Diese Arbeit wurde vorgelegt am
Lehr- und Forschungsgebiet Theorie der hybriden Systeme

Serverbasierte Anwendung von Neuronalen Netzen auf Mobiletelefonen zur Lösung des Okklusions-Problems

Solving the occlusion problem on mobile phones via server-based application of neural networks

Bachelorarbeit
Informatik

August 2023

Vorgelegt von
Presented by

Vladimir Rzaev
Matrikelnummer: 355361
vladimir.rzaev@rwth-aachen.de

Erstprüfer
First examiner

Prof. Dr. rer. nat. Erika Ábrahám
Lehr- und Forschungsgebiet: Theorie der hybriden Systeme
RWTH Aachen University

Zweitprüfer
Second examiner

Prof. Dr. rer. nat. Thomas Noll
Lehr- und Forschungsgebiet: Software Modellierung und Verifikation
RWTH Aachen University

Betreuer
Supervisor

Dr. Henning Petzka
Lehr- und Forschungsgebiet: Theorie der hybriden Systeme
RWTH Aachen University

Contents

1 Introduction

1.1 Motivation

The motivation behind this work is to find models with good performance in monocular depth estimation so that the resulting depth maps can then be used for solving occlusion. These models need to be fast, as it is an application for mobile phones and users generally do not want to stand with their cameras pointing at the same place for too long, but on the other hand also need to provide accurate data to avoid mistakes or, even worse, having to re-do the procedure. As such, it is imperative to find a good balance between speed and performance. Even though there are several works done in regards to occlusion in AR on mobile devices, most of them require either to already have a built-in depth detection in the phone, which is only the case in the most recent smartphones, or they calculate the depth maps by themselves during the run, which is either very slow or has poor performances. We propose to offload the processing of camera images to a server that already has the model to be used loaded, which should save some time and guarantee good results, assuming a good model is used. This has the added benefit that the resulting app will be light-weight and thus a more approachable solution, as larger download sizes tend to dissuade users. We propose to use Monocular Depth Estimation due to it being the simplest approach in regards to usability as there is no calibration necessary or mandatory movement before getting the resulting depth map. This has the downside of being slower than other methods, as monocular depth estimation tends to be more resource-intensive. This has the additional downside that the camera has to remain stable for accurate results, as the processing time and the fact that the depth is only inferred for a single image causes the app to not be useful for dynamic scenes. This is, however, not a problem, as the scenes are static in our case. The end result will provide a convenient framework for developing an app that will help the residents of areas with proposed wind parks to visualise how it will look if and when the wind park is constructed, and as such provide better information in order to help them make an informed decision, be that a positive or negative one. In addition to this, any other AR projects where the camera will work in medium and long ranges might find this paper useful in finding a decision whether to use the more commonly found methods to calculate the depth maps ”by themselves” or to offload it to a server as we do here.

1.2 Related work

To the best of our knowledge, there are no scientific works done with the procedure of offloading the processing of depth maps to a server in AR. Estimating depth maps is however one of the most well-researched topics in AR, as occlusion is one of the most important indicators of quality for AR, as non-occluded virtual objects immediately break the immersion. As such, there is a wide variety of different techniques used. Some of the most commonly used ones are DepthLab [8], which uses Google's ARCore Depth API, Unity's own AR Foundation package, which also uses Google's ARCore Depth API (on Android devices) or Apple AR Kit (on iOS devices). [4] A summary on different techniques for occlusion in AR has been made by Macedo and Apolinário [19].

1.3 Contribution

We introduce a new approach in the topic of occlusion in AR, namely by offloading the depth map estimation to a server, which leads to the overall application being more lightweight and as such more approachable. We make use of pre-existing models for depth estimation, such as MiDaS (and the adaptation by the same authors DPT-Hybrid), as well as Monodepth2. We also propose a faster approach for estimating the depth maps, albeit vastly less accurate and with naive assumptions about the structure of the image where the depth is to be estimated.

1.4 Outline of this work

We will first describe the procedure of our lightweight approach for depth estimation, which has comparatively poor results but high speed. We explain the thought behind this approach and why it fails when the assumptions are not met. We then describe the first actual model, MiDaS, and how it functions. We show example depth maps resulting from applications of the model and also describe in which cases it fails. We then introduce an adaptation of MiDaS, DPT-Hybrid, which leads to improved results but longer processing times. We then introduce our last model, Monodepth2, explain how it works and show its applications and failures of the model. Following that, we describe our experimental approach and its results. Finally, we discuss the results and the possible use cases and give a short outlook on the future of this technology.

2 Models

2.1 Thresholding

A very basic (and now outdated) approach at estimating depth from an image is to use pixel intensity, operating under the assumption that objects that are closer to the camera will have higher intensity (and therefore be darker) than objects that are further away, which will have lower intensity (and therefore be brighter). One approach

for that is Otsu's thresholding [20], wherein we separate the foreground from the background according to a threshold intensity value and then estimate the depth only for foreground pixels. We can already see that this will cause issues with certain types of input images, such as images with bright or even white objects in the foreground or images with dark objects in the background, but nonetheless due to how simple and lightweight the method is, it is still worth considering. As Otsu's algorithm only works with grayscale images, we first convert our input image into grayscale.

Figure 1: Converting image to grayscale

We then apply Otsu's thresholding to the resulting grayscale image, which is an algorithm that automatically detects the best threshold to use in the image.[20] The algorithm starts with normalising the grayscale image

$$p_i = n_i/N \qquad\qquad p_i \geq 0 \sum_{i=1}^{L} p_i = 1$$

where L is the amount of gray levels present in the picture, n_i the amount of pixels at level i and $N = n_1 + n_2 + ... + N_L$ the total amount of pixels in the picture.[20] We now separate the pixels in the image into two classes C_0 and C_1, representing background and foreground objects respectively, by a threshold at level k.[20] This results in C_0 containing pixels with levels $[1, ..., k]$ and C_1 containing $[k + 1, ..., L]$.[20] This results in the following probabilities of class occurence and class mean levels:

$$Pr(C_0) = \sum_{i=1}^{k} p_i = w(k)$$
$$Pr(C_1) = \sum_{i=k+1}^{L} p_i = 1 - \omega(k)$$
$$\mu(k) = \sum_{i=1}^{k} ip_i$$
$$\mu(L) = \sum_{i=1}^{L} ip_i[20],$$

where $\omega(k)$ is the zeroth-order cumulative moment up to k-th level, $\mu(k)$ is the first-order cumulative moment up to k-th level and μ_T is the total mean level of the image.[20] With that, we can calculate the optimal threshold k^* by maximising the between-class variance $\sigma_B^2(k^*)$ (B signifying between-class):

$$\sigma_B^2(k^*) = \max_{1 \leq k < L} \sigma_B^2(k)$$
$$\sigma_B^2(k) = \frac{[\mu_T \omega(k) - \mu(k)]^2}{\omega(k)[1-\omega(k)]}$$

Once that threshold $k^*$is obtained, we use it in combination with cv2's THRESH_BINARY_INV, which sets pixels over the threshold to the value 0 and otherwise 255, signifying foreground.[20] This results in the following image for our previous input image:

Figure 2: Resulting image after applying Otsu's thresholding

As some parts of the bottom of the image were mistakenly labelled as background,

we dilate the image slightly. This has the additional benefit of later ensuring that all borders between foreground and background are properly detected. Dilation takes a kernel, in this case a matrix of odd size filled with 1s, and calculates the maximum value around every pixel in the image to get the new value for each pixel.[25] As we already thresholded the image in the previous step, we only have values of 0 and 1. We chose a $5x5$ kernel to make sure that any mistakes in the previous image are removed. This has the downside of enlarging the wind turbines. This is however not a large issue as wind turbines are already quite big and therefore the difference will not be very noticable.

