VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

PROGRAMMING INTERGRATION PROJECT (CO3101)

A reimplementation of monodepth2

Advisor: Mr. Nguyễn Tiến Thịnh
Students: Nguyễn Đình Thiên Huy - 2152591
Lâm Gia Khánh - 2152651
Lê Quang Minh - 2152760
Lê Trọng Đức - 2152523

HO CHI MINH CITY, NOVEMBER 2023

Contents

Member list & Workload

No.	Fullname	Student ID	Problems	Percentage of work
1	Nguyễn Đình Thiên Huy	2152591	- Train the model - Reimplement the model - Evaluate the model - Tasks distribution	25%
2	Lê Trọng Đức	2152523	- Understand the loss function - Research reference papers - Write report	25%
3	Lê Quang Minh	2152760	- Evaluate the model - Process data loader - Write report - Reimplement the model	25%
4	Lâm Gia Khánh	2152651	- Reimplement the model - Prepare data - Write report	25%

1 Introduction

Determining distance in relation to a camera poses from a single color input image without a second input image to enable triangulation is a persistent challenge, yet it is indispensable for unlocking intriguing applications like autonomous driving, 3D scene reconstruction, and augmented reality (AR). In robotics, acquiring depth is a fundamental requirement for executing diverse tasks, including perception, navigation, and planning.

In deciphering ego-motion and grasping the three-dimensional arrangement of a scene, humans showcase remarkable capabilities, even when considering short timeframes. Take, for instance, our adeptness in navigating a street, effortlessly identifying obstacles, and swiftly responding to avoid them. Despite extensive research in geometric computer vision, attempts to replicate equivalent modeling capabilities for real-world scenarios, marked by non-rigidity, occlusion, and a lack of texture, have proven unsuccessful. The question arises: what sets humans apart in excelling at this task? One plausible explanation suggests that our proficiency stems from cultivating a comprehensive, structural understanding of the world through our cumulative visual experiences. These experiences primarily involve navigating diverse environments and observing a multitude of scenes, allowing us to establish consistent models based on our observations. Through millions of such instances, insights into the regularities of the world have been acquired—recognizing that roads are typically flat, buildings exhibit straight lines, and cars are supported by roads. This accumulated knowledge empowers us to apply a learned framework when interpreting a new scene, even when presented with a solitary monocular image.

Due to challenges in acquiring training datasets that provide accurate ground truth depth for supervised learning, scientists are endeavoring to replicate the human capacity to infer ego-motion and 3D structure. Several recent self-supervised approaches have shown that it is instead possible to train for monocular depth estimation can be achieved using solely synchronized stereo pairs or monocular video.

MonoDepth2 extends the capabilities of its predecessor, MonoDepth, which employed self-supervised learning on paired stereo images. MonoDepth introduced a neural network architecture capable of end-to-end unsupervised monocular depth estimation, featuring a unique training loss mechanism that enforces consistency of depth predictions between the left and right views within the network.

MonoDepth2, instead, prioritizes self-supervised learning using monocular video datasets, a more accessible alternative to acquiring synchronized stereo images. This approach not only simplifies the data acquisition process but also removes the necessity for ground truth labels typically required in supervised learning. MonoDepth2 introduced three innovations: an appearance matching loss to handle occlusion, an auto-masking approach for stationary pixels and a technique to reduce depth artifacts by upsampling depth predictions at intermediate layers and compute all losses at the input resolution.

2 Model

The Depth Network and the Pose Network are the two separate parts that make up the whole network. Although the ResNet18 encoder is used by both networks, their decoder designs are different. The next sections will go into more detail on each decoder.

