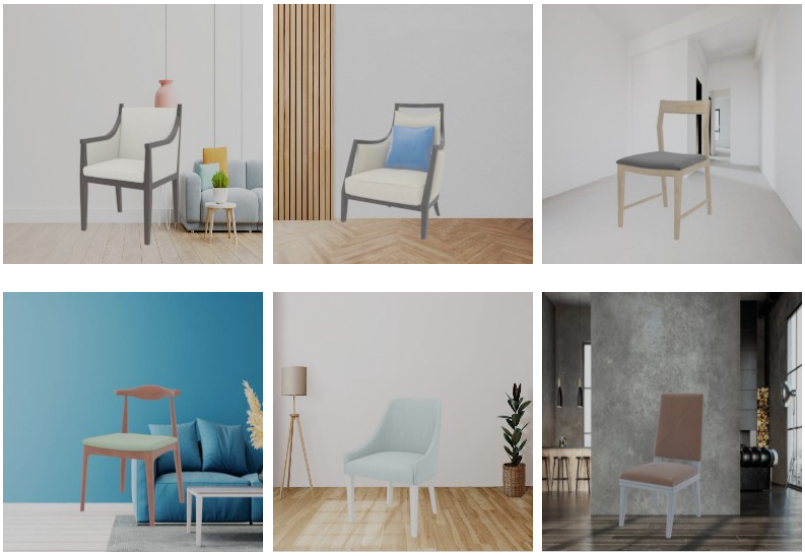

Figure (21) Chair with background

Figure (22) Chair with background and cuts example

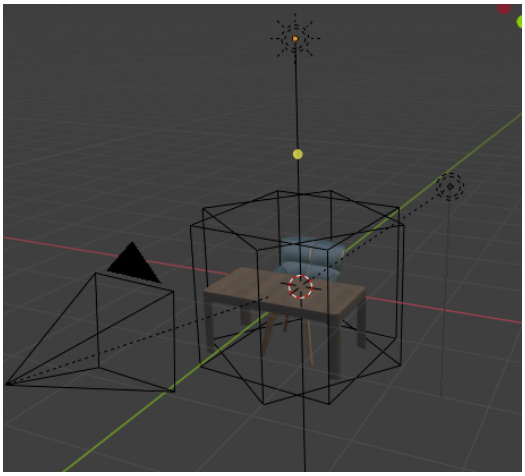

Figure (23) Blender setup with light and table

lead to segmentations with unrecognisable parts of a chair. Because the table blocks part of the chair, the mask is often only the bottom or top part of the chair and sometimes both with a gap in between as shown in Fig 24. This

Figure (24) Bad chair segmentation when occluded by table

behaviour is specific to chairs and does not apply to most other furniture types. An alternative is to use the predicted box contour of the predicted chair object in the image. This will keep all the clutter in the background, but will mostly still contain the full chair instead of small segmented parts. When the bottom or top half of the chair is occluded by the table, still only half the chair might be detected. In chapter 6.4 will be shown that this is not an issue. Using the masks from instance segmentation applied to other furniture types that suffer less from occlusions might lead to better rotation estimation accuracy compared to using every pixel inside the box contour. Depending on the detected object in the image, either the mask or the box

Figure (25) Final chair dataset

contour can be used. The resulting dataset is depicted in Fig 25. The rotation estimation results of a model applied to this dataset is described in chapter 6.4. The total time to generate this dataset was about 4.5 hours.

6 Results of rotation estimation

6.1 Model architecture

A CNN with the structure depicted in Fig 26 was trained on this dataset. The model parameters are shown in table 1. The dataset was split into 80% used for training and 20% used for testing. Note that this architecture is not the same as the final architecture from chapter 4.3.1. This architecture (from Fig 26) was used for each version of the dataset. Afterwards, transfer learning was applied and the architecture from section 4.3.1 was used. Everything was executed on the same computer with the specs listed in section 11.

