MONODEPTH2 PAPER REVIEW

Le Tran Nguyen Khoa
2152674
khoa.lesteve@hcmut.edu.vn

Nguyen Dang Hoai Nam
2152181
nam.nguyencshcmut@hcmut.edu.vn

Dinh Viet Thanh
2152966
thanh.dinhalex19cs@hcmut.edu.vn

November 30, 2023

1 Introduction

Monocular depth estimation is a process of assigning a depth map to each pixel of an image corresponding to its position in 3D space using a single image. Applications on depth estimation including Robotics - where machine's navigation in 3D space through a 2D camera is crucial, Virtual Reality - where virtual objects are constructed to reflect the relative distance between itself and the device's camera, and Endoscopy in medicine, where using a pair of cameras (traditional stereo) to estimate depth is difficult due to narrow space.

In this review, we discuss the innovations of a self-supervised monocular depth estimation model in the paper "Digging Into Self-Supervised Monocular Depth Estimation" (Monodepth2 [1]) as well as the network architecture for this method. We also provide the experiments on our re-implemented model on smaller subset of datasets from KITTI [2] and NYUv2 [3].

2 Background

In this section, we demonstrate our research on the topic of depth estimation, as well as reviewing recent literature of monocular depth estimation using self-supervised learning methods. These background are theories and motivations which we believe are helpful to build up the intuition and insight of Monodepth2.

2.1 Traditional stereo matching

To find the 2D depth map for a 3D scene, stereo matching calibrates intrinsic parameters and extrinsic parameters of a pair of cameras (stereo). It is done by finding the reliable correspondence points between two images in a stereo pair, then establishing a common transformation scheme (Fundamental matrix). After that, using epipolar geometry, a scene point is computed as the intersection of two lines extending from the optical center and projected point of each camera.

It is worth noting that, from one single image, the machine can not infer the disparity (or depth map) of one scene, since there are infinitely possible disparities having the same projected coordinates on the 2D capture of the camera. In order to make predictions of depth map, further constraints must be imposed on the model. This problem is called scale ambiguity in depth estimation. Several methods involves using epipolar geometry (as discussed in this section), distance to vanishing point, and inverse depth parameterization.

Camera Calibration. Image formation of a 3D scene on a 2D stereo pair is usually modeled using the intrinsic parameters (such as focal length and optical center) and extrinsic parameters (relative position & orientation with respect to the other camera in a stereo pair). Intrinsic parameters are usually determined at the beginning by looking at the camera configuration. Some cameras put this information into the header of the image file and can be easily retrieved when needed, thus these parameters are not subject to calibration. Extrinsic parameters are different. There

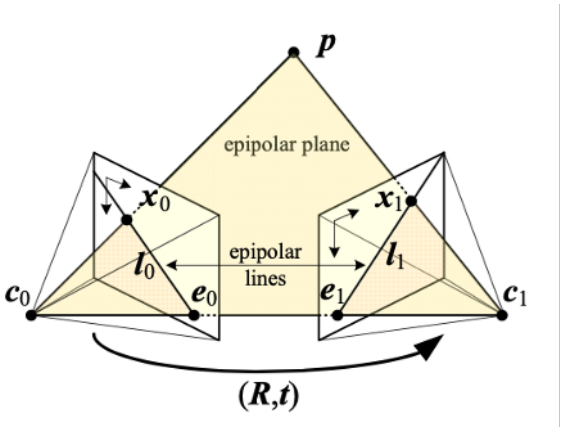


Figure 1: Epipolar geometry model between two cameras with c_0, c_1 as optical centers, x_0, x_1 as projected points. (image courtesy of [4])

are devices with a stereo pair installed at the factory, thus the extrinsic parameters can be computed in advance and come with the device's specification. However, it is more often that the images are taken from two distinctive cameras, whose parameters are not known between each other thus they require further calibration. It is shown in [5] that, using the epipolar constraints and singular value decomposition, the relationship between two 2D points on two cameras can be deduced into a single equation containing the fundamental matrix. This matrix encapsulates the information of intrinsic parameters and extrinsic parameters. Solving for the fundamental matrix is a least-squares problem that can be efficiently optimized once the correspondence points **u** and **v** are found.

Finding correspondence points. The correspondence points **u** and **v** are 2D points on each camera that representing the same 3D scene. The points believed to represent the same feature (sparse correspondence) of a scene are chosen to feed into the fundamental matrix equation to solve for the fundamental matrix. The process of finding sparse correspondence on two different cameras involves scanning on one image to see whether this pixel represents the same scene as the pixel on the other image in the stereo pair.

Computing the depth. After the fundamental matrix is found, points of one image can be inferred from the other image, and vice versa via epipolar lines (dense correspondence). Once the coordinates of every pair of points in the stereo are found, we can find the scene point in 3D coordinates by intersecting two rays coming from cameras.

2.2 Monocular depth estimation

Monocular depth estimation is a problem where the machine learning model makes inferences on the disparity (depth maps) from one single image. Although the term "monocular", we have learnt above that with a single image, the machine can not make a conclusion about the depth map of one scene. To achieve this goal, other constraints must be imposed on the model to limit the number of possible depth maps, and it is usually done via the cost function in self-supervised learning methods. In most cases, the model is actually trained using pairs of images itself. So, a monocular model still needs "stereo pairs" in some sense in order to achieve monocular prediction.