2.1 Depth Estimation

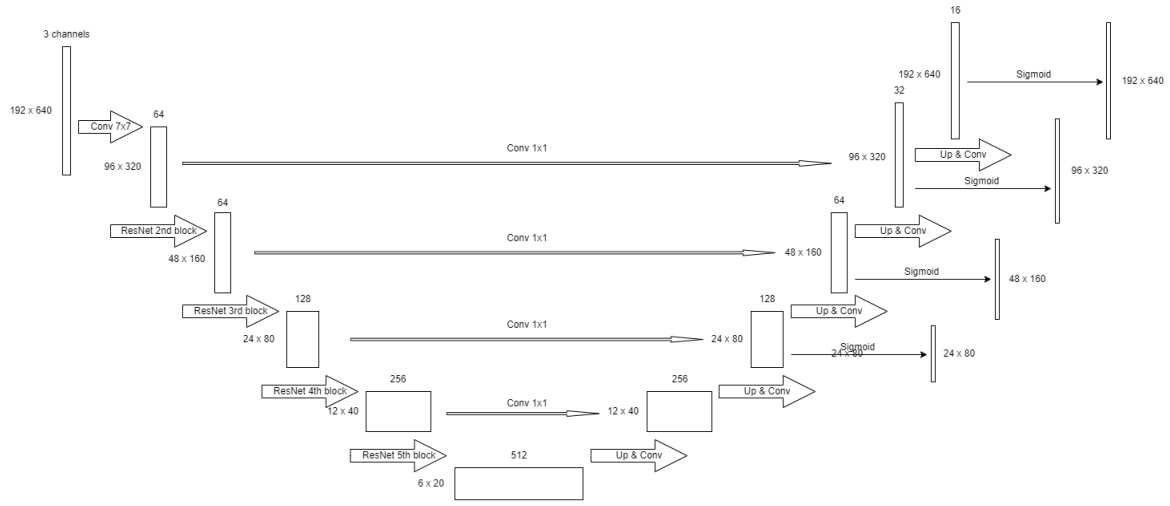

Figure 1. Depth Estimation model

The deep network employs the U-Net architecture, characterized by a fully convolutional network with skip connections. U-Net is the most widely used structure for depth estimation since it consists of an encoder–decoder with skip connections, allowing for efficient information flow and performing well when the large dataset is unavailable [1]. The kernel size for all layers are 3x3, except the first layer of the ResNet18, which is 7x7. The width and height of feature maps are reduced/increased by the factor of 2 through each layer.

There are diverse encoder options available for depth estimation models. For instance, [2] utilizes DispNet with multi-scale side predictions, while MonodepthV2 opts for the ResNet architecture to encode input images. Given the superior accuracy of MonodepthV2, we have chosen the ResNet architecture. The encoder, taking a single image as input, leverages ResNet18 to extract features. The resulting features are then fed into the decoder block, consisting of four blocks aligned with the "scale" or convolutional blocks in ResNet18. Within each block, two convolutional layers are applied, each followed by the ELU activation function.

To elaborate further, the decoder is structured with a series of up-convolution layers. The initial up-convolution layer processes the input feature map from the previous decoder stage, performing upsampling through transposed convolution. This operation doubles the spatial resolution of the feature map. The subsequent upconvolution layer takes the output of upconv_0 and, if it is not the first scale, concatenates it with the corresponding feature map from the encoder. This establishes a skip connection, enabling the decoder to leverage high-resolution information from the encoder. For each block (scale), a disparity map is generated by passing through a

3x3 convolutional layer and applying a sigmoid function on all of the feature maps. It's noteworthy that the output of this layer is not propagated as the input to the upper scale. Instead, it is recorded as an output for use in calculating the loss, a process to be covered in a later section.

Above is a brief description of the flow of the depth network in our model. For the network details, please refer to the network detail section below.

2.2 Camera Transformation Estimation (Camera Pose Estimation)

Normally, if the model is trained using a stereo image dataset like in monodepth1, the disparity can be calculated by the horizontal difference between two images. With monocular depth estimation aimed at the video sequence, the disparity has to be calculated by the horizontal difference of a feature between consecutive video frames. The main issue is that 2 consecutive frames in a video sequence and 2 images taken from different perspectives at the same time are two different matters. Given a stereo dataset with data about the difference in position and rotation between the two perspectives, we can estimate depth easily. Video sequence doesn't have such data.

Unfortunately, there is no solution that can completely remedy the problem. Instead, leaning toward minimizing such issues, this task has to be solved by relying on training another model for estimating camera transformation. The idea here is after getting the depth map from the depth network and creating a point cloud map for each pixel from the depth map, the point cloud map will be updated again with the matrix containing data about camera transformation between two frames. With the updated point cloud map and the source frame, the model tries to predict the target frame as accurately as possible using inverse image warping with bilinear interpolation. For how accurate the predicted target frame is compared to the actual target frame, loss, which will be mentioned in the next section, shows that and it is also used for training the camera pose estimation model. To make it easier to predict ego-motion, we mainly interested in dealing with video sequence that has motion mainly came from camera motion and has little to no motion came from each individual objects in every frame.

For network's input, it consists of features of 2 consecutive images in the video sequence. Its output is the camera transformation between 2 frames (ego-motion, parameterized as a 4x4 transformation matrices)

The model utilizes 4 layers of CNN(Convolutional Neural Network). The first layer is used for getting the essential features in the target frame and the source frame. The next 3 layers use the output of the previous layer to get the difference in the camera's position and rotation between 2 frames. After that, the difference will be translated into a matrix containing data about camera transformation. Through many research and experiments, using 4 layers and multiplying the output by 0.01 yields the best result without causing problems such as overfitting. All layers use ReLU as an activation function.

Notes in implementation:

For image features input, the model reuses the features from the output of the depth encoder.

Due to using a three-frame sequence that includes a past, a present, and a future frame for input and the past and future frames serving as source frames for calculating loss, there will be 2 estimations, 1 for transformation from the present frame to the past frame, 1 for transformation from the present frame to the future frame.

2.3 Network Detail

Encoders for both network are resnet18.