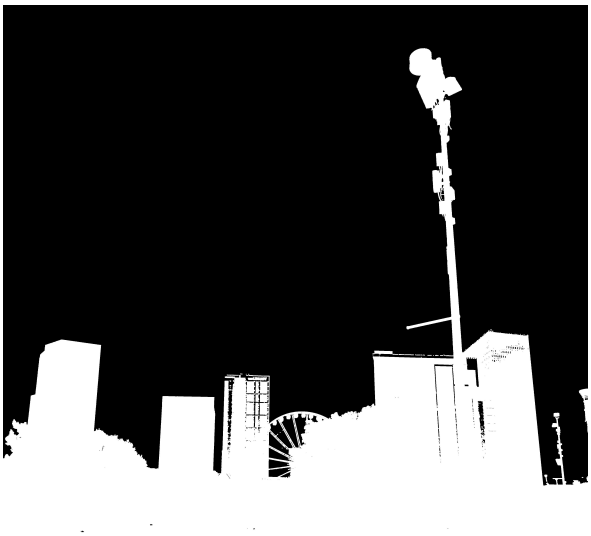

Figure 3: Resulting image after applying dilation with a $5x5$ kernel

We then generate a depth map with the same dimensions as the original image except filled with zeros. For each pixel in the grayscale image where the mask is white (255), we calculate the depth value based on the pixel's intensity. The intensity value of the pixel is first normalized to the range [0, 1] by dividing by 255, and then scaled to the desired depth range (e.g., [0, 255]). This has the clear downside of assuming that dark objects are closer to the camera, which is a clear failure case. For example: When a white car is close to the camera, this algorithm will deem it to be very far away due to having low colour intensity. However, as the algorithm is designed with a use case in nature in mind, where white is a rare colour (assuming it's not snowing) this is an acceptable failure case.

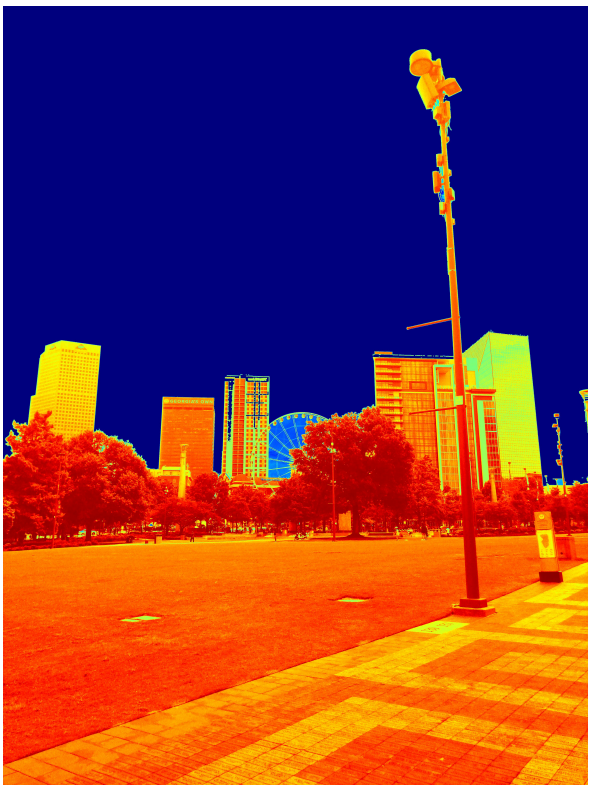

Figure 4: Resulting image after estimating depth values based on intensity, colours corresponding to their depth values on a scale from blue (very far away, depth 255), to red (very close, depth 0)

As this is a bit messy with some rather jagged lines, we apply cv2's two-dimensional Gaussian blur to the image.[25] A two-dimensional Gaussian blur works by initialising a kernel, which in this case is an odd-sized matrix, in our case 11x11, with values following a Gaussian distribution. The kernel is then placed on each pixel, where the weighted average is calculated by multiplying the kernel with the surrounding corresponding pixels element-wise.[11] This results in closer pixels having higher weight than pixels that are further away. After the multiplications are done, they are summed up to obtain a new value for the center pixel.[11] This process is then repeated for each pixel in the image.

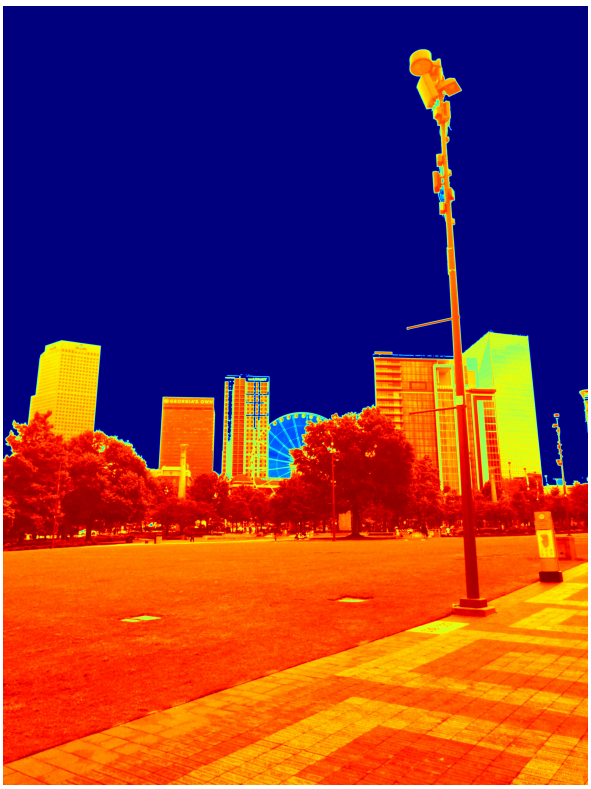

Figure 5: Resulting image after calculating depth values based on intensity

Clearly the foreground is not very accurate as it classifies almost everything as the same depth but it provides satisfying data for the background.

Figure 6: Failure case. From left to right: Input, Thresholded image, Output depth

Unfortunately however, this procedure does not always work, as when objects in the foreground have the same colour as the background, they might be detected as background and therefore receive a background depth. Another point of failure is non-monotonous colouring of foreground objects, as the wind turbine that is closest to the camera receives different depth values for the red and white parts of its rotor.