2.2.1 Overview of Learning-based Depth Estimation

Early approaches try to make inferences on the hyperparameters in a stereo pipeline. Proposal methods modeled stereo parameters as Probabilistic Graphs such as Markov Model or Conditional Random Fields [6] and were trained using pairs of images (stereo). Another approach is optimizing the matching cost, in which the model makes direct prediction on the disparity, then uses that disparity to compute the difference each pixel (or each neighborhood) between two images, the goal is to try to minimize that difference by optimizing the value of disparity [7]. Recent works on stereo pipeline focus on disparity selection among different views, and post-refinement of depth images. [8]. The problem of learning the parameters of the traditional pipeline is the images must be available in stereo form, i.e at least a pair of images, and it is by nature can not infer the depth maps from one single image, making monocular depth estimation not possible.
Another approach is to use end-to-end 2d Deep learning architectures. The features are learned within the deep network without the need of predefined parameters. Rather, the stereo information is used to construct the cost function to be

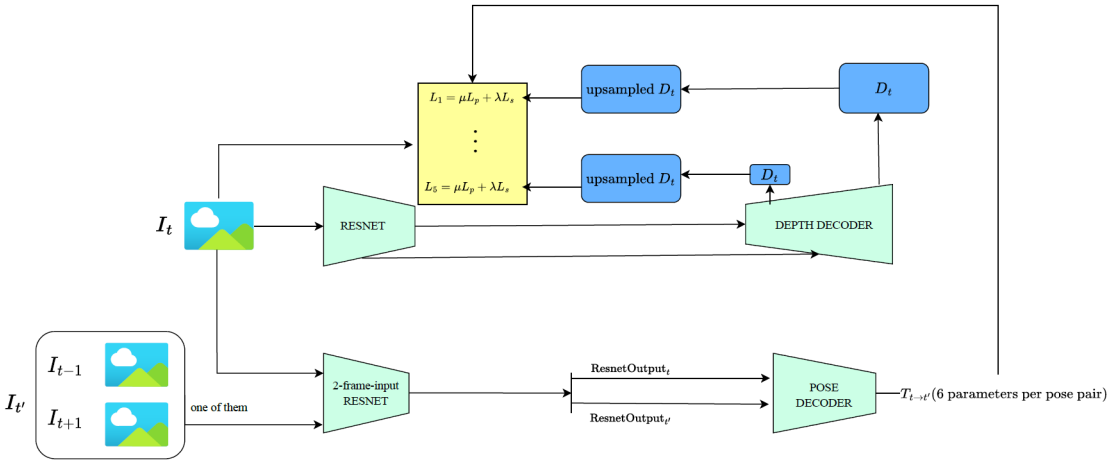


Figure 2: General architecture of Monodepth2

optimized. Residual [**?**] and Auto-encoder architectures are applied in inferring depth images [9] [10]. Yet, most of these approaches still adopted supervised methods, requiring the dataset to be labeled beforehand. Acquiring labeled depth (disparity) images in reality is usually a difficult task which may not be feasible in certain scenarios.

2.2.2 Self-supervised monocular depth estimation

The absence of ground-truth disparity images raised the problem of a self-supervised model for depth estimation. In this direction, the problem is usually formulated as an image reconstruction problem from estimated depth maps. One of the first proposal of a self-supervised monocular model involved inverse warping, in which the left image is reconstructed using the disparity prediction of itself and the right image. The reconstructed image is then compared with the ground-truth left image to evaluate the quality of depth prediction made earlier by the network, [11]. A later proposal added an additional pose prediction network in complement with the depth prediction network [12], with the goal to have more parameters of image warping to be learnable by the networks. These approaches had introduced interesting ideas to the field of self-supervised depth estimation. Another proposal explored a deeper insights of image warping by considering photometric consistency (Monodepth 1 [13]) within a warped stereo of a video sequence. Later on, the selection of adjacent frames to tackle occlusions problem in monocular depth estimation and stationary pixel auto-masking (Monodepth 2 [1]) to mitigate inherent problems of monocular images, this is the model we will discuss in the remaining part of this report. The latest contribution of this field was the MiDas [14] which is not a specific architecture but rather a set of methods to mix diverse datasets for training and evaluate on unseen datasets (zero-shot cross-dataset transfer) in order to achieve robust, well-generalized monocular depth estimation. In conclusion, self-supervised monocular depth estimation is an emerging novel approach to depth estimation, paving the way to explore a wider range of datasets for a variety of applications.

3 Architecture

3.1 Networks

The architecture of Monodepth2 consists of a U-Net like depth estimation network and a pose estimation network. For depth estimation network, the encoder employs the standard Resnet18 [15], while the decoder is a manually designed CNN, inheriting the ideas from Monodepth1 [13]. The network outputs a depth image at each layer which is then upsampled to the original size before feeding into the loss function. For pose estimation networks, the encoder uses a modified version of Resnet18 that could accept multiple input images, and the decoder is designed as a CNN accepting outputs from both depth and pose encoders. Although both encoders share several properties, the encoders were advised to be coded separately since it could produce higher performance than the shared option [14].

The flow of data through the model can be described as follows. (1) The frame at time t I_t is fed into the depth network. One of two adjacent frames (I_{t-1} or I_{t+1}, together with I_t are also fed into the pose network meanwhile. (2) Then, the final output of the pose encoder is pushed into the pose decoder. On the other hand, each layer of the depth decoder takes in output at each layer of the encoder and produces a depth image with according dimension, these output depths are then upsampled to the original frame resolution. (3) With the output disparity maps at immediate layers of depth

decoder and the output (as a single 6-parameters tensor containing relative pose between frames) of pose decoder, together they are used to compute the photometric reprojection error. Notably, the error was not averaged between frames, but rather, one of the adjacent frames is selected to minimize this error, this is proved [1] to bring significant improvements to performance.