Depth Decoder						
Layer	**k**	**s**	**channels**	**res**	**Input**	**Activation**
upconv5	3	1	256	32	econv5	ELU
iconv5	3	1	256	16	↑upconv5, econv4	ELU
upconv4	3	1	128	16	iconv5	ELU
iconv4	3	1	128	8	↑upconv4, econv3	ELU
disp4	3	1	1	1	iconv4	Sigmoid
upconv3	3	1	64	8	iconv4	ELU
iconv3	3	1	64	4	↑upconv3, econv2	ELU
disp3	3	1	1	1	iconv3	Sigmoid
upconv2	3	1	32	4	iconv43	ELU
iconv2	3	1	32	2	↑upconv2, econv1	ELU
disp2	3	1	1	1	iconv2	Sigmoid
upconv1	3	1	16	2	iconv2	ELU
iconv1	3	1	16	1	↑upconv1	ELU
disp1	3	1	1	1	iconv1	Sigmoid

Pose Decoder						
Layer	**k**	**s**	**outch**	**dwf**	**Input**	**Activation**
pconv0	1	1	256	32	econv5	ReLU
pconv1	3	1	256	32	pconv0	ReLU
pconv2	3	1	256	32	pconv1	ReLU
pconv3	1	1	6	32	pconv3	-

Where

- k is kernel size.
- s is stride.
- outch is the number of output channels.
- dwf is the downscaling factor for a layer relative to its input image.
- input of each leyer if denoted ↑ means it is a 2x nearest-neighbor upsampling of the layer.

3 Loss function

In the monodepth2 model, the authors proposed three loss innovations that lead to large improvements in monocular depth estimation :

- A novel appearance matching loss to address the problem of occluded pixels that occur when using monocular supervision.
- A novel and simple auto-masking loss to ignore training pixels that violate camera motion assumptions.
- A multi-scale appearance matching loss to reduce the depth artifacts.

Before diving into those formulas, we briefly review the existing literature on the subject of neural networks for image processing and the subject of measures of image quality.

3.1 Preliminaries

3.1.1 Structural Similarity Index - SSIM

The term Structural Similarity Index (SSIM) was first introduced in the 2004 IEEE paper [3], "Image Quality Assessment: From Error Visibility to Structural Similarity".

During processing, compression, or reproduction, digital images are prone to a wide range of distortions, any of which might degrade visual quality. To measure the degradation, the traditional framework was to quantify the visibility of errors between a distorted image and a reference image, however, they have shown areas for improvement due to relying on several strong assumptions and generalizations. In this article, the authors introduce an alternative complementary framework for quality assessment based on the degradation of the structure of the image.

The authors have made some essential points. Firstly, most image quality assessment techniques rely on quantifying errors between a reference and a sample image. A common metric is to quantify the difference in the values of *each of the corresponding* pixels between the sample and the reference images. For example, the *mean squared error* (MSE) was one of those popular metrics. The majority of the proposed models have followed a strategy of modifying the MSE measure so that errors are penalized by their visibility. These measurement were widely used since they are simple to calculate and optimize.

$$MSE = \frac{1}{h * w * c} \sum_{i=0}^{h-1} \sum_{j=0}^{w-1} \sum_{k=0}^{c-1} [img(i,j,k) - imgRef(i,j,k)]^2 \tag{1}$$

Equation (1) is the formula of the MSE when comparing the two images of size $h * w * c$ pixels. However, some limitations appear with these error-sensitivity approaches: The Quality Definition Problem (mention in [4]); The Suprathreshold Problem; The Natural Image Complexity Problem; The Decorrelation Problem; and The Cognitive Interaction Problem. All these problems are mentioned in these articles here.

Secondly, based on the fact that natural images are highly structured (pixels have strong connections when they are close together) and the *human visual perception system (HVS)* is highly adapted to extract structural information from the viewing field, hence identifying the *differences between the information* extracted from a reference and a sample scene. Hence, a metric that replicates this behavior will perform better on tasks that involve differentiating between a sample and a reference image.

The SSIM Index

The Structural Similarity Index (SSIM) metric extracts 3 key *features* from an image:

- Luminance
- Contrast
- Structure

The comparison between the two images is performed based on these 3 features. Fig 1 given below shows the arrangement and flow of the Structural Similarity Measurement system. Signal X and Signal Y refer to the Reference and Sample Images.

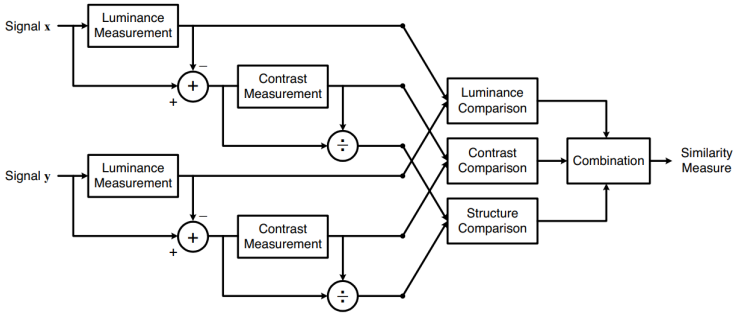

Figure 2. The Structural Similarity Measurement System

This system calculates the Structural Similarity Index between 2 given images which is a value between -1 and +1. A value of +1 indicates that the 2 given images are **very similar or the same** while a value of -1 indicates the 2 given images are **very different**. Often these values are adjusted to be in the range [0, 1], where the extremes hold the same meaning.