2.2 MiDaS

MiDaS is a training setup for a ResNet50 convolutional neural network, named after the 50 layers of its architecture.[15] ResNet50 is one of the most commonly used neural networks for monocular depth estimation. It addresses the issue of accuracy in neural networks improving for the first few layers of the network, which are essentially the building blocks of any neural network, and then rapidly degrading when adding more layers. The layers serve several functions, mostly related to processing and transforming input data, which in turn enables the network to learn and recognise patterns, which is their essential function. The innovation of ResNet50 consists of the introduction of residual blocks (hence the name ResNet = Residual Network), which each consist of several layers. These blocks forward the unprocessed input to later layers through the ”shortcut” or ”skip connections”, which allow the gradient to backpropagate much more directly, thus avoiding the ”vanishing gradient” problem, that being the gradient becoming smaller for each backpropagation, as they are the products of the results of the layers they travel through, which are very low in early layers.[15] The ResNet50 architecture is then trained on a multitude of different datasets with possibly incompatible annotations.[21] It is based on the concept of zero-shot cross-dataset transfer, which is a protocol that trains a model on certain datasets but tests it on completely new datasets that have not yet seen before during training.[21] This is especially useful to avoid running into the same biases over and over as every dataset has their own specifics, even though they are ”aiming to sample the visual world ’in the wild’”.[27] Torralba and Efros [27] came to the conclusion that if a model is trained only on one dataset, it has a drop in performance testing on different sets compared to testing on itself due to ”over-learning aspects of the visual data that relates to the dataset and not to the ultimate visual task”, as table 1 by Torralba and Efros [27] shows. They describe the problem for classification and detection of ”car” and ”person” for a model that has been trained on one model and tested on others. With some outliers, the drops in performance for differing sets can be quite large, such as a 78% drop when training on the Caltech101 [10] set for ”car” detection, which can be explained by poor generalisation in the datasets and therefore a poor representation of scenarios outside of testing.[27]
As such, it is beneficial to train on several datasets to cover a wider spectrum of biases and as such be more prepared for test cases outside of the datasets it was trained on.[21] With that in mind, MiDaS trains on 5 different datasets, namely DIML Indoor[16], MegaDepth[17], ReDWeb [32], WSVD [30], as well as a number of frames from movies.[21] As previously mentioned, every dataset has its own bias, thus selecting 5 different datasets expands the possible categories of images. MiDaS also uses previously unseen datasets for testing, namely DIW[6], ETH3D [23], Sintel [5], KITTI [12], NYUDv2 [24] and TUM-RGBD [26]. Table 1 by Ranftl et al. [21] visualises all the different characteristics of the used datasets for both training and testing. As all datasets used in training provide medium accuracy, it is to be expected that the model will not be able to detect details for which a high accuracy is needed. The sets used in training are all rather high diversity, a high amount of generalisation is to be expected,

especially considering that most datasets combine both indoor and outdoor images. DIML provides static indoor images [16], MegaDepth provides mostly static outdoor images with a highly accurate ground truth for the background, as it uses wide baseline multi-view stereo reconstruction for its ground truth[17], ReDWeb on the other hand is rather small, but provides "diverse and dynamic scenes with ground truth that was acquired with a relatively large stereo baseline"[21], WSVD is a collection of static videos from the internet and only consists of links to those videos [30], which makes it problematic to recreate results obtained from a model that has been trained on WSVD, as videos might get deleted or taken down, thus resulting in a different dataset to the one the model has been trained on. The last dataset MiDaS uses is a collection of 3D Movies, as they provide a wide range of high quality images (that being the single frames) "from human-centric imagery "story- and dialogue-driven Hollywood films to nature scenes with landscapes and animals in documentary features",[21] thus making it an excellent source of training material for any model with ambitions of generalisation. This however does not come without its own challenges. As films are made with the goal of providing a "visually pleasing viewing experience while avoiding discomfort for the viewer"[21], thus having variable disparity ranges, which leads to inaccuracies in the dataset due to these variations. Another issue is that none of the configurations of camera and rigs are known and also vary during a film. Lastly, films usually get edited in post-production, thus leading to even more alterations compared to the original camera footage. Nonetheless, these issues can get fixed. MiDaS selected movies according to the following criteria: 1: Using only physical stereo camera 2: balancing realism and diversity (get a diverse but realistic set), 3: only blu-ray available films to get high quality images. For pre-processing, MiDaS starts by center-cropping all frames to 1880x800, splitting the film into "chapters", dropping first and last due to credits.[21] With the help of FFmpeg [1], individual clips are then extracted and sampled. Even though stereo matching is designed to work on positive disparity ranges, as opposed to the both positive and negative extracted here, that issue gets rectified by having image pairs.[21] As the images are paired, one can use optical flow algorithms to check for disparities in the pairs and thus reject pairs if their disparities are too large. MiDaS then filters out any sky pixels and adds the remaining frames to the dataset.[21]

With all these different datasets, the biggest challenge is to get consistency across the sets. As previously shown in Table 1, the used datasets provide ground truth in different forms, be that absolute depth, depth to an unknown scale or disparity maps.[21] Thus we require "an output space that is compatible with all ground-truth representations and is numerically well-behaved."[21] In addition, to measure how well the model is doing, we need a loss function that can handle the different datasets.[21] The idea here is to create an overall loss function L_l comprising of scaled similarity loss between the predicted and target images, as well as a regularisation loss, which aims to improve the network's generalisation.

$$L_l = \frac{1}{N_l} \sum_{n=1}^{N_l} L_{ssi}(\hat{d}^n, (\hat{d}^*)^n) + \alpha L_{reg}(\hat{d}^n, (\hat{d}^*)^n)$$

$$L_{ssi}(\hat{d}, \hat{d}^*) = \frac{1}{2M} \sum_{i=1}^{M} \rho(\hat{d}_i - \hat{d}_i^*)$$
$$L_{reg} = (\hat{d}, \hat{d}^*) = \frac{1}{M} \sum_{k=1}^{K} \sum_{i=1}^{M} (|\nabla_x R_i^k| + |\nabla_y R_i^k|),$$

where N_l is the training set size, ∇_x, ∇_y is the Del operator (e.g. the partial derivatives with respect to the x and y dimensions), α is set to 0.5, $\hat{d}, \hat{d}^*$ are the aligned prediction and ground truth, L_{ssi} is the scale- and shift-invariant loss, M the number of pixels with valid ground truth, K is the amount of scale levels, which is set to 4, $R_i = \hat{d}_i - \hat{d}_i^*$ and R_k the difference of disparity levels at scale k. L_l is the final loss function.