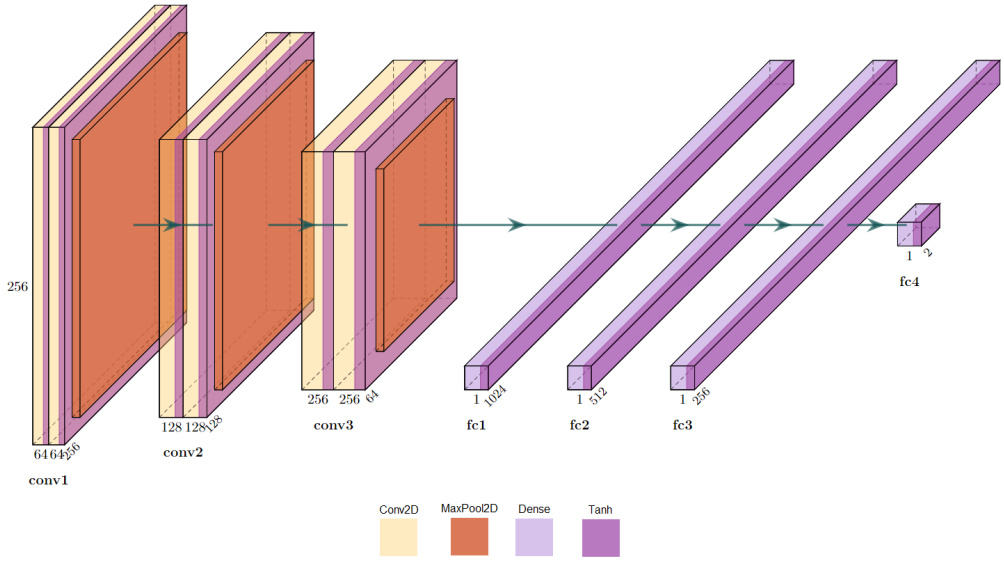

Figure (26) First version of rotation estimation CNN architecture

6.2 Results on the first dataset (no noise or occlusions)

The results of the architecture in Fig 26 being trained and tested on this dataset are summarised in table 2. Fig 27 depicts the distribution of errors. This is a great result but it does not generalise well to real images of chairs. The reason for this is that this dataset does not contain the realism (or noise) that normal indoor scene images would contain. This is further discussed in chapter 6.4.

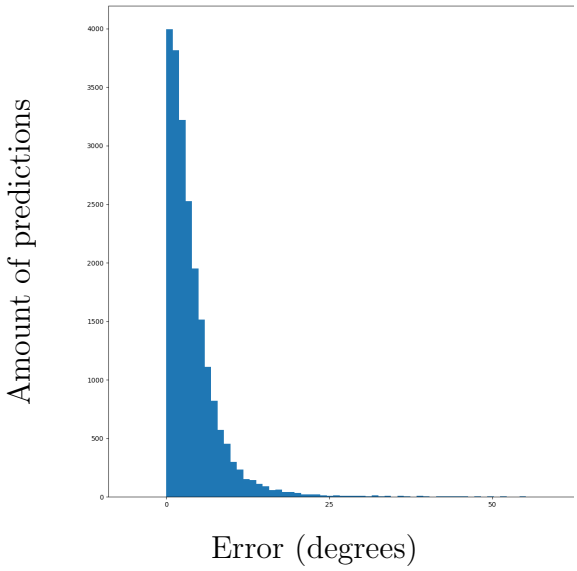

Figure (27) Accuracy Dataset 1

Loss	MSE
Optimizer	SGD
Learning Rate	0.015
Batch Size	32

Table (1) Model Parameters

Epochs	161
Training duration	60 hours
Validation loss	0.009
Average error	3.5 degrees

Table (2) Train and Test results of the first dataset

6.3 Results on the second dataset (noise and occlusions)

Again the same model from Fig 26 was trained on this dataset with the same parameters as in table 1. The results are summarised in table 3 and the error distribution is shown in fig 28 . Even though this dataset was partially more realistic, part of the dataset contained images that were unrecognisable

chairs because of the added cuts. Not enough information in these images were visible to identify a rotation. The idea to simulate occlusions can still be improved, which is explored in the following section.

Epochs	82
Training duration	33 hours
Validation loss	0.032
Average error	12.5 degrees

Table (3) Train and Test results of the second dataset (with noise and occlusions)

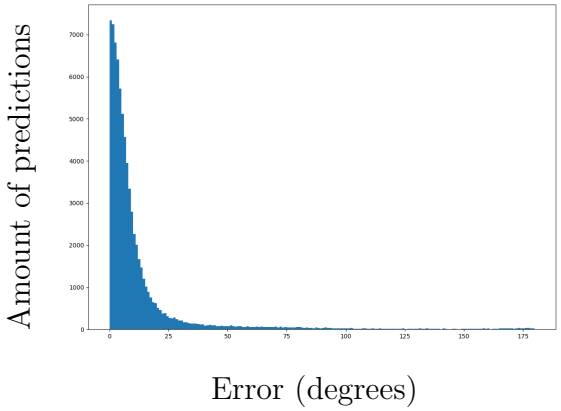

Figure (28) Error histogram of the model trained on the dataset explained in chapter 5.1.1

6.4 Results on the third dataset (noise, improved occlusions and lighting conditions)

The model from Fig 26 with the same parameters as in table 1 was trained on this dataset. The results of this process are summarised in table 4. An error histogram is shown in Fig 29. This is a pretty good result but it could still be improved by applying transfer learning. This time, a new model with the architecture illustrated in Fig 12 was trained and tested on this dataset. It achieved much better results, as summarised in table 5 and depicted in Fig 30 a. Also, the model that was trained on the first version of the dataset,

Epochs	48
Training duration	18 hours
Validation loss	0.024
Average error	6.4 degrees

Table (4) Train and Test results of the third dataset (with noise, occlusions and lighting conditions)