Now, let's explore how these features are represented mathematically and how they contribute to the final SSIM score.

First, the **luminance** of an image signal x is measured by averaging over all the pixel values in that image. It is denoted by μ and the formula is given below,

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i$$

Secondly, the **contrast** of an image signal x is measured by taking the *standard deviation* of all the pixel values. It is denoted by σ and represented by the formula below,

$$\sigma_x = \left(\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)^2\right)^{\frac{1}{2}}$$

Finally, the **structure**, The structural comparison is done by using a consolidated formula (more on that later) but in essence, we divide the input signal with its standard deviation so that the result has unit standard deviation which allows for a more robust comparison.

$$(x - \mu_x)/\sigma_x$$

That was how to calculate the three key features luminance, contrast, and structure of an image signal. To compare the two signal images, we continue with the comparison functions that can compare the two given images on these parameters. Suppose x and y are two non-negative image signals, which have been aligned to each other.

The **luminance comparison function** $l(x, y)$ between two image signals x and y is defined by

$$l(x, y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}$$

where C_1 is a constant to ensure stability when the denominator becomes 0. C_1 is given by,

$$C_1 = (K_1 L)^2$$

L is the dynamic range for pixel values (we set it as 255 since we are dealing with standard 8-bit images). K_1 is a small constant $\ll 1$.

Next is the **contrast comparison function** $c(x, y)$ between two image signals x and y is defined by

$$c(x, y) = \frac{2\sigma_x\sigma y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}$$

where C_2 is given by

$$C_2 = (K_2 L)^2$$

Similar to K_1, K_2 is also a small constant $\ll 1$.
The **structure comparison function**

$$s(x, y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma y + C_3}$$

where σ_{xy} is defined as

$$\sigma_{xy} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y)$$

Finally, the SSIM score is defined as

$$S(x, y) = [(x, y)]^\alpha \cdot [c(x, y)]^\beta \cdot [s(x, y)]^\gamma$$

where $\alpha > 0, \beta > 0, \gamma > 0$ denotes the relative importance of each of the metrics. An important point is that the three components are relatively independent. To simplify the expression, if we assume, $\alpha = \beta = \gamma = 1$ and $C_3 = C_2/2$, we can get a specific form of the SSIM index,

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

However, there's one more thing worth noticing. The authors have pointed out that for image quality assessment, it is useful to apply the SSIM index *locally* rather than *globally*. First, image statistical features are usually highly spatially non-stationary. Second, image distortions, which may or may not depend on the local image statistics, may also be space-variant. Third, at typical viewing distances, only a local area in the image can be perceived with high resolution by the human observer at a one-time instance. Finally, localized quality measurement can provide a spatially varying quality map of the image, which delivers more information about the quality

degradation of the image and may be useful in some applications.

Instead of applying the above metrics globally (i.e. all over the image at once) it's better to apply the metrics regionally (i.e. in small sections of the image and taking the mean overall). This method is often referred to as the *Mean Structural Similarity Index*.

The authors use an 11x11 circular-symmetric Gaussian Weighing function (basically, an 11x11 matrix whose values are derived from a Gaussian distribution) $\mathbf{w} = \{w_i | i = 1, 2, \ldots, N\}$ which moves pixel-by-pixel over the entire image. At each step, the local statistics and SSIM index are calculated within the local window. The estimates of local statistics $\mu_x, \sigma_x, \sigma_{xy}$ are then modified accordingly as

$$\mu_x = \sum_{i=1}^{N} w_i x_i$$

$$\sigma_x = \left(\sum_{i=1}^{N} w_i (x_i - \mu_x)^2 \right)^{\frac{1}{2}}$$

$$\sigma_{xy} = \sum_{i=1}^{N} w_i (x_i - \mu_x)(y_i - \mu_y)$$

Once computations are performed all over the image, we simply take the mean of all the local SSIM values and arrive at the global SSIM value.

$$MSSIM = \frac{1}{M} \sum_{j=1}^{M} SSIM(x_j, y_j)$$

where X and Y are the references and the distorted images, respectively; x_j and y_j are the image contents at the jth local window; and is the number of local windows of the image.

3.1.2 Photometric reconstruction error

Neural networks have become central in several areas of computer vision and image processing. The impact of the loss layer of neural networks, however, has not received much attention in the context of image processing: the default and virtually only choice is l_2. In the paper "Loss Functions for Image Restoration with Neural Networks", by Hang Zhao et al [5], the authors have compared the performance of several losses and propose a novel, differentiable error function.

Firstly, the authors have noticed that the loss layer of a neural network can effectively drive the learning of the network to produce the desired output quality. Despite many recent works have focused on tuning the architecture of the network. The image quality assessment that was widely used was the mean square error l_2.

However, the l_2 suffers from many limitations because of many assumptions implicitly made when using l_2. l_2 loss assumes that the impact of noise is independent of the local characteristics of the image. l_2 loss also works under the assumption of white Gaussian noise, which is not valid in general.