With that, MiDaS uses the ResNet-based architecture by He et al. [15]. The encoder gets initialised with pre-trained weights, other layers are random with a 10^{-4} learning rate for random layers and 10^{-5} for pre-trained (to avoid over-learning).[21] Decay rates are set to 0.9 and 0.999. In addition to that, some images are flipped or randomly cropped to augment the data. As the ResNet50 model is pre-trained on the ImageNet dataset, it is expected to perform better on that dataset and images that are similar to it. The model is then trained for 60 epochs on 72000 images. Ground-truth disparity is shifted and scaled to [0,1] for all sets.[21]

For testing, the model uses sets that have not been seen before, though with some minor modifications. For DIW MiDaS created a validation set, which is a set that is set aside during training in order to assess the model's performance by using some score to compare the prediction to the ground truth. The DIW validation set consists of 10000 images. The official test set [6] of 74441 images is used for testing.[21] For NYU the official test set [24], containing 654 images, is used.[21] For KITTI the official validation set [12], containing 3,712 images and the Eigen test set [9], containing 697 images are used.[21] For ETH3D and Sintel, the set had to be shrunk down a bit, as not all images had ground truth available and are thus not suitable for calculating loss. For TUM only the images in the dynamic subset are used that contain humans in indoor environments, in order to get a wider spectrum of images.[21]

As the methods of providing ground-truth vary between the datasets, different error rates need to be used as well. For DIW the Weighted Human Disagreement Rate is used. For datasets with relative depth as ground-truth, root mean squared error is used. For datasets providing accurate absolute depth, the mean absolute value of the relative error is used.[21] For KITTI, NYU and TUM, the percentage of pixels with $\delta = max(\frac{z_i^*}{z_i}, \frac{z_i}{z_i^*}) > 1.25$ is used.[21] This naturally comes with more problems, as each dataset has differently sized/scaled images. Therefore images need to be rescaled to be uniform. In order to achieve this, the images are rescaled so that the larger axis equals 384 pixels, with the shorter one being a multiple of 32, while maintaining aspect ratio.[21] As KITTI has a very high aspect ratio, this would lead to very small input images.[12] Thus, only for KITTI, the shorter axis is set to 384 pixels, with the rest of the scaling working the same way.[21] As tables 3 and 4 by Ranftl et al. [21] show, the performance of a model when fine-tuning on another set and testing on yet another set massively declines. This is due to poor generalisation in the datasets and the model thus getting "confused" when fine-tuning on images that are so different from the ones

used in training.
To fix this issue, Ranftl et al. [21] introduce dataset mixes for training, consisting of various combinations of ReDWeb [32], DIML Indoor [16], Movies, Megadepth [17] and WSVD [30] As tables 6 and 7 by Ranftl et al. [21] show, the performance greatly improves when using a mix of datasets as opposed to just one, due to much higher generalisation. The improvements when using the full mix range from 7% on Sintel [5] to 38.5% on NYU [24]. This shows that having a wider range of training images is extremely beneficial for overall performance, which can also be seen by the comparatively good performance of models trained only on RW. Even though it is a very small set, it covers a wide variety of image types and is therefore somewhat of a mix as well, even if a small one. Even though the results are good, there are naturally some failures and biases present in the model, such as lower parts of an image being perceived as closer, which can be fixed by adding rotated images to the datasets. It is however likely not needed, as humans tend to take pictures with the ground being closer than objects in higher parts of the image.[21] It is also of no concern for our use case, as any images used in our application will naturally have the ground as the closest part of the image due to being mostly used in fields. Another failure are reflecting objects, as the model does not perceive them as such and thus attempts to calculate depth for the objects being reflected.[21] This is once again not an issue for our case, as reflecting objects will not be present in areas for proposed wind parks, as they are mostly empty. Another issue is that thin objects and objects that are very far away are usually not recognised. This could however be caused by low resolution images and depth maps generally not providing good data on background objects.[21] In addition to the regular MiDaS model, Ranftl et al. [22] later added additional transformers, which usually consist of an encoder-decoder architecture, where the encoder creates lower-dimensional representations of an image and the decoder generates a depth map based on the learned features. In order to do that, transformers separate the image into patches, which are square or rectangular areas of an image with something in common, usually pixel values.[28] These patches are then either used as input for convolutional layers or, in this case, used by the encoder to extract features from them. It is important to mention that the patches can, but do not have to, overlap in an image, leading to easier extraction of certain features.[22] Transformers are extremely useful for dense prediction tasks, as the self-attention mechanism, which is at the core of transformers, captures the relationship between pixels and their context in the image. The mechanism works by calculating a so-called "attention score" for each patch in the image and thus determining its importance in regards to the image. Without going into detail, these scores are effectively descriptions of how relevant other pixels are for a given pixel. As such, Ranftl et al. do not use convolutional neural networks.[22] These additional transformers effectively transform an image into a bag of words, as every image patch takes the role of a word. These patches are then flattened into vectors and individually embedded.[22] As such, the transformer has a global range due to every word can see each other word, unlike in convolutional networks, where the range is gradually increased.[22] Ranftl et al. introduce 3 different variants of this procedure, which results in 3 different models. For the Base model the previously

described patch-based embedding and 12 transformer layers are used, the large model uses the same embedding but 24 transformer layers and a wider feature size. The hybrid model, which we use in our application, uses ResNet50 for embeddings instead, which is then followed by 12 transformation layers.[22] The Dataset mix from before gets extended by 5 additional datasets, resulting in a training set containing 1.4 million images. Table 1 by Ranftl et al. [22] shows that the DPT-Hybrid model produces results roughly in the middle between the DPT-Large model and the regular MiDaS model, as well as vastly outperforming models by other autors.

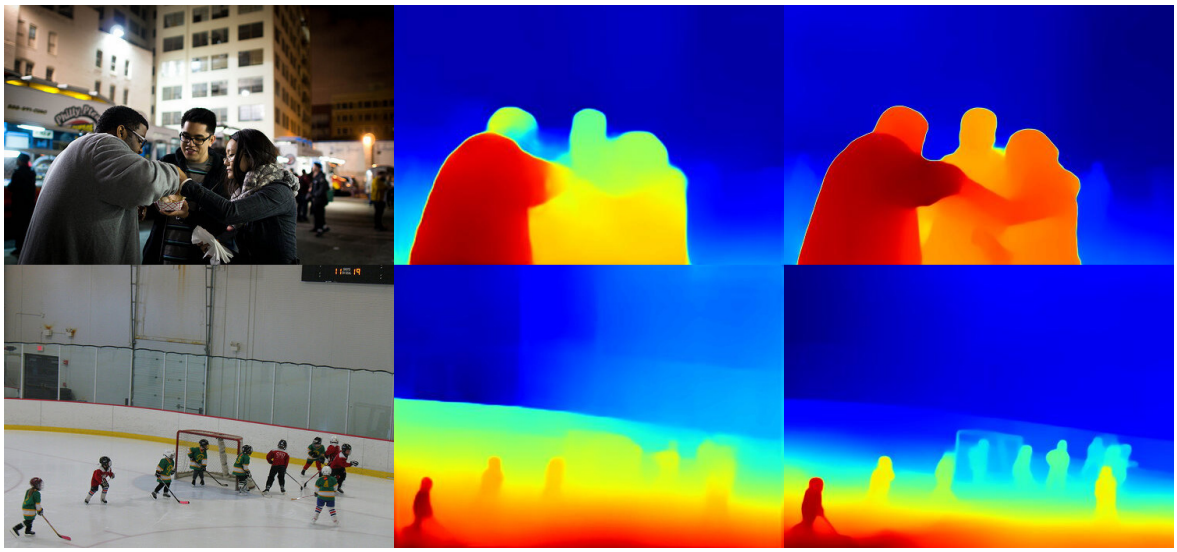

Figure 7: Comparison of resulting depth images with DPT-Hybrid and the regular MiDaS model on randomly chosen images from the DIW [6] dataset.