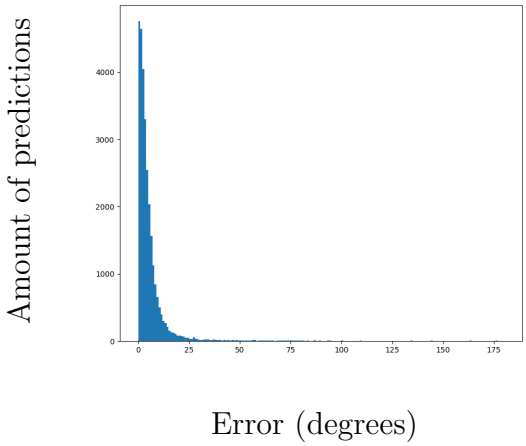

Figure (29) Error histogram of the model trained on the latest dataset explained in chapter 5.1.2

Epochs	24
Training duration	6 hours
Validation loss	0.009
Average error	3.5 degrees

Table (5) Results of VGG-16 variant trained and tested on the third dataset (with noise, occlusions and lighting conditions)

which did not contain backgrounds or a table, was also tested on this new dataset. The error of this test is depicted in Fig 30 b. This shows that the first model does not generalise well to more realistic images of indoor scenes, but the last model does while maintaining a very low error.

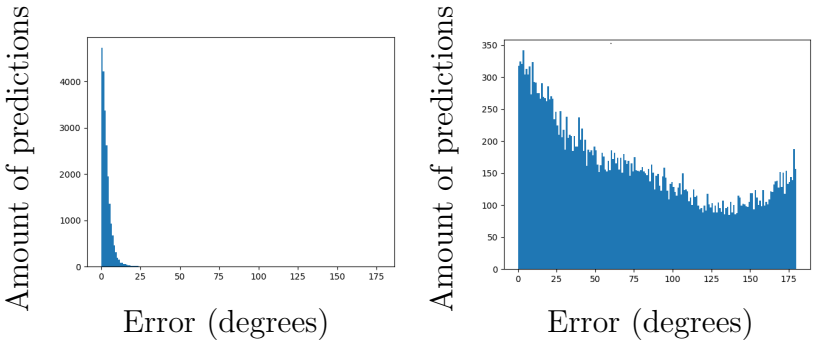

Figure (30) (a) Error histogram of the latest model making predictions on the latest dataset (b) Error histogram of the first model making predictions on the latest dataset

7 Results of location estimation

The location estimation will be discussed here based on some results. Recall from chapter 4.4 that we use the depth map estimated by AdaBins to project pixels to a 3D location. An example of a location estimation is shown in Fig 31. In this case, Mask R-CNN was not able to detect the farthest left chair. However, it detected a chair in the couch. The red dots indicate the chosen pixel that is projected into 3D to obtain a location for that object. The depth map, estimated by AdaBins, also gives a quite accurate depth representation. The resulting 3D locations for each object are depicted in Fig 31 d and visualised by a green sphere. Comparing this with the original image, it is a good result. Next, this step in combination with the rotation estimation will be discussed.

8 Results of the whole pipeline

The results of all steps combined are discussed here. The user interface itself will be discussed in chapter 9. Fig 32 depicts 3 results with input images from real indoor scenes, which will now be discussed.

Figure (31) Location estimation example. (a) The original image. (b) The pixels that represent each object's location. (c) The estimated depth map. (d) The locations in 3D.

Figure (32) Results: Row 1 are the original images. Row 2 are the detected masks by Mask R-CNN. Row 3 are the estimated depth maps by AdaBins. Row 4 are the reconstructed 3D scenes.

In the first example, the rotation and placement of each object is pretty accurate. Only the rotation of the closest 2 chairs is off by about 15 degrees. The rotation of the chairs that are behind the table are very accurate, which means that the model can handle occlusions well.
In the second example, the chair and tables' rotation are also very accurate. Occlusions are again handled well and clutter (on and around the table) does not mess up the rotation estimations.
In the third example, a couple of mistakes were made while estimating the chairs' rotations. The left 2 chairs are both rotated differently even though they both face the same direction. A reason for this might be that the many holes in the chair causes the model to extract the wrong features from the chair and therefore make worse predictions. The rotation and location of the table and the right chair are pretty accurate. In all cases, Mask R-CNN was

able to detect every occurance of chairs and tables and AdaBins was able to estimate a pretty accurate depth map.

9 User Interface

In the center of the screen, the 3D scene is shown which is the result of the conversion from 2D to 3D. At the right of the screen is a panel with 2 buttons located, 1 for each furniture type in the image. By clicking one of these buttons, a user can select a new model for that furniture type. Because we do not have a 3D model for the actual furniture models in the image, a wooden chair and table is always selected and loaded by default. After clicking one of the buttons, the user is presented with the screen shown in figure 33 a. Here, the user can choose which style the model should be. After choosing a style, the user is again presented with another menu, shown in figure 33 b. Now the user can choose which material the model should be made of. If the user wants to change chairs and selects American Country

Figure (33) a) Style select menu b) Material select menu

as the style and Rough Cloth as the material, the user is presented with the chairs shown in Fig 34

Figure (34) Chairs in the style of American Country and of material type Rough Cloth

10 Limitations and Scene recommendations

There are a couple of things that the application has trouble with. One of those is Mask R-CNN not being able to detect some furniture types like tables in low lighting conditions. One recommendation is to always provide enough light for the entire scene when taking a picture. Another difficulty is the rotation estimation of chairs with many holes of different shapes. This is probably the reason why the rotation of the 2 left chairs in the third result of Fig 32 is a bit off. Also mirrors in a scene will cause the program to make mistakes, as it can not tell the difference between furniture types that occur in a mirror and those that do not.