Secondly, the authors focus on the image restoration tasks and study the effect of different metrics on the network's loss layer. They compare l_2 against three error metrics: l_1, SSIM, and MS-SSIM, and propose a motivation use of each metric. The tasks the authors evaluated were: image super-resolution, JPEG artifacts removal, and joint denoising plus demosaicking.

For a function error $\mathcal{E}$, the loss for a patch P can be written as

$$\mathcal{L}^{\mathcal{E}}(P) = \frac{1}{N}\sum_{p\in P}\mathcal{E}(p)$$

where N is the number pixel p in the patch

The l_1 error

For the task of joint denoising and demosaicking, l_1 provides a significant improvement when compared with the l_2 because of the splotchy artifacts in flat regions produced by l_2. (see Fig. 1 in [5])
The formula of the l_1 loss:

$$\mathcal{L}^{l_1}(P) = \frac{1}{N}\sum_{p\in P}|x(p) - y(p)|$$

Moreover, the main difference between l_1 and l_2 loss is the convergence properties. l_2 gets stuck more easily in a local minimum, while for l_1 it may be easier to reach the better minimum.

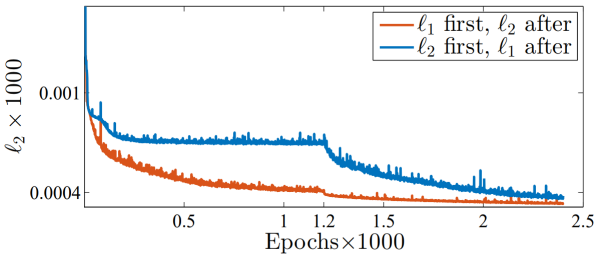

Figure 3. l_2 loss on the testing set for the two networks that switch loss functions during training.

The network trained with l_1 only (before epoch 1200 in the plot) achieves a better l_2 loss than the one trained with l_2. Another result that is worth mentioning here is that the network trained with l_1 achieves a lower l_2 loss than the network trained with l_2. However, neither of these two networks outperforms Mix (the proposed lost function I will explain later) on any of the perceptual metrics they used, confirming that the advantage of the proposed loss goes beyond convergence considerations (see table 1 in [5]).

The SSIM and MSSIM index

The authors omitted the dependence of means and standard deviations on pixel p. Means and standard deviations are computed with a Gaussian filter with standard deviation σ_G, G_{σ_G}.
The loss functions for SSIM can be written as:

$$\mathcal{L}^{\text{SSIM}}(P) = 1 - \text{SSIM}(\tilde{p})$$

where $\tilde{p}$ is the center pixel of patch P, and the SSIM

$$\text{SSIM}(p) = l(p) \cdot cs(p)$$

The choice of σ_G has an impact on the quality of the processed results of a network that is trained with SSIM. For smaller values of σ_G the network loses the ability to preserve the local structure and the splotchy artifacts are reintroduced in flat regions. For large values of σ_G, we observe that the network tends to preserve noise in the proximity of edges.

Rather than fine-tuning the σ_G, the authors propose to use the multi-scale version of SSIM, MS-SSIM. Given a *dyadic pyramid of M levels* (see Fig 4) , MS-SSIM is defined as

$$\text{MS-SSIM}(p) = l_M^{\alpha}(p) \cdot \prod_{j=1}^{M} cs_j^{\beta_j}(p)$$

where l_m and cs_j are the luminance and the contrast we've already mentioned, now they are just at the scale M and j, respectively. For convenience, the author set $\alpha = \beta_j = 1$, for $j = \{1, ..., M\}$.

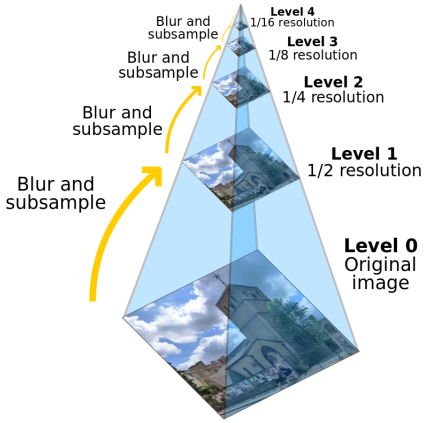

Figure 4. Visual representation of a dyadic pyramid with 5 levels

And we can compute the loss for patch P as

$$\mathcal{L}^{\text{MS-SSIM}}(P) = 1 - \text{MS-SSIM}(\tilde{p})$$

However, the computation of a pyramid of M levels is a very computationally expensive task.

After analyzing, the authors have pointed out a few observations. Both SSIM and MS-SSIM are not sensitive to a uniform bias on a flat region. This is particularly true in bright regions. MS-SSIM solves the issue of noise around edges but does not solve the problem of the change of colors in flat areas.

The Mix Loss function: MS-SSIM + l_1

MS-SSIM is not particularly sensitive to uniform biases, but it can cause changes in brightness or shifts in colors. Moreover, MS-SSIM preserves the contrast in high-frequency regions better than the other loss functions the authors experimented with.