We can see that DPT-Hybrid provides better details on the foreground as well as more accurate values but the regular MidaS model has an advantage in the background.

2.3 MonoDepth2

Another model that is well suited for the task is MonoDepth2, developed by Godard et al. [14], which is a model using self-supervised training on stereo pairs and/or monocular video. Video however comes with the added challenge that in addition to estimating the depth of the single frames of the video, it also needs to estimate the motion that happened between two images that form a pair, as otherwise there will be a lot of flickering in the resulting depth video.[14] On the other hand, using only stereo data for training causes the camera-position estimation to be "a one-time offline calibration" which can cause issues regarding occlusion [13] As such, we will focus on the monocular video for this model and the combination of monocular and stereo. Godard et al. [14] propose three innovations that lead to greatly improve depth estimation when training on stereo pairs or video. These innovations are:

- Appearance matching loss in order to combat the issue of occluded pixels
- Auto-masking to ignore pixels without relative camera motion
- Reducing depth artifacts by sampling the images at input resolution [14]

As MonoDepth2 is a self-supervised depth estimation model, it operates without ground truth and instead uses image reconstruction error, where the model receives a set of stereo pairs or sequences as input and then tries to reconstruct the image by hallucinating a depth map for an image and then projecting that on the following images,[14] effectively trying to reconstruct an image from the POV of another image. As this can lead to a high amount of wrong depths for certain pixels, depending on the relative position of the views. Godard et al. [14] formulate this problem as "minimization of a photometric reprojection error at training time."

$$L_p = \sum_{t'} pe(I_t, I_{t'\to t})$$
$$I_{t'\to t} = I_{t'}\langle proj(D_t, T_{t\to t'}, K)\rangle$$
$$pe(I_a, I_b) = \frac{\alpha}{2}(1 - SSIM(I_a, I_b)) + (1-\alpha)||I_a - I_b||_1 [14]$$

Where L_p is the photometric projection error, I_t is the input image, $I_{t'}$ is the source view, pe is a photometric reconstruction error, $\langle\rangle$ are the sampling operators, D_t is a depth map, $T_{t\to t'}$ is the relative pose for for each source view with regard to the image's pose, $proj()$ are the 2D coordinates for projected depths in the source view and K pre-computed intrinsics of all the views, which we assume to be identical.[14] SSIM is the structural similarity as described by Wang et al. [31]. For α in pe Godard et al. [14] chose a value of 0.85. For sets with stereo pairs, the source view $I_{t'}$ is the second image in the pair, as the relative pose is known.[14] As relative poses are unknown in the case of monocular image sequences, Monodepth2 uses the 2 frames that are temporally adjacent to the input frame as source frames, so $I_{t'} \in \{I_{t-1}, I_{t+1}\}$[14] For the combined training on both stereo and monocular sequences, MonoDepth2 uses both the temporally adjacent frames as well as the opposite stereo pair.[14]

As unsupervised models generally produce lower quality depth maps than fully supervised ones, Godard et al. propose a number of changes in regards to conventional unsupervised training. For one, self-supervised models usually average out the reprojection error when working with multiple source images, which leads to issues if a pixel is visible in the target image but not in the source images. To fix this issue, Godard et al. propose to take the minimum error instead of the average. As such, the formula for per-pixel photometric loss changes to:

$$L_p = \min_{t'} pe(I_t, I_{t' \to t})[14]$$

Another common issue is that self-supervised monocular training usually assumes that there is a static scene and a moving camera. Due to our use case involving a moving camera, even if slightly, this might cause inaccurate data. If one of the aforementioned assumptions turns out to be false, performance significantly declines, such as assigning infinite depth to objects that are usually in motion during training.[18] To rectify this issue, Godard et al. propose a mask that filters out any pixels that do not change appearance between two frames. This could be due to objects moving at the same speed as the camera, the camera being static or just low texture regions.[14] As such, the mask μ only includes the loss of pixels that have a lower reprojection error of the warped image than the original image:

$$\mu = [\min_{t'} pe(I_t, I_{t' \to t}) < \min_{t'} pe(I_t, I_{t'})]$$

[14] where [ ] is the Iverson bracket, which evaluates to 1 if the condition inside the bracket is met and 0 if not. This mask would then filter out any sequences with static cameras and also prevent objects that move at the same speed as the camera, thus remaining in more or less the same spot in the image, to "contaminate" the loss.[14] This leads to the final loss function:

$$L = \mu L_p + \lambda L_s$$
$$L_s = |\delta_x d_t^*| e^{-|\delta x I_t|} + \delta_y d_t^* | e^{-|\delta y I_t|}$$

where $d_t^* = d_t / \overline{d_t}$ is the "mean-normalized inverse depth" from [29].[14] λ is selected as 0.001 [14]. Godard et al. use a ResNet18 encoder instead of the commonly used ResNet50 as it is a lot faster due to having fewer layers and parameters.[14] An encoder has the task compressing high-dimensional data, such as images in this case, to a lower dimension, such as a numerical representation of the image. They then extract certain features from that representation, such as edges or textures and create embeddings, which are low-dimensional representations of the image by mapping each pixel's features to a lower dimension, once again as a numerical representation. These embeddings can then be optimised through applying a loss function and attempting to minimise the loss through training. The architecture is pre-trained on ImageNet [7], just like MiDaS, as this improves performance.[14] The model is then trained for 20 epochs on the Eigen split[9] of the KITTI dataset [12] with a batch size of 12, a uniform resolution of 640x192 and a learning rate of 10^{-4} for the first 15 epochs and 10^{-5} for

the remaining epochs.[14] The training takes 12 and 15 hours for the monocular and monocular plus stereo models respectively.[14] Table 1 by Godard et al. [14] shows that Monodepth2 outperforms most other common models for both monocular training and the combination of monocular and stereo training, even without pre-training. As expected, the results improve greatly when combining monocular and stereo training. To visualise the results, we use the same random images from DIW used for visualising the MiDaS results before.