11 Computer specs and Implementation details

The hardware specifications are summarised in table 6. Everything is written

Processor	Intel(R) Core(TM) i5-8500 CPU @ 3.00GHz
RAM	16.0 GB DDR4
System type	64-bit operating system, x64-based processor
Operating system	Windows 10 Education

Table (6) Computer hardware specifications

in the python computer language. For all the neural networks, the module Keras from the machine learning library TensorFlow is used. The rendering of the 3D models is done with PyOpenGL, an OpenGL library for python. The interactive User Interface is built with the cross-platform GUI toolkit called wxPython, which allows OpenGL to be easily combined with UI. To generate the datasets, python in blender (bpy) is used, which is python with custom functions that allow making modifications to the scene.

12 Future Work

One of the more obvious things to add is the support of more furniture types. For demonstration purposes, only chairs and tables have been used in this research, but the ideas can easily be extended to other furniture types like lamps, wardrobes, beds, etc. Also adding walls to the scene can be added. This would require the detection of walls in an image. After a wall is added to the 3D scene, a user would be able to change the texture/color of the wall to see how that would look like. The whole idea could also be modified to support Augmented Reality-based scene modifications.

13 Conclusion

In the context of my research, an application was developed that is able to convert an image to a 3D scene. Further more, this application allows the user to swap furniture models for new models in a different style.

A pipeline of machine learning models and other algorithms was constructed to convert an image to a 3D scene, known as 3D scene reconstruction. Firstly, instance segmentation was applied to the image. This was done by the Mask R-CNN framework which was pre-trained on the COCO dataset. Instance segmentation detects each furniture object in the image together with their masks. Secondly, each detected object was provided to a rotation estimation model that estimated the furniture's rotation relative to the camera using a CNN derived from the well-known VGG-16 architecture. Thirdly, a depth map of the input image was estimated by the AdaBins depth estimation model. Lastly, the depth was used to convert each pixel to a 3D location. The rotations and locations were combined to place a 3D model at the estimated location with the estimated rotation.
This scene is shown to the user in a user interface, where the user can swap furniture models for models in different styles.

The scene reconstruction achieves pretty accurate results in most cases, but can make mistakes in low lighting environments or when estimating the rotation of chair models with many holes. There are a couple of improvements and additions that can still be made, like adding support for walls and other furniture types. I hope to inspire future researchers to extend this work and discover it's full potential in many use cases.

During this research, I learned that it can be very hard to reach optimal results when training a CNN regressor. It needs a lot of hyperparameter tuning and this can take days. I also learned that it is not easy to combine many models and algorithms that each have a small error and still obtain an accurate result. In general, I learned a lot of new information about machine learning.
In this research, I partially solved the problem of converting an image to a 3D scene. There are still some aspects that I wanted to implement but did not have enough time for, like the support for walls and ways to make the initial models of the scene resemble the real models more. My method

during this research was to first build the general model pipeline structure, and afterwards go into the details of each part. I learned about myself that this was the best way for me to estimate how much work there is still left to do.

References

Bhat, S. F., Alhashim, I., & Wonka, P. (2021, jun). AdaBins: Depth estimation using adaptive bins. In *2021 IEEE/CVF conference on computer vision and pattern recognition (CVPR).* IEEE. Retrieved from https://doi.org/10.1109%2Fcvpr46437.2021.00400 doi: 10.1109/cvpr46437.2021.00400

Dai, J., He, K., Li, Y., Ren, S., & Sun, J. (2016). *Instance-sensitive fully convolutional networks.* arXiv. Retrieved from https://arxiv.org/abs/1603.08678 doi: 10.48550/ARXIV.1603.08678

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 ieee conference on computer vision and pattern recognition* (p. 248-255). doi: 10.1109/CVPR.2009.5206848

Denninger, M., & Triebel, R. (2020a, 11). 3d scene reconstruction from a single viewport. In (p. 51-67). doi: 10.1007/978-3-030-58542-6_4

Denninger, M., & Triebel, R. (2020b, 11). 3d scene reconstruction from a single viewport. In (p. 51-67). doi: 10.1007/978-3-030-58542-6_4

Flynn, J., Neulander, I., Philbin, J., & Snavely, N. (2016). Deepstereo: Learning to predict new views from the world's imagery. In *The ieee conference on computer vision and pattern recognition (cvpr).* Retrieved from http://www.cv-foundation.org/openaccess/content_cvpr_2016/html/Flynn_DeepStereo_Learning_to_CVPR_2016_paper.html

Fu, H., Gong, M., Wang, C., Batmanghelich, K., & Tao, D. (2018). *Deep ordinal regression network for monocular depth estimation.* arXiv. Retrieved from https://arxiv.org/abs/1806.02446 doi: 10.48550/ARXIV.1806.02446