On the other hand, l_1 preserves colors and luminance (an error is weighted equally regardless of the local structure) but does not produce quite the same contrast as MS-SSIM

To capture the best characteristics of both error functions, the author proposes a combination:

$$\mathcal{L}^{\text{Mix}} = \alpha \cdot \mathcal{L}^{MS-SSIM} + (1 - \alpha) \cdot G_{\sigma_G^M} \cdot \mathcal{L}^{l_1}$$

where the authors omitted the dependence on patch P for all loss functions, and empirically set $\alpha = 0.84$.

Because the photometric reconstruction cost of Monodepth2 was inspired by the proposed error Mix function, we will continue with the photometric reconstruction error of Monodepth2. In Monodepth2, pe is the photometric reconstruction error and it is represented by:

$$pe(I_a, I_b) = \frac{\alpha}{2}(1 - \text{SSIM}(I_a, I_b)) + (1 - \alpha)\|I_a - I_b\|_1$$

where $\alpha = 0.85$, and I_a is the reconstructed image and I_b is the input image. Monodepth2 uses a simplified SSIM with a 3×3 block filter instead of Gaussian.

3.1.3 Edge-aware smoothness loss

It is well known that the photometric reconstruction errors are non-informative in homogeneous regions of the scene. The photometric loss is solely based on the comparison between the predicted and actual pixel values. In the homogeneous or textureless regions with similar pixel values, this loss may not provide strong gradients for the model to learn meaningful depth information. Moreover, if the corrected pixel is located in a low-texture region or far from the current estimation, it would make the gradient go wrong.

To deal with these problems, a regularization term is needed. A smoothness loss acted as regularization is added to the loss function to enhance the smooth depth predictions. The idea is to penalize depth maps that exhibit abrupt or discontinuous changes, leading to more visually plausible and coherent depth predictions.

The loss function of the edge-aware smoothness in Monodepth2 was inherited from the disparity smoothness loss of Monodepth1 directly. We will show the formula of these functions in the later part, we shall discuss the idea of these functions. The authors of Monodepth recognize the correlation between the Stereo Matching task and the Monocular Depth Estimation task. Both share the common objective of understanding the 3D structure of a scene. They also face similar challenges, such as handling occlusions, dealing with textureless regions, and addressing the correspondence problem.

Therefore, the idea of the smoothness loss function in Monodepth2 is inherited from the paper "PM-Huber: PatchMatch with Huber Regularization for Stereo Matching" by Philipp Heise et al. in ICCV2013 [6]. The main contribution of this article by Philipp Heise et al. is an explicit variational smoothness model for the PatchMatch algorithm using quadratic relaxation.

PatchMatch

First, we need to understand the PatchMatch algorithm. In 2009, Barnes et al. introduced the PatchMatch algorithm [7], the purpose of this algorithm is to find similar patches between two images efficiently. Its algorithm contains three steps: random initialization, propagation, and random search.

The motivation of the PatchMatch was that when we have two images of M pixels and patches of m pixels, the naive brute-force approach has a complexity of $O(m * M^2)$ since it must compute the patch distance over m pixels for each patch position in the first image and each patch position in the second image.

There have been some improvements, like using a Tree structure to get down to $O(m * M * log(M))$ and dimensionality reduction techniques like PCA. But it was still far too slow to be used in real-time for image editing.

The algorithm begins with randomly matching each patch in A with a patch in B. If we're lucky, we may get some pairs already similar enough. Then, for each iteration, we have two steps. In the first step, we check if our neighbors can give us a better candidate patch. If it is, we just propagate this benefit, i.e., just update the patch. In the second step, we continue looking for a

better patch in our neighborhood by random search in order to get out of a local optimum. Fig 5 below is an example.
PatchMatch algorithm is based on some key ideas:

1. Dimensionality of offset space: Instead of clustering the $O(m)$ patches in patch space ($dim\ m$), we can search inside the 2D offset space.
2. Natural structure of images: Adjacent pixels are likely to have adjacent nearest-neighbors. Thus, we can perform the search in an interdependent manner.
3. The law of large numbers: Even a random offset is likely to be a good guess for many patches over a large image (M).

At first glance, drawing random guesses doesn't seem like an efficient way of solving an optimization problem. However, when we combine with the second point, everything starts to make sense. Each pixel searches randomly but also communicates with its neighbors to see if they have found a more relevant offset. Thus, if a random draw results in a good offset, it can be propagated over a whole region of the image.

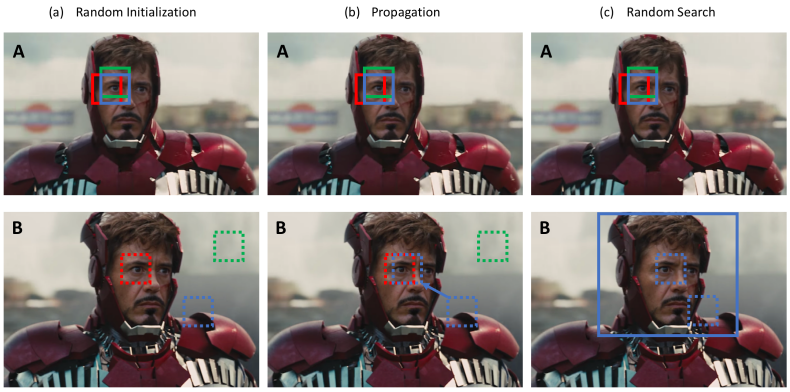

Figure 5. Example of PatchMatch Algorithm

With PatchMacth, Image Inpainting, and Texture Synthesis can now be performed in real time. Moreover, this algorithm has unlocked a new level of performance and efficiency in patch-based image processing.