3.1.1 Resnet encoders

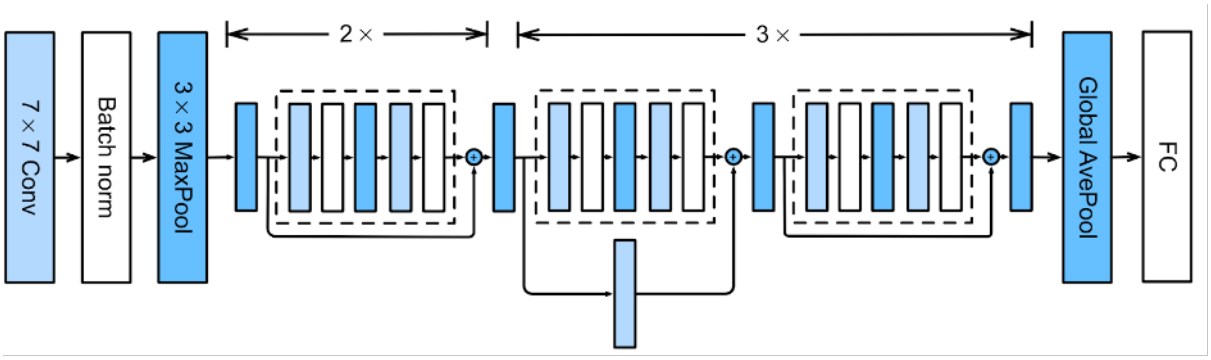


Figure 3: Architecture of original Resnet18, consisting of 8 building blocks. [15] (image courtesy of d2l.ai)

To avoid confusion in the terminologies, we treat 'a layer' as a stack of several building blocks. When 'a layer' is mentioned, we mean the group of these stacked building blocks. The data flows into the first building block, passes through multiple building blocks, and then through the last activation layer. We call such groups 'a layer'. This 'layer' is different from a functional layer like convolutional layer, activation layer or weighted layers where their functionality is clearly defined.

Depth encoder. Depth encoder is built using a standard Residual Neural Network with 18 weighted layers (Resnet18 [15]. In detail, the network contains a stack of building blocks, the input of one block is duplicated and accumulates to the output of the same block before passing into the next. One building block may contain several convolutional and activation layers, depending on the implementation. In Resnet18 particularly, apart from 3×3 convolutional layers, batch normalization is also applied to reduce the learning time, and ReLU is used in the activation layer. In Monodepth2, the output at each block (after activated using ReLU) is taken out to feed to the Depth decoder, which is different from the traditional use of Resnet where only the outputs after the final fully connected layer are considered. In the implementation, these outputs are returned from the network in the form of a list. Although it is 'standard Resnet18', the average pooling layer and fully connected layer are not used since we only need the ReLU output from the blocks, not the global pooling.

Pose encoder. Pose encoder also uses Resnet18 [15], but with little modification to accept multiple input images at once. Instead of a $3 \times 64 \times 3 \times 3$ convolutional layer (3 channels, 64 filters, 3×3 convolution kernel), the modified network added 3 additional channels, making the first convolutional layer to be $6 \times 64 \times 3 \times 3$. Another major difference from the depth encoder is that in this pose encoder, only the output of the final block is used to feed into the pose decoder.

3.1.2 Depth decoder

Depth decoder is a continuation of the depth encoder, together they form a U-Net like depth prediction network. In one depth decoder block, the image from the immediate lower block is fed into a convolutional layer as an input for this block. Then, the convoluted image is upsampled using nearest neighbor by a factor of 2. After that, the image is accumulated with another encoder's output of the same dimension, before being fed into a convolutional layer. From this convolutional layer, there are two flows of data: (1) the output image is passed on to the upper blocks of the decoder, and (2) the output image is fed into a convolution layer to find disparity image D_t with Sigmoid σ activation. Doing the same for other blocks, each block of the network will produce a corresponding disparity image. As a whole, the network returns a multi-scale set of depth images at different resolutions.

All convolutional layers in this decoder have 3×3 kernel size, stride 1. The number of channels in the output of blocks from top to bottom are predefined [1] as 256, 128, 64, 32, 16, respectively.