Fu, H., Jia, R., Gao, L., Gong, M., Zhao, B., Maybank, S., & Tao, D. (2020). *3d-future: 3d furniture shape with texture.* arXiv. Retrieved from https://arxiv.org/abs/2009.09633 doi: 10.48550/ARXIV.2009.09633

Gao, G., Lauri, M., Zhang, J., & Frintrop, S. (2018). Occlusion resistant object rotation regression from point cloud segments. *CoRR*, *abs/1808.05498*. Retrieved from http://arxiv.org/abs/1808.05498

Goldman, R. (2011, 03). Understanding quaternions. *Graphical Models*, *73*, 21-49. doi: 10.1016/j.gmod.2010.10.004

He, K., Gkioxari, G., Dollár, P., & Girshick, R. B. (2017). Mask R-CNN. *CoRR*, *abs/1703.06870*. Retrieved from http://arxiv.org/abs/1703.06870

Joshi, U., & Guerzhoy, M. (2017, 05). Automatic photo orientation detection with convolutional neural networks. In (p. 103-108). doi: 10.1109/CRV.2017.59

Komorowski, J., & Rokita, P. (2012, 09). Extrinsic camera calibration method and its performance evaluation. In (p. 129-138). doi: 10.1007/978-3-642-33564-8_16

Li, Y., Qi, H., Dai, J., Ji, X., & Wei, Y. (2016). *Fully convolutional instance-aware semantic segmentation.* arXiv. Retrieved from https://arxiv.org/abs/1611.07709 doi: 10.48550/ARXIV.1611.07709

Li, Z., Wang, X., Liu, X., & Jiang, J. (2022). *Binsformer: Revisiting adaptive bins for monocular depth estimation.* arXiv. Retrieved from https://arxiv.org/abs/2204.00987 doi: 10.48550/ARXIV.2204.00987

Pinheiro, P. O., Collobert, R., & Dollar, P. (2015). *Learning to segment object candidates.* arXiv. Retrieved from https://arxiv.org/abs/1506.06204 doi: 10.48550/ARXIV.1506.06204

Ren, S., He, K., Girshick, R. B., & Sun, J. (2015). Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, *abs/1506.01497*. Retrieved from http://arxiv.org/abs/1506.01497

Ritchie, D., Wang, K., & an Lin, Y. (2018). *Fast and flexible indoor scene synthesis via deep convolutional generative models.*

Rothe, R., Timofte, R., & Van Gool, L. (2015, 12). Dex: Deep expectation of apparent age from a single image.. doi: 10.1109/ICCVW.2015.41

Scharstein, D., & Szeliski, R. (2002, 04). A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, *47*, 7-42. doi: 10.1023/A:1014573219977

Seitz, S., & Dyer, C. (1997). Photorealistic scene reconstruction by voxel coloring. In *Proceedings of ieee computer society conference on computer vision and pattern recognition* (p. 1067-1073). doi: 10.1109/CVPR.1997.609462

Simonyan, K., & Zisserman, A. (2014). *Very deep convolutional networks for large-scale image recognition.* arXiv. Retrieved from https://arxiv.org/abs/1409.1556 doi: 10.48550/ARXIV.1409.1556

Smolyanskiy, N., Kamenev, A., & Birchfield, S. (2018). *On the importance of stereo for accurate depth estimation: An efficient semi-supervised deep*

neural network approach. arXiv. Retrieved from https://arxiv.org/abs/1803.09719 doi: 10.48550/ARXIV.1803.09719

Song, S., Yu, F., Zeng, A., Chang, A. X., Savva, M., & Funkhouser, T. (2017). Semantic scene completion from a single depth image. *Proceedings of 30th IEEE Conference on Computer Vision and Pattern Recognition*.

Stefan, P., Pablo, B., & Vittorio, F. (2020). Corenet: Coherent 3d scene reconstruction from a single RGB image. *CoRR*, *abs/2004.12989*. Retrieved from https://arxiv.org/abs/2004.12989

Vasiljevic, I., Kolkin, N., Zhang, S., Luo, R., Wang, H., Dai, F. Z., ... Shakhnarovich, G. (2019). DIODE: A Dense Indoor and Outdoor DEpth Dataset. *CoRR*, *abs/1908.00463*. Retrieved from http://arxiv.org/abs/1908.00463

Xiang, Y., Schmidt, T., Narayanan, V., & Fox, D. (2017). *Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes.* arXiv. Retrieved from https://arxiv.org/abs/1711.00199 doi: 10.48550/ARXIV.1711.00199

Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., ... He, Q. (2019). *A comprehensive survey on transfer learning.* arXiv. Retrieved from https://arxiv.org/abs/1911.02685 doi: 10.48550/ARXIV.1911.02685

Appendix

13.1 Summary

Introduction
The need for virtual versions of real world scenes keeps increasing. Designing these with 3D modelling tools requires skill and is time-consuming. It would be more beneficial to automatically generate these with machine learning models, because it can happen fast, accurate and requires no skill. There are many applications that require these virtual scenes. An example is the need for virtual scenes for autonomous robots to learn how to drive in a realistic environment. Another example is in the architecture domain. When people want to redesign their living space and want to try different style furniture, they often do not know if it fits their current living space. Maybe it does not fit their wall color or the style of a table. It would helpful if there was a tool available that would just take an image of the scene as input and output a virtual version of that scene. This would allow the user to make modifications to that scene. There already exist many ways to convert one or multiple images to 3D scenes. By focusing on the redesign of a living space, a user-friendly prototype is developed that can convert an image to a virtual scene. By only using a single image as input, it is as user-friendly as possible. Specifically, the application should be able to display the 3D scene and allow modifications to it, like swapping furniture for other furniture pieces in different styles. To achieve these goals, a pipeline of machine learning models and other algorithms is constructed which will be explained next.