PatchMatch Stereo

In 2011, Bleyer et al. introduced PatchMatch Stereo [8], which is an extension of the PatchMatch algorithm that uses the same randomized search strategy to find correspondences between patches in left and right stereo images and reconstruct the 3D scene.

However, the implementation and the optimization problem being solved in PatchMatch Stereo are completely different from the original PatchMatch algorithm. It's not simply using the PatchMatch algorithm as a dependency, but rather it's inspired by its propagation scheme to solve the optimization problem of *stereo matching*.

Stereo matching is a task that finding correspondences between points or features between two images of the same scene taken from different viewpoints. With stereo matching, we first match every pixel in the left image with its corresponding pixel in the right image so we can derive the disparity between the two images. Disparity is a term that refers to the distance between two corresponding points in the left and right image of a stereo pair. The disparity of

all the pixels can then be represented as an *intensity image* in which each pixel represents the distance values, this image is called the *disparity map*.

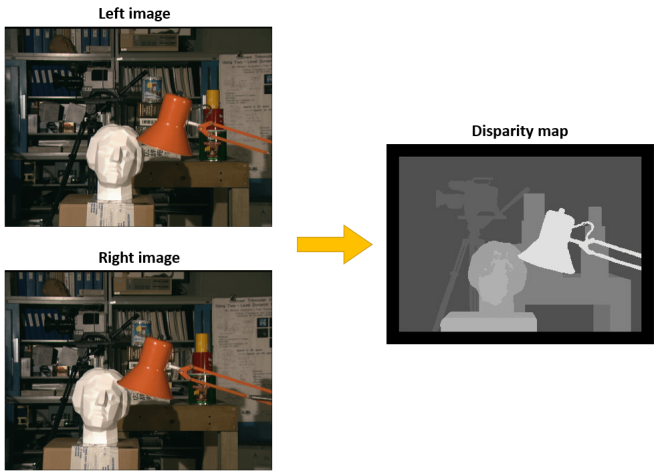

Figure 6. Disparity map derived from stereo matching

The idea of finding the correspondence pixel is that we could take a small patch around that pixel in the image then slide it across the other image and see where it matches the most. However, we don't have to scan across the entire image as the point is bound to lie on a line shown in the below figure (The Epipolar Line). This constraint is called the *The Epipolar Constraint*.

The intuition is that in the real world, the point can lie anywhere on the line joining its projection and the camera center. So we can reason that the projection of this point in the other image can lie anywhere on the projected line. This projected line is called the epipolar line. So now our search space is reduced to this line.

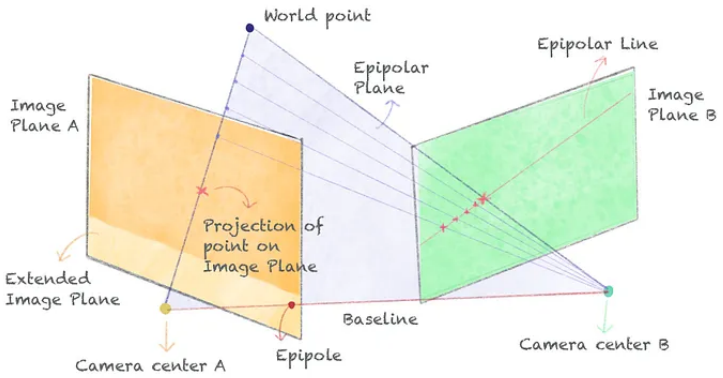

Figure 7. Epipolar Geometry

If the two image planes are parallel to each other, the epipolar lines of the two images will have the same Y-coordinate (we get rid of the math here), and the epipolar lines will be parallel in the image space. And we can calculate the disparity by formulas.

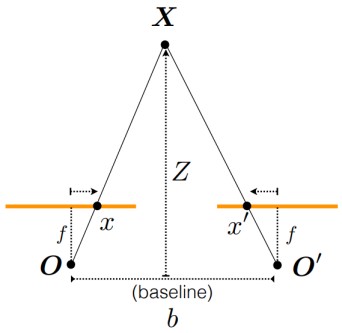

Figure 8. The two image planes are parallel

The disparity in the parallel case can be computed by

$$d = x - x' = \frac{bf}{Z}$$

The disparity is inversely proportional to the depth of the object. Specifically, the larger the disparity between the two images, the closer the object is to the cameras.

However, in the real world, it's hard to make the image planes exactly parallel. There is a technique that can make these image planes parallel which is called *Stereo Rectification*.