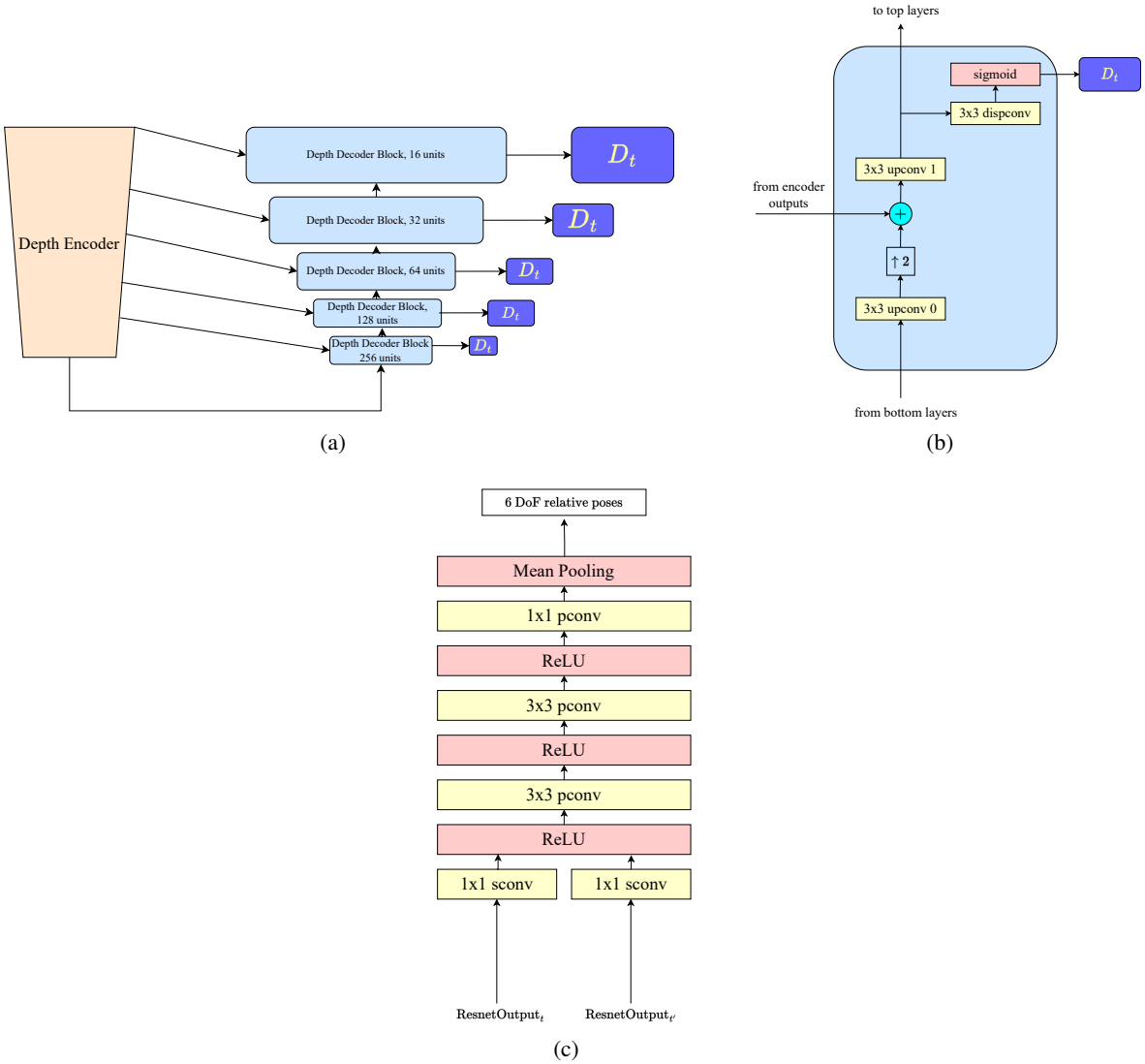


Figure 4: (a) The general architecture of Depth Decoder. (b) The structure of a block of Depth Decoder. (c) The architecture of Pose Decoder.

3.1.3 Pose decoder

The pose decoder learns the representation of relative pose between two frames as a pair of axis-angle, with 3 parameters for each property. As the pose encoder produces two outputs for a pair of frames, the pose encoder will take in that pair of frames. These frames are passed into two separate convolutional layers (squeezing-layers), then concatenated with each other to go through one single ReLU activation. The data propagates through several more convolutional and activation layers, before being averaged to output the final 6 Degree of Freedom relative pose parameters.

All layers have stride 1. The number of channels in the output of all convolutional layers are 256 except the last layer who has only 1 channel, corresponding to one single pair of image the network is trying to infer the pose for.

3.2 Loss function

Modifications in the loss function are the primary innovations of this paper [https://arxiv.org/pdf/1806.01260.pdf]. Three major proposal include: (1) minimum instead of average reprojection loss, (2) depth maps are upsampled back to original resolution before computing the loss, and (3) auto-masking for static pixels in a monocular image sequence. In

this section, we will explore the construction of the loss function for self-supervised learning from monocular sequences with a top-down perspective.

The total training loss function is described as:

$$L = \mu * L_p + \lambda * L_s$$

where, μ is the mask to filter out all static pixels in a monocular sequence, L_p is the photometric reprojection loss, λ is smoothness term (chosen as $\lambda = 0.001$) and L_s is smoothness loss.

3.2.1 Photometric reprojection loss L_p

The photometric reprojection loss is defined as the photometric reprojection error between target I_t and warped $I_{t'\to t}$, where $I_{t'} \in \{I_{t-1}, I_{t+1}\}$ is an adjacent frame that minimizes this error.

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

Image reprojection $I_{t'\to t}$

$$I_{t'\to t} = I_{t'} \langle proj(\, D_t, T_{t\to t'}, K) \,\rangle$$

A reprojection (warping) of image I'_t on target I_t is done by sampling I'_t based on the projected depth map. In Monodepth2, we perform monocular training in which only a monocular sequence of images of one view is fed into the networks, i.e: $I_{t'} \in \{I_{t-1}, I_{t+1}\}$. Note that the depth map is first upsampled back to the original resolution (same resolution as the original I_t) before performing the projection. This modification helps to avoid inconsistent contribution of loss between outputs at different layers, as well as incomplete low-resolution depth maps to affect the final output. Upsampling process employs bilinear interpolation to fill up the holes of the upsampled image.

Photometric reprojection error $pe(I_a, I_b)$

$$pe(\, I_a, I_b) \,= \frac{\alpha}{2}(\, 1 - SSIM(\, I_a, I_b) \,+ (1-\alpha)\|I_a - I_b\|$$

Photometric reprojection loss is defined as a weighted loss of complement of SSIM and L1-norm between two images. The weight $\alpha = 0.85$ is chosen by the author. *Structural Similarity Index Measure* (SSIM) [16] is a weighted index of luminance, contrast and structure to evaluate the similarity between two images. There are several ways to compute this index, but in Monodepth2, the author chose the simplified version, in which all the weights of luminance, contrast and structure are assumed to be equal. As a result, the equation for SSIM could be deduced as

$$SSIM(x, y) = \frac{(\, 2\mu_x\mu_y + C_1)\,(\, 2\sigma_{xy} + C_2)}{(\, {\mu_x}^2 + {\mu_y}^2 + C_1)\,(\, {\sigma_x}^2 + {\sigma_y}^2 + C_2)}$$

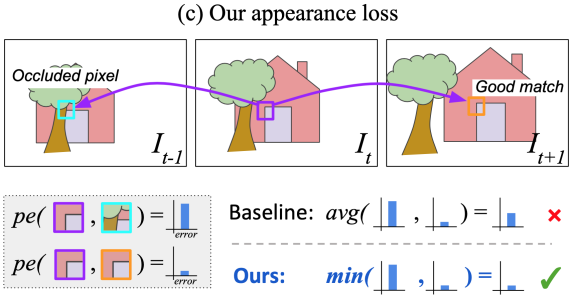
where, $C_1 = 0.01^2, C_2 = 0.03^2, \mu$ is computed with block filter.