Process pipeline
A pipeline is constructed that converts an RGB image to a 3D virtual scene. It consists of multiple models and algorithms that each solve part of the problem.
First of all, instance segmentation is applied to the image. This is done by Mask R-CNN, an instance segmentation architecture that achieves state-of-the-art results. Instance segmentation is the segmentation of an image, where each segmented part is also annotated with a class label. The implementation of Mask R-CNN that is used in this research is called Detectron2, the successor of Detectron. It has been pre-trained on the COCO dataset, which already contains many of the important furniture categories like chairs

and tables. Both the mask and the box of a detected object is used as input to other steps.

The second step is pose estimation. This pose consists of a location and an orientation. It describes how an object is positioned relative to the camera (of the smartphone). For this prototype, both the location and the rotation around the up axis are of interest. The other rotational axis are usually not important because furniture stands upright.

The rotation estimation is done by a Convolutional Neural network (CNN), that receives the image part inside a bounding box from Mask R-CNN and outputs a rotation around the up axis. Furniture pieces like chairs are often occluded by tables or other objects. That is why the CNN should be able to estimate rotations given many variations and parts of furniture. Transfer learning is also applied to the rotation estimation. By starting from the VGG-16 network architecture and modifying it to fit our own output, the model trains a lot faster and reaches higher accuracy. The reason for this is that the weights of the pre-trained VGG-16 network already contain information about the features of a chair and a table. These only have to be mapped to a new output. The model used requires a 244x244 input image and outputs 2 values: a sin and a cos. The reason why it does not just output a single value (the angle in degrees) is that an angle of 355 degrees is very close to an angle of 5 degrees. This cyclic nature requires output values that are cyclic in nature, like cos and sin. Together they form angles ranging from 0 to 360 degrees.

The location estimation tries to place each object at the correct location in the virtual scene. To is done by choosing a single pixel for each object. Next, that pixel is projected into 3D by using the depth of that pixel and the equations from Fig 36. To get the depth of each pixel, we need a way to estimate a depth map from the original image. The AdaBins depth estimation architecture is used for this, which achieves state-of-the-art results. Also some intrinsic camera parameters are used. Fx and fy are the focal lengths while px and py are the principal points. These can be extracted from the metadata and also roughly estimated.
Once the 3D location of a pixel representing an object has been obtained, the object has to be placed at that location. Now the decision of which part of the 3D model should be placed at that location has to be made. This

$$x' = (x - px) * depth/fx$$

$$y' = (y - py) * depth/fy$$

$$z' = depth$$

Figure (35) Equations (a) - (c) used to project a pixel coordinate to a 3D location

should be the vertex that corresponds to the selected pixel. The selected pixel is a pixel located near the center of the object. To find the vertex of the model that corresponds to this pixel, a ray is casted from the camera's location (calculated earlier) to the 3D model, which is rotated as it appears in the scene. The end location of the ray is offsetted by the same amount as the selected pixel is offsetted from the bounding box center. By using this special technique, the correct vertex can be found and this can be placed at the calculated location.

Synthetic Dataset Generation

To be able to train a CNN that can estimate the rotation of an object in an image, training data is needed. Ideally, this is real labelled data, but as that was not available during this research, synthetic data was generated. By generating images from a 3D model dataset, a labelled image dataset was constructed. This dataset should be as realistic as possible. To generate the dataset, Blender was used. Blender allows scripting with python to make modifications to the scene. By looping through each model, importing it to the scene and rotating the camera around it, many images were generated and stored with the camera's rotation at that point. To collect enough training data, an interval of 5 degrees was chosen. That means that the camera rotates 5 degrees and renders an image 72 times for each chair.

There were multiple versions of the dataset, each slightly improving the previous one. The first dataset contained images of chairs rotated with no background or any noise. This lead to accurate predictions with an average error of 3.5 degrees. However, this trained model does not generalise well to noisy data from the real world, which is why a new dataset was constructed. In this dataset, each image contained a random realistic background to add

more noise to the image. Also each image was segmented by Mask R-CNN to remove as much background as possible while providing an image as input to the rotation estimation model. Lastly, multiple parts were cut from each image to replicate the occlusions that chairs suffer from. The model trained on this dataset did not achieve high accuracy (average error of 12.5 degrees), due to there being many unrecognisable chair images. The cut idea still had to be improved. In the final dataset, random lighting conditions were added to the blender scene. Also, to simulate the occlusions, a table was added to the scene to provide realistic occlusions. This achieved a better result (average error of 6.4 degrees), but it could still be improved by applying transfer learning. This increased the accuracy to an average error of 3.5 degrees, which is the same as the first model but on a much noisier dataset.