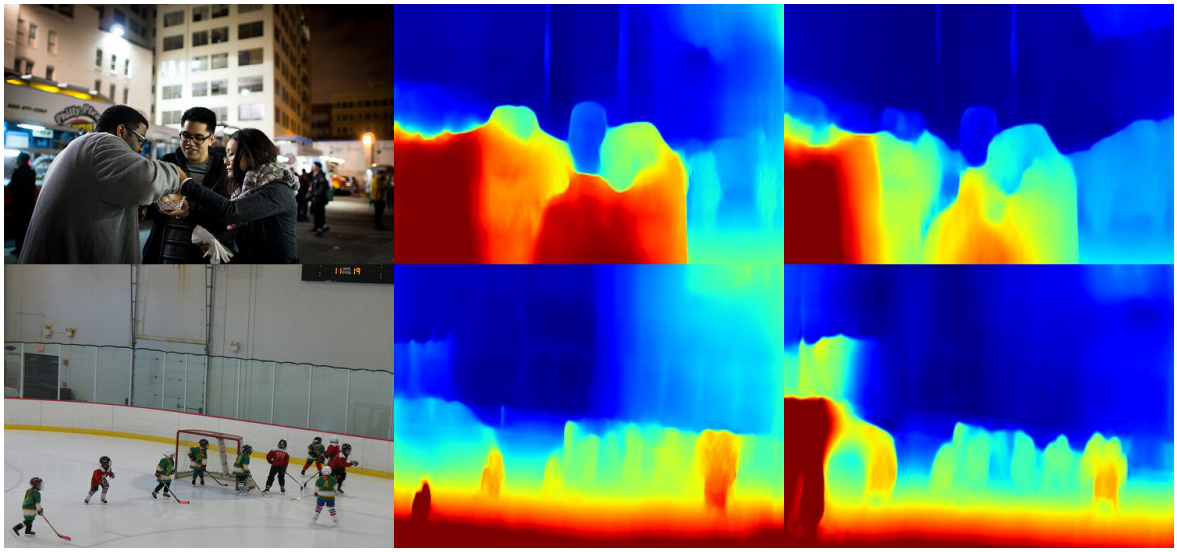

Figure 8: [14]From left to right: Input image, Monodepth2 trained only monocularly, Monodepth2 trained both monocularly and stereo.

Results for close range clearly leave a lot to be desired, but for medium and high range, they are fairly accurate.

3 Experiment

For our experiment, we set up a local Django server that can receive images as input and then passes them to a model through a subprocess that processes them and returns the corresponding depth maps. For this experiment, we choose our thresholding method described earlier, MiDaS[21], DPT-Hybrid[22] and Monodepth2[14] trained on both monocular and stereo. We will compare their performance as well as the amount of time it takes to send the image, process it and receive an answer from the server. It is important to stress that we're using a local server and as such the latency is lower than it would be on regular servers. In all cases we use a 3840x5120 pixel image with a total size of 4.69MB, shot with a Huawei Mate 20 lite phone camera (20 MP). For the models requiring a GPU, we use a single Nvidia GeForce RTX 2060 Super. As our metric to compare the models/methods we use AbsRel, which is short for Absolute Relative Error. The way the error is calculated is by taking the absolute error (ground truth - prediction) and normalising it by the ground truth again (making it relative).

For AbsRel a low value is better, as the error is smaller. We choose the ETH3D [23] dataset for testing, as it has not been seen by any model before except in testing for MiDaS and also contains mostly outdoor static images. [21] We additionally introduce a rating system for evaluating their performance, which is the multiplication of the AbsRel score with the time it takes to receive an answer from the server, with a lower score being better.

	Thresholding	MiDaS	DPT-Hybrid	Monodepth2
Time to result (without server)	0.35s	2.73s	3.07s	4.02s
Time to result (with server)	3.04s	5.51s	5.77s	6.48s
AbsRel on ETH3D	0.849	0.164	0.098	0.113
Score	2.58	0.90	0.57	0.73

Table 1: Time needed to load a picture, send it to the server and receive a depth map, as well as AbsRel on average for the ETH3D dataset and the resulting score

The AbsRel values coincide with our observations from the previous sections, with DPT-Hybrid providing the best results but being slower than MiDaS. Monodepth2 is the slowest and its results are better than MiDaS but slightly worse than DPT-Hybrid (according to AbsRel). When looking at our score for the methods, we once again see that DPT-Hybrid has the best performance, followed by Monodepth2 and MiDaS. Our thresholding method, even though it is a lot faster than the others, has an immensely worse score than them due to its AbsRel error rate being a lot higher.

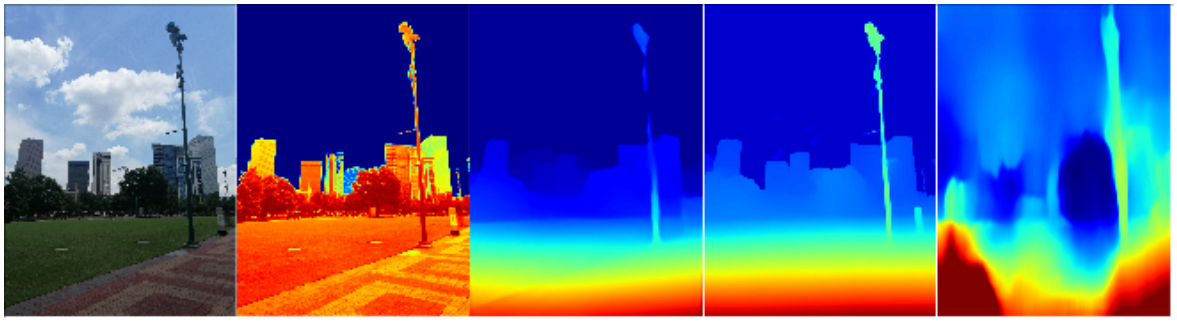

Figure 9: [14]From left to right: Input image used for the previous table, depth map estimation with the thresholding method, depth map estimation with the MiDaS model, depth map estimation with the DPT-Hybrid model, depth map estimation with the Monodepth2 model trained on monocular and stereo. All depth maps are visualised with the jet colour scheme, using a colour range from red (very close) to blue (very far)

```
def process_image(image_data):
    # Pass the image data to the processing program using subprocess

    command = ['python', 'PATH TO PROCESSING SCRIPT']
```

```
    process = subprocess.Popen(command, stdin=subprocess.PIPE, stdout
   =subprocess.PIPE)
    stdout, stderr = process.communicate(input=image_data)

    # Get the directory path of the current file
    current_dir = os.path.dirname(os.path.abspath(__file__))


    #Read the depth maps from the CSV files
    depth_map_path = glob.glob(PATH TO DEPTH MAP)
    depth_map_values = []

    with open(depth_map_path, 'r') as f:
            depth_map_reader = csv.reader(f)
            depth_map_values.extend([row for row in depth_map_reader
   ])


    return depth_map_values
```

Listing 1: The core of our processing, depending on the method we choose for our testing, that code gets executed with an input image. The resulting depth map is then read and returned.

```
while (true)
{
    // Wait for the polling interval
    yield return new WaitForSeconds(pollInterval);

    // Capture a screenshot
    yield return new WaitForEndOfFrame();
    Texture2D texture = ScreenCapture.CaptureScreenshotAsTexture();

    // Convert the texture to bytes
    byte[] imageBytes = texture.EncodeToPNG();
    Destroy(texture);

    // Create a UnityWebRequest to upload the image
    WWWForm form = new WWWForm();
    form.AddBinaryData("image", imageBytes, imageName, "image/png");

    UnityWebRequest request = UnityWebRequest.Post(serverURL, form);
    yield return request.SendWebRequest();
    // Get the response text
    string responseText = request.downloadHandler.text;
    Debug.Log("Server Response: " + responseText);}}
```

Listing 2: Image uploading process in Unity. The camera takes a screenshot and uploads that image to the server. The script then waits for the server's response.