The goal of photometric reprojection error is to penalize the difference between the original image I_t and the reconstructed image $I_{t'}$ base on pose and depth. The better pose predicted and depth map inferred, the more similar the original and reconstructed are.

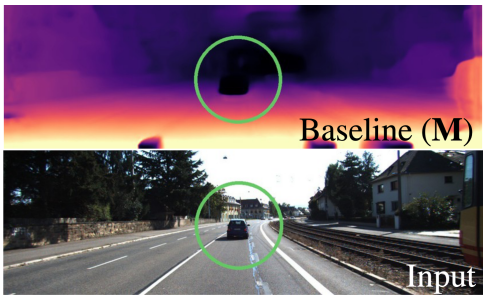
Loss as minimum error Instead of averaging the error in a group of two adjacent frames, the minimum one is selected as the representative error. This helps to mitigate the problem of pixel occlusions between adjacent frames, which is common in training a depth estimation model using monocular sequences. Pixel occlusion is a scenario where the reconstructed frame has significant differences compared to the original due to pixels of the adjacent image being occluded by some object. In terms of monocular training, occlusion of adjacent frames, by nature, does not mean the reconstructed image from depth maps is bad, because there is no changes in the ground-truth depth of one frame when its background is occluded. Naively averaging the errors can be over-sensitive, leading to high errors overall, and the quality of depth estimation of one frame is not correctly reflected.

3.2.2 Stationary pixel auto-masking μ

In recent self-supervised monocular models, a common assumption is the camera is moving and the scene is static. An emergent problem from this assumption is when the object moves at the same velocity with the camera, that object seems to "stand still". If an object of one frame stays static as compared to the adjacent frame, the model will perceive



(a) **Loss as minimum error.** The model chooses an adjacent frame that minimizes the photometric reprojection error instead of averaging all the errors. This modification allows the depth prediction of one frame to be more accurately reflected, i.e we do not falsely penalize predictions that are actually good. (image courtesy of [1])



(b) **Stationary pixel problem.** Problem of moving objects that violates assumptions of many monocular models. Intuitively, the car is pretty close to the camera in the scene, but since it is moving at the same velocity, the model perceived it as a "very far" object, leading to a hole on the depth map. (image courtesy of [1])

Figure 5: Loss as minimum error and Stationary pixel problem

it as an infinitely far away object from the camera, having a very "deep" value in the depth map. This problem is shown in practice as disappearing objects on the depth map (see 5).

To tackle this problem, Monodepth2 introduced an auto-masking mechanism to filter out stationary pixels between a target image and its adjacent frame. The mask is computed by comparing the errors between the reconstructed image and the actual adjacent frame. If the error of the adjacent frame $I_{t'}$ relative to I_t is too low, the pixel appears to be static from frame to frame, such pixels can contaminate the loss with infinite depth, therefore we assign it with value 0 on the mask to stop the gradient flowing through this pixel and stop the update at the pixel in a batch. The formulation for the mask is described as follows, the result of this mask would be a tensor with the same dimension as images, containing $0s$ and $1s$.

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

The error $pe(I_t, I_{t'})$ was not compared with any predetermined threshold but rather compared with the error of the reconstructed image. This is an interesting insight. At the beginning, the error $pe(I_t, I_{t'\to})$ tends to be very high since initial predictions are usually bad, in such cases, the pixels are only masked as 1 if there is a big difference between I_t and warping source $I_{t'}$, that could potentially have larger error than the reconstruction. This means that in the very beginning, the model only learns a few pixels that change significantly and ignores the pixels that are relatively static. As prediction gets better, i.e $pe(I_t, I_{t'\to})$ gets lower on a whole, the model starts to consider more stationary pixels, and the mask is only 0 if the warping frame I'_t is perfectly the same as the target image I_t. By doing this, the mask adapts to learn more pixels as the training process proceeds instead of being rigidly set from the beginning.

Although this auto-masking setting helps to overcome assumption violation, we argue that there are drawbacks. While this technique performs well on the intended dataset (KITTI [2] which has drastically changing scenes of traffic videos, allowing the model to learn the most dynamic pixels at the beginning, it may not perform well on datasets where the scenes do not change much frame to frame such as NYUv2 (indoor scenes) [17] where the pixels are mostly static, or shift very slowly compared to the previous frame. The assumption that the camera is moving and the scene is static is also harder to be violated for indoor scenes, making this technique not appropriate for such datasets.

In the experiments, we will demonstrate our results when auto-masking is disabled on NYUv2 datasets.

3.2.3 Smoothness loss L_s

$$L_s = |\partial_x d_t^*| e^{-|\partial_x I_t|} + |\partial_y d_t^*| e^{-|\partial_y I_t|}$$

In order to encourage locally smooth depth maps, another loss that penalizes the depth gradients is introduced. In details, the loss is the sum of L1-norm of depth gradients in x and y direction, each is inversely scaled by the magnitude of image edge in the same direction. That is, if the pixels on the original image are constant fields, the loss (as L1-norm depth gradient) will be large. This mechanism helps the depth map within an object smooth with little artifacts, while still keeping clear edges between them. Note that smoothness loss is not scale-invariant, smaller scale always results in smaller smoothness loss, thus posing scale ambiguity problems. Therefore, in the setting of smoothness loss, the author

chose normalized depth map over mean $d_t^* = d_t / \bar{d}_t$ instead of pure d_t, this is empirically proved in [12] to reduce the effects of scale ambiguity in monocular depth estimation.