Results

The whole pipeline has been tested on multiple realistic indoor scene images. It achieved high accuracy in most cases. Sometimes, the rotation estimations are a bit off because of holes in the chair models or there not being enough light. There are a couple of improvements and additions that can still be made, like adding support for walls and other furniture types. This prototype shows that it can be very promising to apply these concepts in real applications.

13.2 Samenvatting

Introductie

De behoefte aan virtuele versies van scènes uit de echte wereld blijft toenemen. Het ontwerpen hiervan met 3D modelleertools vereist vaardigheid en is tijdrovend. Het zou voordeliger zijn om deze automatisch te genereren met machine learning modellen, omdat dit snel en nauwkeurig kan gebeuren en geen vaardigheid vereist. Er zijn veel toepassingen die deze virtuele scènes nodig hebben. Een voorbeeld is de behoefte aan virtuele scènes voor autonome robots om te leren rijden in een realistische omgeving. Een ander voorbeeld is te vinden in de architectuur. Wanneer mensen hun woonruimte opnieuw willen inrichten en meubels met een andere stijl willen proberen, weten ze vaak niet of die meubels wel bij hun huidige woonruimte passen. Misschien past het niet bij hun muurkleur of de stijl van een tafel. Het zou handig zijn als er een hulpmiddel beschikbaar was dat alleen een afbeelding van de scène als invoer zou nemen en een virtuele versie van die scène zou produceren. Dit zou de gebruiker in staat stellen wijzigingen aan te brengen aan die scène. Er bestaan al veel manieren om één of meer afbeeldingen om te zetten in 3D-scènes. Door te focussen op de herinrichting van een leefruimte wordt een gebruiksvriendelijk prototype ontwikkeld dat een beeld kan omzetten naar een virtuele scène. Door slechts één beeld als input te gebruiken, is het zo gebruiksvriendelijk mogelijk. De applicatie moet met name in staat zijn om de 3D-scène weer te geven en aanpassingen toe te staan, zoals het verwisselen van meubels voor andere meubelstukken in verschillende stijlen. Om deze doelen te bereiken is een pijplijn van machine learning modellen en andere algoritmen geconstrueerd, die hierna zal worden toegelicht.

Process pijplijn

Er wordt een pijplijn geconstrueerd die een RGB-beeld omzet in een virtuele 3D-scène. De pijplijn bestaat uit meerdere modellen en algoritmen die elk een deel van het probleem oplossen. Dit wordt gedaan door Mask R-CNN, een architectuur voor instance-segmentatie die de beste resultaten bereikt. Instantiesegmentatie is de segmentatie van een afbeelding, waarbij elk gesegmenteerd deel ook wordt geannoteerd met een klasse-etiket. De implementatie van Mask R-CNN die in dit onderzoek wordt gebruikt, heet Detectron2, de opvolger van Detectron. Het is voorgetraind op de COCO dataset, die al veel van de belangrijke meubelcategorieën bevat, zoals stoelen en tafels.

Zowel het masker als de box van een gedetecteerd object wordt gebruikt als input voor andere stappen. De tweede stap is de pose schatting. Deze pose bestaat uit een locatie en een oriëntatie. Het beschrijft hoe een object is gepositioneerd ten opzichte van de camera (van de smartphone). Voor dit prototype zijn zowel de locatie als de rotatie rond de opwaartse as van belang. De andere rotatie-assen zijn meestal niet van belang omdat meubels rechtop staan.
De schatting van de rotatie wordt gedaan door een Convolutioneel Neuraal Netwerk (CNN), dat het beelddeel binnen een bounding box ontvangt van Mask R-CNN en een rotatie rond de opwaartse as terug geeft. Meubelstukken zoals stoelen worden vaak bedekt door tafels of andere objecten. Daarom moet het CNN in staat zijn rotaties in te schatten bij veel variaties en onderdelen van meubels. Transfer learning wordt ook toegepast op de schatting van de rotatie. Door te starten met de VGG-16 netwerkarchitectuur en die aan te passen naar onze eigen output, traint het model veel sneller en bereikt het een hogere accuraatheid. De reden hiervoor is dat de gewichten van het voorgetrainde VGG-16 netwerk al informatie bevatten over de kenmerken van een stoel en een tafel. Deze hoeven alleen maar te worden omgezet in een nieuwe output. Het gebruikte model vereist een 244x244 input afbeelding en geeft 2 waarden: een sin en een cos. De reden waarom het niet slechts één enkele waarde (de hoek in graden) uitvoert, is dat een hoek van 355 graden zeer dicht bij een hoek van 5 graden ligt. Deze cyclische aard vereist uitvoerwaarden die cyclisch van aard zijn, zoals cos en sin. Samen vormen zij hoeken van 0 tot 360 graden.
De plaatsbepaling probeert elk object op de juiste plaats in de virtuele scène te plaatsen. Dit wordt gedaan door een enkele pixel te kiezen voor elk object. Vervolgens wordt die pixel geprojecteerd in 3D door gebruik te maken van de diepte van die pixel en de vergelijkingen uit fig. Om de diepte van