We use Unity to simulate a camera with a virtual environment[3] for testing purposes. We attach a script to the camera that constantly sends images of what it sees to

the server for processing. The resulting depth images are then returned to the Unity environment by the server. Unity offers the option of applying a depth shader, where any objects that are to be inserted at a given depth are compared to the depth values in that given area. Creating a depth shader is however a very complex and unnecessary procedure as several pre-made ones already exist. The way a depth shader works is that for every pixel in the area, if the depth at which the object is to be inserted is higher than the depth of the pixel in the real world, or the virtual environment in our simulation, it is occluded in order to enhance realism. DepthLab[8] can be used to achieve this (only on Android 8.1 or higher), which uses Google's ARCore for depth estimation but the framework can be adapted to receive depth maps from another source. Google unfortunately does not disclose how exactly they estimate depths in ARCore and as such it is not possible to determine an exact error rate for a dataset. We show a comparison between the depth maps from DepthLab (using ARCore) and our models.

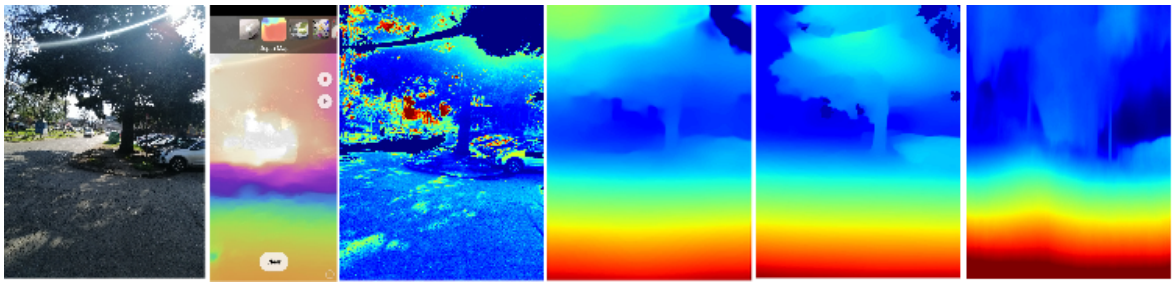

Figure 10: Comparison of depth maps when facing into the sun. From left to right: Input image, depth map produced by DepthLab [8], our thresholding method, MiDaS, DPT-Hybrid and Monodepth2.

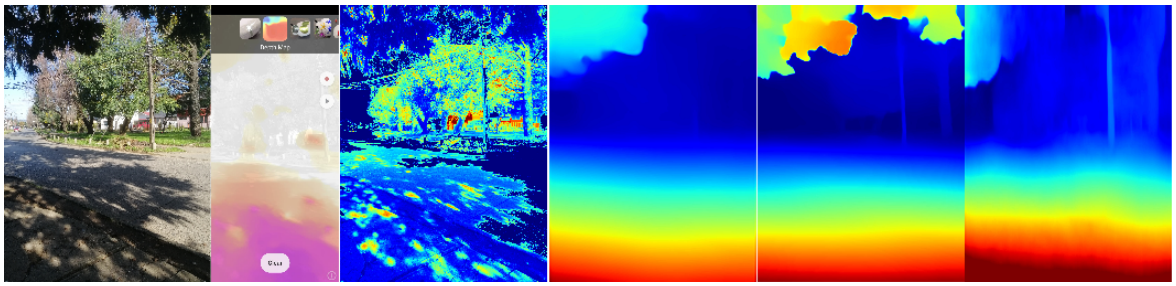

Figure 11: Comparison of depth maps when facing away from the sun. The same order as in the previous figure.

We can see that DepthLab (and by extension ARCore) gives good depth values for close range but breaks down for higher ranges. This is due to the ARCore version used by DepthLab being limited to a distance of 8.191m. ARCore has however had an update increasing its effective range to 65.535m.[2] As shown and described in the previous table, we can see that DPT-Hybrid provides the best results, closely followed

by Monodepth2 and MiDaS. Clearly visible failures by the models are that Monodepth2 doesn't provide good details, as it is quite blurred, as well as DPT-Hybrid, and MiDaS in particular, get "tricked" by the light when facing into the sun.
DepthLab results in the following occlusion effects:

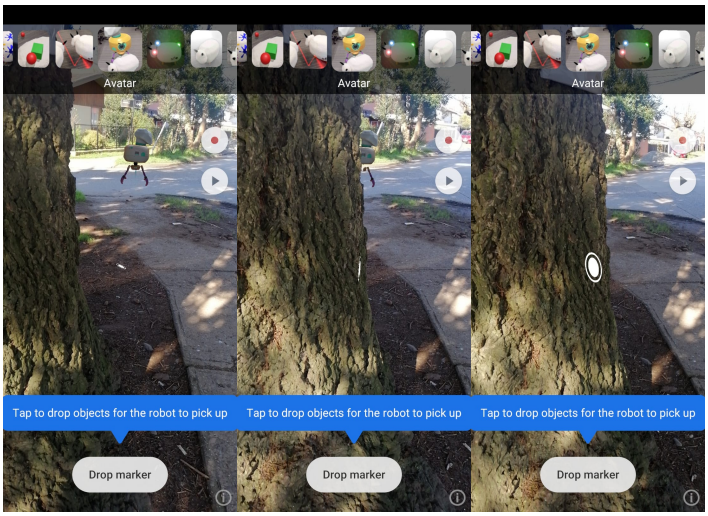

Figure 12: Occlusion effects in DepthLab in close range

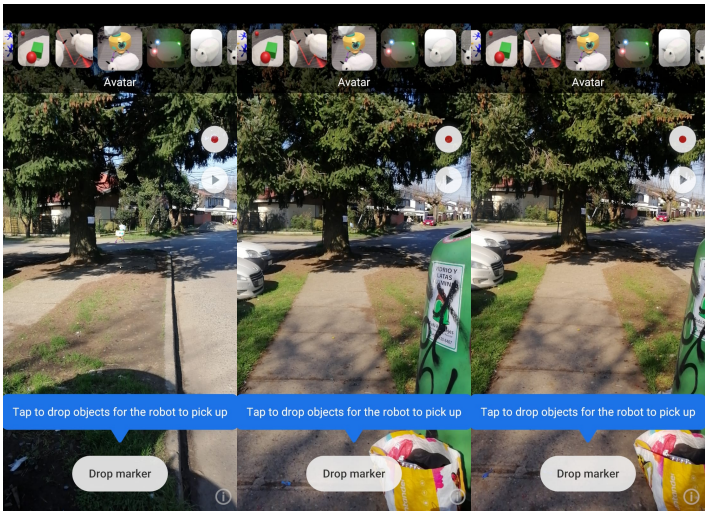

Figure 13: Occlusion effects in DepthLab in medium range

As previously described, we can see that the occlusion effects are satisfying for close range, but degrade for higher ranges.