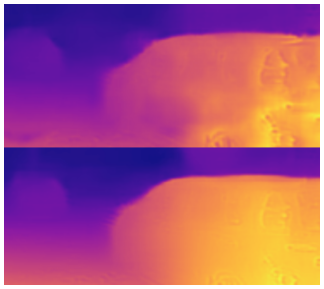


Figure 6: Up: the depth map without smoothness loss, the structure of the car is distorted, Bottom: the depth with smoothness loss, the internal structure is preserved. (image courtesy of [12])

4 Experiments

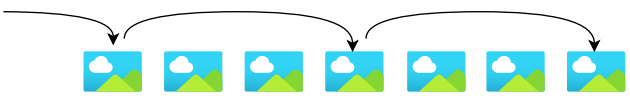
We perform experiments on the Monodepth2 model by training the networks from scratch and from checkpoint. Two datasets we are considering is KITTI [2] and NYUv2 [3], which are two popular depth estimation benchmarks. Due to limited computing resources, we could only train the model on a smaller subset of the above datasets. The training is done from scratch for KITTI datasets, and from our KITTI checkpoint for NYUv2. We observed better performance while doing transfer learning from KITTI to NYUv2, but not vice versa. The performance may be far behind from the author's implementation, but we believe it is sufficient to demonstrate the idea of this model.

4.1 Datasets

Since the datasets are quite large to be stored on our Google Drive for a long time, we are sorry that we are not able to keep it in our repository for a period before it is deleted. Meanwhile, it is still accessible via this link. We show below how we gather and construct the datasets so that they can be re-created in the same manner.

We use 2 datasets for this project. First, for the KITTI dataset, the selected data is downloaded from the Raw Data of website The KITTI Vision Benchmark [2]. With each video clip, there is a folder with the name under the format of *[recorded_date]_drive_[record_order]_sync*. The datasets are selected randomly from different scenes.

In a video folder, there are 6 folders including `image_00`, `image_01`, `image_02`, `image_03`, `oxts`, `velodyne`, in which, `image_00` and `image_01` are 2 folders of black-and-white images and `image_02` and `image_03` are 2 folders of colored images with right and left camera angle. We chose to use the folder `image_02`, `image_03` and velodyne for training. For every 3 continuous frames of a scene, we only keep the first frame and remove the next 2 frames. That efficiently sizes down the dataset while still keeping the major changes between frames for depth prediction. After sizing down the original data, our dataset contains 4634 images: 3570 images for training, 864 images for validation and 200 images for testing. The next step is renaming for the file with the ID format of 10 digits, and order starting from 0. Lastly, we create 2 text files for `train.txt` and `test.txt` with a line of format *[folder_name] [imageID] [l or r]*. In `train.txt`, these lines will be shuffled while the `test.txt` will be kept in the original order.



For the NYUv2 dataset, we downloaded a smaller version of NYUv2 [Kaggle NYUv2]. This dataset contains 50,688 images. We also splitted it into 44899 training items, 5589 validation items and 200 testing items. Initially, each video's folder will only have a series of images with name extension `.jpg` for normal images, and `.png` for depth images. Therefore, we design to classify them into 2 folders: `depth` and `image` . If image and depth belong to one frame, they will have the same file's name in 10 digits format. Finally, we create 2 files: `train.txt` and `test.txt` with a proportion of 10% for test set. Both files have content in format: *[folder_name] [ID] [l]*, and these lines in the train file are shuffled.

You can refer to our dataset [our data link], which is actually our dataset used for training.

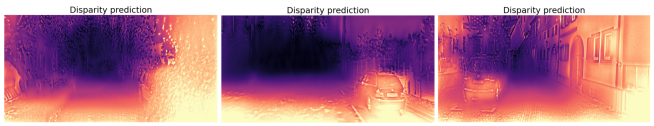


Figure 7: The training process for KITTI as recorded at batch 50, batch 150, batch 600

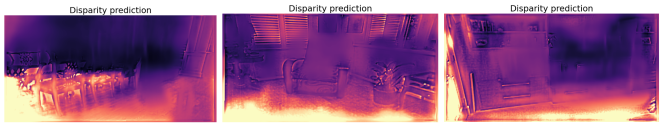


Figure 8: The training process for NYUv2 as recorded at training progresses

4.2 Model setting and Training

The authors of Monodepth2 tried to implement many variants of their model to conduct ablation studies on each characteristic of the model. In this experiments, we chose to go with one direction of the implementation, including the core ideas to demonstrate the performance of Monodepth2.

KITTI model: We decided to reimplement the model major ideas using non-pretrained, separated ResNet18-encoders, with SSIM loss, auto-masking, min-reprojection, and full-res multi-scale. ResNet50 gets better results but at the expense of longer training and test times, we did not choose this option. In the training process, we used free T4 GPU Google Colab with limited computation (and CPU if we reach the limit of the day) to train our model. We chose Adam Optimizer with batchsize 1, and resize every image to 640x192 resolution. We save and validate the model every 50 batches.

NYUv2 model: Most of the basic configurations on the KITTI model are kept to train NYUv2 datasets. However, we make several modifications for this model to fit this new dataset. (1) Due to the argument we made above about auto-masking, we decided to disable auto-masking while training with this dataset. (2) We also have to change the intrinsics matrix K since the cameras used to capture the scenes in KITTI and NYUv2 are not the same, in particular NYUv2 used [Microsoft Kinect cameras] which has the calibrated intrinsics matrix

- $f_{x\_RGB} = 5.1885790117450188e + 02$
- $f_{y\_RGB} = 5.1946961112127485e + 02$
- $c_{x\_RGB} = 3.2558244941119034e + 02$
- $c_{y\_RGB} = 2.5373616633400465e + 02$

as described in `camera_params.m` of Matlab toolbox of NYUv2, and (3) we perform transfer learning from our model trained with KITTI from sratch earlier to continue the training on NYUv2, this empirically produces better results than training the model from scratch.