$$x' = (x - px) * depth/fx$$

$$y' = (y - py) * depth/fy$$

$$z' = depth$$

Figure (36) Vergelijkingen (a) - (c) om een pixel naar een 3D locatie te projecteren

elke pixel te bepalen, hebben we een manier nodig om een dieptekaart van de originele afbeelding te schatten. Hiervoor wordt de AdaBins architectuur voor diepteschatting gebruikt, waarmee state-of-the-art resultaten worden bereikt. Ook worden enkele intrinsieke cameraparameters gebruikt. Fx en fy zijn de focal lengths, terwijl px en py de principal points zijn. Deze kunnen uit de metadata worden gehaald en ook ruw worden geschat.
Zodra de 3D-locatie van een pixel die een object vertegenwoordigt is verkregen, moet het object op die locatie worden geplaatst. Nu moet worden besloten welk deel van het 3D-model op die plaats moet worden geplaatst. Dit moet het hoekpunt zijn dat overeenkomt met de geselecteerde pixel. De geselecteerde pixel is een pixel die zich nabij het centrum van het object bevindt. Om het hoekpunt van het model te vinden dat overeenkomt met deze pixel, wordt een ray gecast van de locatie van de camera (eerder berekend) naar het 3D model, dat wordt geroteerd zoals het verschijnt in de scène. De eindlocatie van de straal wordt evenveel verschoven als de geselecteerde pixel wordt verschoven van het centrum van de bounding box. Door deze speciale techniek te gebruiken, kan de juiste vertex worden gevonden en deze kan op de berekende plaats worden geplaatst.

Synthetische Dataset Generatie
Om een CNN te kunnen trainen dat de rotatie van een object in een afbeelding kan schatten, zijn trainingsgegevens nodig. Idealiter zijn dit echte gelabelde gegevens, maar aangezien die tijdens dit onderzoek niet beschikbaar waren, werden synthetische gegevens gegenereerd. Door beelden te genereren uit een 3D model dataset, werd een gelabelde afbeelding dataset geconstrueerd. Deze dataset moest zo realistisch mogelijk zijn. Voor het genereren van de dataset is gebruik gemaakt van Blender. Blender maakt via scripting met python mogelijk om wijzigingen in de scène aan te brengen. Door elk model te doorlopen, het te importeren in de scène en de camera eromheen te draaien, werden vele beelden gegenereerd en opgeslagen met de rotatie van de camera op dat punt. Om voldoende trainingsgegevens te verzamelen, werd een interval van 5 graden gekozen. Dat betekent dat de camera 5 graden draait en voor elke stoel 72 keer een beeld rendert.
Er waren meerdere versies van de dataset, elke versie verbeterde de vorige een beetje. De eerste dataset bevatte beelden van geroteerde stoelen zonder achtergrond of enige ruis. Dit leidde tot nauwkeurige voorspellingen met een gemiddelde fout van 3,5 graden. Dit getrainde model is echter niet

goed te generaliseren naar gegevens met ruis uit de echte wereld, en daarom werd een nieuwe dataset geconstrueerd. In deze dataset bevatte elk beeld een willekeurige realistische achtergrond om meer ruis aan het beeld toe te voegen. Ook werd elk beeld gesegmenteerd door Mask R-CNN om zoveel mogelijk achtergrond te verwijderen en tegelijkertijd een beeld te leveren als invoer voor het rotatieschattingsmodel. Tenslotte werden uit elk beeld meerdere delen geknipt om de occlusies na te bootsen waaraan stoelen lijden. Het model getraind op deze dataset behaalde geen hoge nauwkeurigheid (gemiddelde fout van 12.5 graden), omdat er veel onherkenbare stoelbeelden in de dataset zaten. Het knip-idee moest nog worden verbeterd. In de uiteindelijke dataset werden willekeurige lichtomstandigheden toegevoegd aan de blender scene. Ook werd, om de occlusies te simuleren, een tabel toegevoegd aan de scène om realistische occlusies te geven. Dit gaf een beter resultaat (gemiddelde fout van 6.4 graden), maar het kon nog verbeterd worden door transfer learning toe te passen. Dit verhoogde de nauwkeurigheid tot een gemiddelde fout van 3,5 graden, wat hetzelfde is als het eerste model, maar op een veel ruisrijkere dataset.

Resultaten
De hele pijplijn is getest op meerdere realistische beelden van scenes. Het bereikte een hoge nauwkeurigheid in de meeste gevallen. Soms wijken de schattingen van de rotatie een beetje af door gaten in de stoelmodellen of doordat er niet genoeg licht is. Er zijn nog een paar verbeteringen en aanvullingen mogelijk, zoals ondersteuning voor muren en andere meubeltypen. Dit prototype laat zien dat het veelbelovend kan zijn om deze concepten in echte toepassingen te gebruiken.