4 Conclusion

In conclusion, we evaluated several approaches for server-based depth map estimation, each providing their own advantages and drawbacks. Even though our introduced

thresholding method is very simple and fast, it is near useless for actually estimating a depth map and should therefore be discarded. In contrast, the other models provide highly satisfying depth maps but need a long time to receive a response due to the server communication, which will only be longer when the server is not implemented locally anymore. As such, the user will have to remain stationary for a while and hold the phone still, due to the depth map not corresponding with the actual position of the user anymore if they move, which is a suboptimal solution. However, as DepthLab requires the user to have a fairly new phone, server-based occlusion is a good solution for older phones, as they would not be able to process a model for depth estimation anyway. Another benefit of server-based application of the depth map estimation is the small size of the app, as all processing is handled on the server and all that remains for the app to do is to send a request to the server, receive an answer and implement it into its occlusion shader. The users will therefore have to weigh their options whether to have a large app that does the processing quickly due to not having to communicate with a server, albeit slightly worse, or to have a small app that offloads the processing to an external server but have to wait to get a more accurate response while having to remain stationary during that time.

5 Future work

Due to our work being focused on providing a framework for estimating depth maps for occlusion fixing through a server, we leave out the actual implementation of the occlusion shader in Unity. As the field of depth estimation as well as occlusion is constantly evolving, the models for depth estimation will undoubtedly improve and as such this work can serve as a base for testing improved models. The technology for depth estimation built into phones will also definitely improve and as such it will definitely be worth to reevaluate at a later time whether it is worth it to offload the depth estimation to a server or to leave it all up to the app.

References

[1] URL https://ffmpeg.org/.

[2] URL https://developers.google.com/ar/develop/depth/changes?hl=en.

[3] URL https://assetstore.unity.com/packages/3d/environments/landscapes/low-poly-simple-nature-pack-162153.

[4] URL https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@5.0/manual/index.html.

[5] Daniel J. Butler, Jonas Wulff, Garrett B. Stanley, and Michael J. Black. A naturalistic open source movie for optical flow evaluation. In Andrew Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid, editors, *Computer Vision – ECCV 2012*, pages 611–625, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-33783-3.

[6] Weifeng Chen, Zhao Fu, Dawei Yang, and Jia Deng. Single-image depth perception in the wild. 04 2016.

[7] 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. doi: 10.1109/CVPR.2009.5206848.

[8] Ruofei Du, Eric Turner, Maksym Dzitsiuk, Luca Prasso, Ivo Duarte, Jason Dourgarian, Joao Afonso, Jose Pascoal, Josh Gladstone, Nuno Cruces, Shahram Izadi, Adarsh Kowdle, Konstantine Tsotsos, and David Kim. DepthLab: Real-time 3D Interaction with Depth Maps for Mobile Augmented Reality. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, UIST '20, pages 829–843. ACM, 2020. doi: 10.1145/3379337.3415881.

[9] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. 11 2014.

[10] Li Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *2004 Conference on Computer Vision and Pattern Recognition Workshop*, pages 178–178, 2004. doi: 10.1109/CVPR.2004.383.

[11] Estevão S. Gedraite and Murielle Hadad. Investigation on the effect of a gaussian blur in image filtering and segmentation. In *Proceedings ELMAR-2011*, pages 393–396, 2011.

[12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer*

Vision and Pattern Recognition, pages 3354–3361, 2012. doi: 10.1109/CVPR.2012.6248074.

[13] Clement Godard, Oisin Aodha, and Gabriel Brostow. Unsupervised monocular depth estimation with left-right consistency. 07 2017. doi: 10.1109/CVPR.2017.699.

[14] Clément Godard, Oisin Aodha, Michael Firman, and Gabriel Brostow. Digging into self-supervised monocular depth estimation. 11 2019. doi: 10.1109/ICCV.2019.00393.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

[16] Youngjung Kim, Hyungjoo Jung, Dongbo Min, and Kwanghoon Sohn. Deep monocular depth estimation via integration of global and local predictions. *IEEE Transactions on Image Processing*, 27(8):4131–4144, 2018. doi: 10.1109/TIP.2018.2836318.

[17] Zhengqi Li and Noah Snavely. Megadepth: Learning single-view depth prediction from internet photos. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2041–2050, 2018. doi: 10.1109/CVPR.2018.00218.

[18] Chenxu Luo, Zhenheng Yang, Peng Wang, Yang Wang, Wei Xu, Ram Nevatia, and Alan Yuille. Every pixel counts ++: Joint learning of geometry and motion with 3d holistic understanding. *IEEE Trans. Pattern Anal. Mach. Intell.*, 42(10):2624–2641, oct 2020. ISSN 0162-8828. doi: 10.1109/TPAMI.2019.2930258. URL `https://doi.org/10.1109/TPAMI.2019.2930258`.

[19] Márcio C. F. Macedo and Antônio L. Apolinário. Occlusion handling in augmented reality: Past, present and future. *IEEE Transactions on Visualization and Computer Graphics*, 29(2):1590–1609, 2023. doi: 10.1109/TVCG.2021.3117866.

[20] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979. doi: 10.1109/TSMC.1979.4310076.

[21] R. Ranftl, K. Lasinger, D. Hafner, K. Schindler, and V. Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE Transactions on Pattern Analysis amp; Machine Intelligence*, 44(03):1623–1637, mar 2022. ISSN 1939-3539. doi: 10.1109/TPAMI.2020.3019967.

[22] René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. Vision transformers for dense prediction. *ICCV*, 2021.

[23] Thomas Schöps, Johannes L. Schönberger, Silvano Galliani, Torsten Sattler, Konrad Schindler, Marc Pollefeys, and Andreas Geiger. A multi-view stereo benchmark with high-resolution images and multi-camera videos. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2538–2547, 2017. doi: 10.1109/CVPR.2017.272.

[24] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In Andrew Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid, editors, *Computer Vision – ECCV 2012*, pages 746–760, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-33715-4.

[25] Himanshu Singh. *Advanced Image Processing Using OpenCV*, pages 63–88. Apress, Berkeley, CA, 2019. ISBN 978-1-4842-4149-3. doi: 10.1007/978-1-4842-4149-3_4. URL `https://doi.org/10.1007/978-1-4842-4149-3_4`.

[26] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 573–580, 2012. doi: 10.1109/IROS.2012.6385773.

[27] Antonio Torralba and Alexei A. Efros. Unbiased look at dataset bias. In *CVPR 2011*, pages 1521–1528, 2011. doi: 10.1109/CVPR.2011.5995347.

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL `http://arxiv.org/abs/1706.03762`.

[29] Chaoyang Wang, José Miguel Buenaposada, Rui Zhu, and Simon Lucey. Learning depth from monocular videos using direct methods. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2022–2030, 2018. doi: 10.1109/CVPR.2018.00216.

[30] Chaoyang Wang, Simon Lucey, Federico Perazzi, and Oliver Wang. Web stereo video supervision for depth prediction from dynamic scenes. 04 2019.

[31] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004. doi: 10.1109/TIP.2003.819861.

[32] Ke Xian, Chunhua Shen, Zhiguo Cao, Hao Lu, Yang Xiao, Ruibo Li, and Zhenbo Luo. Monocular relative depth perception with web stereo data supervision. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 311–320, 2018. doi: 10.1109/CVPR.2018.00040.