4.3 Result

After training, we have 2 versions of models: KITTI Model and NYUv2 Model. We tested our models and got a comparison between them on 2 test sets: 200 images KITTI test set and 200 images NYUv2 test set. There is our result:

Model	Kitti test set	NYUv2 test set	abs_rel	sq_rel	rms	log_rms	σ1	σ2	σ3
Kitti Based	✓		0.142	1.183	5.432	0.231	0.834	0.934	0.969
		✓	0.291	8.200	22.858	0.351	0.481	0.780	0.928
NYUv2 Based	✓		0.169	1.545	5.788	0.250	0.806	0.927	0.966
		✓	0.316	9.213	24.114	0.386	0.425	0.737	0.908

Figure 9: Result Table

From the result table, we can see that our Kitti model works well on both datasets and easily converges in the training section. On the other hand, in the training phase, we found that it's very hard to fit the NYUv2 dataset and the result from this model is not as good as the Kitti model. The figure 10, 11, 12 and 13 shows us some result images of these models on 2 datasets in testing phase.

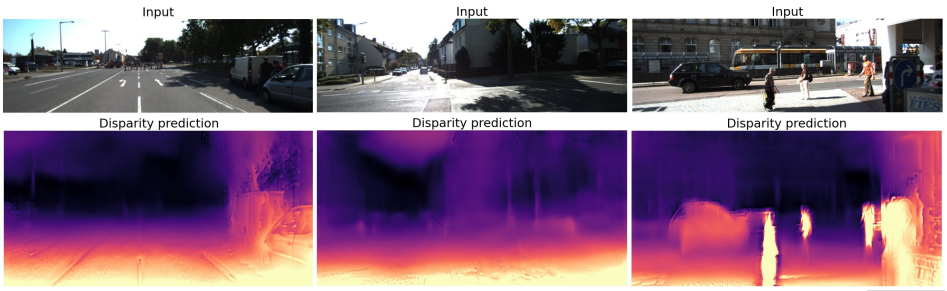


Figure 10: Kitti model tested on Kitti test set

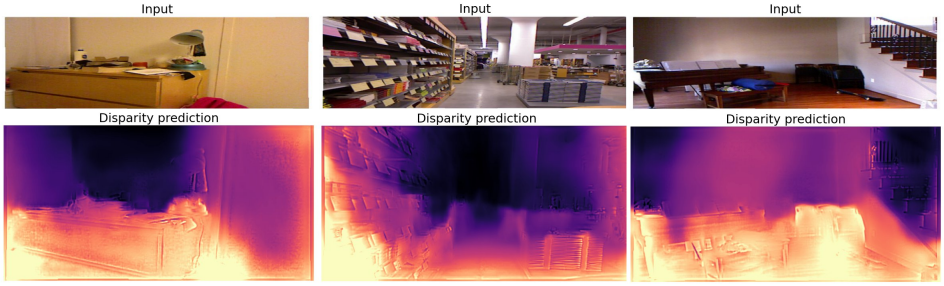


Figure 11: Kitti model tested on NYUv2 test set

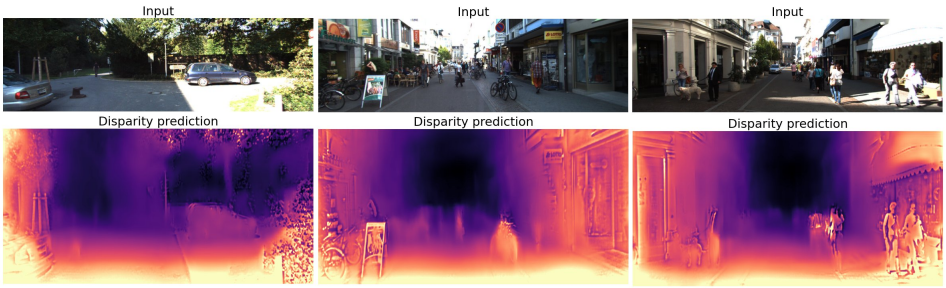


Figure 12: NYUv2 model tested on Kitti test set

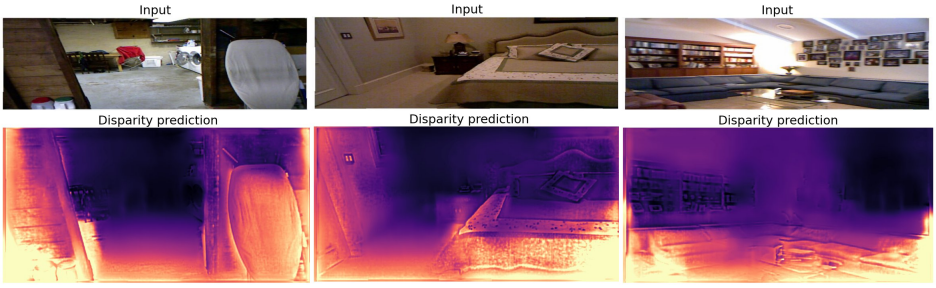


Figure 13: NYUv2 model tested on NYUv2 test set

4.4 Recommendations for improvement on NYUv2 model

Apart from the modifications such as auto-masking as we described above, there are also a few suggestions that we believe will help Monodepth2 fit better with NYUv2 and other indoor scenes overall.

As described in [18], the major challenges of self-supervised methods for monocular depth estimation based on depth and pose networks are: (1) in-consistent depth between different scenes, there is no "horizon" in indoor images, making

it difficult to decide what are the furthest points of a scene, and (2) rotational motion is dominant in indoor scenes in contrast with translational motion for outdoor scenes, raising difficulties for existing configuration of pose networks. In the same paper, the author introduced several novel methods to improve indoor predictions of existing outdoor-by-design models. The first is depth factorization, this new setting helps to overcome the problem of variable "deepest depth" for indoor scenes, and promote adaptive depth range (factorization) over the scenes. A new self-attention network was introduced , the output of this network is then fed to a probabilistic regression model to make inference on global scale of the scene. The second is using a residual pose network instead of pure auto-encoder network to overcome failure cases of existing pose networks when working with rotation-dominant poses. This technique basically advocates the learning of rotational parameters a residual pose network. In fact, the pose prediction propagates through two pose networks instead of just one in Monodepth2. These modifications help the model the perform better for indoor scenes.

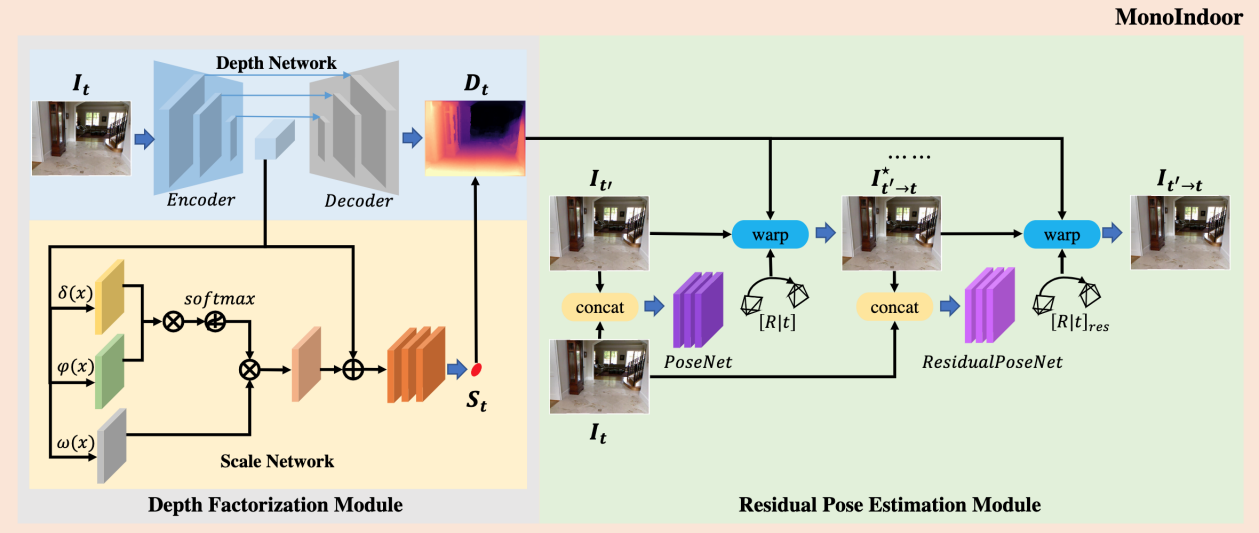


Figure 14: Modifications of existing Monodepth2 pipeline (MonoIndoor) help to overcome inherent problems of monocular depth estimation models who did not generalize well for indoor scenes.

5 Conclusion and Acknowledgements

During the course of this project, we had learnt so much about how to read a paper and implement it. We also had useful discussion with each other about the potentials, drawbacks of the model not only with the teammates but also members of other teams to learn from each other. We learnt how to read a complex code base about Machine Learning and try to deduce the most relevant features of it. Above all, we start to realize the need of a solid knowledge foundation from the very beginning which can only be accumulated during the first and second year.

We also faced so much difficulties during the execution, including lack of knowledge on the existing Computer Vision field, lack of understanding on state-of-the-art deep learning models and architecture, as well as too small practical activities on common deep learning frameworks making the implementation a very stressful experience. Also, we were confused and scared of large datasets and make incorrect judgements which reduced the morale of the team. After all, there is still a lot to be learn and improve before we create something useful.

We, Le Tran Nguyen Khoa, Nguyen Dang Hoai Nam and Dinh Viet Thanh would like to express our gratitude to Dr. Nguyen Tien Thinh who raised a very interesting problem for us to explore. This is the very first step for our interest to dive deeper in Machine Learning world.

References

[1] Clément Godard, Oisin Mac Aodha, and Gabriel J. Brostow. Digging into self-supervised monocular depth estimation. *CoRR*, abs/1806.01260, 2018.

[2] 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.

[3] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[4] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer Cham, 2nd edition, 2023.

[5] Quang-Tuan Luong and Thierry Viéville. Canonical representations for the geometries of multiple projective views. *Comput. Vis. Image Underst.*, 64:193–229, 1996.

[6] Li Zhang and Steven M. Seitz. Estimating optimal parameters for mrf stereo from a single image pair. 29(2):331–342, feb 2007.

[7] Jure Zbontar and Yann LeCun. Stereo matching by training a convolutional neural network to compare image patches. *CoRR*, abs/1510.05970, 2015.

[8] Johannes L. Schönberger, Sudipta N. Sinha, and Marc Pollefeys. Learning to fuse proposals from multiple scanline optimizations in semi-global matching. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pages 758–775, Cham, 2018. Springer International Publishing.

[9] Jiahao Pang, Wenxiu Sun, Jimmy S. J. Ren, Chengxi Yang, and Qiong Yan. Cascade residual learning: A two-stage convolutional neural network for stereo matching. *CoRR*, abs/1708.09204, 2017.

[10] Philipp Fischer, Alexey Dosovitskiy, Eddy Ilg, Philip Häusser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. *CoRR*, abs/1504.06852, 2015.

[11] Ravi Garg, Vijay Kumar B. G, and Ian D. Reid. Unsupervised CNN for single view depth estimation: Geometry to the rescue. *CoRR*, abs/1603.04992, 2016.

[12] Chaoyang Wang, José Miguel Buenaposada, Rui Zhu, and Simon Lucey. Learning depth from monocular videos using direct methods. *CoRR*, abs/1712.00175, 2017.

[13] Clément Godard, Oisin Mac Aodha, and Gabriel J. Brostow. Unsupervised monocular depth estimation with left-right consistency. *CoRR*, abs/1609.03677, 2016.

[14] Katrin Lasinger, René Ranftl, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *CoRR*, abs/1907.01341, 2019.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[16] 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.

[17] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[18] Pan Ji, Runze Li, Bir Bhanu, and Yi Xu. Monoindoor: Towards good practice of self-supervised monocular depth estimation for indoor environments. *CoRR*, abs/2107.12429, 2021.