Stereo image rectification is the process that involves transforming a pair of stereo images using homographies so that the corresponding epipolar lines become aligned and parallel. Once the images have been rectified, stereo matching can be performed more easily and accurately since corresponding points lie on the same horizontal line.

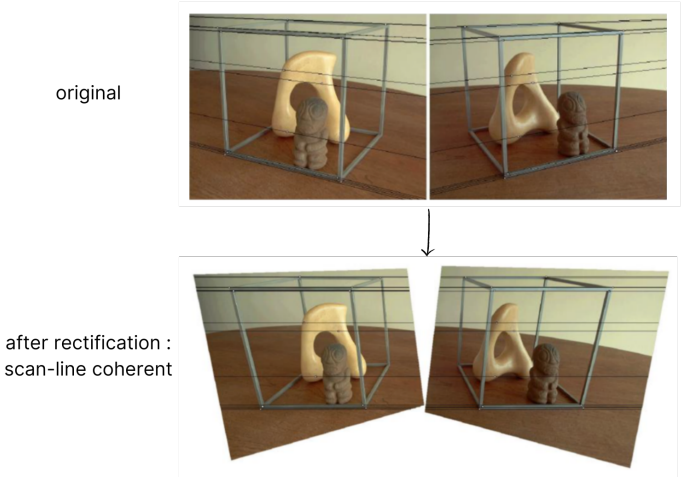

Figure 9. Stereo Rectification example

The matching process is usually done by searching a window of pixels around each feature in one image and looking for a similar feature in the other image within a corresponding disparity range. However, looking only for a patch translation imposes a strong assumption, namely that the disparity is locally constant, i.e., a fronto-parallel plane. Slanted or curved surfaces do indeed

cause changes in the shape and size of the projection of an object on the image plane. Moreover, we only perform at the pixel level when we measure with integer-valued disparities and this can cause the "staircase effect" in the depth map. In summary, we can't just roll a rectangular patch over the image. We need to warp it according to the normal of the local plane.

PatchMatch Stereo enables us to handle slanted surfaces and sub-pixel precision. The brilliant idea of the authors of PatchMatch Stereo was to mimic the propagation scheme of PatchMatch but on an array of plane hypotheses instead of mere 2D offsets.

As for PatchMatch, PatchMatch stereo can leverage the fact that neighboring pixels are likely to belong to the same local plane.

To avoid unnecessary future conversions between meters and pixels, a plane is directly represented by the coefficients a, b, and c of a 2D estimator that transforms pixel coordinates in the image (u, v) into disparity d. Each pixel has its local plane estimate, hence the index p. Since it's only a local model, (u, v) should stay relatively close to p.

$$d_p(u, v) = (a_p \;\; b_p \;\; c_p) \cdot \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

Unlike when dealing with integer disparities, we have infinite possible plane candidates. This will, however, remain computationally feasible since the PatchMatch scheme will only provide a small set of plane candidates to compare.

As for the initialization, plain random values for the a, b, and c coefficients won't correctly sample the plane space. The idea is to sample both a random disparity in the allowed range and a random unit vector for the plane normal. They can then be merged into a plane equation.

Three different types of propagation are involved:

1. Spatial propagation: use estimates of neighboring pixels. The idea behind this form of propagation is that spatial neighboring pixels are likely to have similar planes.

2. View propagation: exploit the strong coherency that exists between left and right disparity maps.

3. Temporal propagation: the plane estimate at the same pixel in the previous/next frame. (Works only on stereo video sequences)

Like in PatchMatch, the authors explore new plane candidates by randomly perturbing the disparity and the normal. This step is known as Plane Refinement.

They also include a post-processing step occlusion treatment via left/right consistency checking.

PM-Huber: PatchMatch with Huber Regularization for Stereo Matching

Finally, we can come back to the paper that inspired the smoothness loss function for Monodepth2. So far, we have covered some basic knowledge about PatchMatch and the stereo-matching problem. In this paper by Philipp Heise et al., they propose a method that integrates the PatchMatch stereo algorithm into a variational smoothing formulation using quadratic relaxation. The resulting algorithm allows the explicit regularization of the disparity and normal gradients using the estimated plane parameters.

In the context of stereo matching, there are two broad classes of algorithms for computing visual correspondence. Local algorithms estimate correspondence independently at each pixel, typically using correlation with fixed-size windows. Global algorithms find the best disparity configuration f. In general, while local algorithms are faster, global algorithms give the best

results. The difference is most striking at disparity discontinuities and in low-texture regions. Most global algorithms are based on a standard energy minimization framework.

Given two rectified stereo color images I_1, $I_2 : (\Omega \subset \mathbb{R}^2 \to \mathbb{R}^3)$, a disparity map $d : \Omega \to \mathbb{R}$ and a normal map $n : \Omega \to \{x \in \mathbb{R}^2 : |x| \leq 1\}$, the aim is to minimize an energy of the form:

$$E(d, n) = \lambda E_{data}(d, n) + E_{smooth}(d, n)$$

The data term describes the similarity between pointwise matches in the stereo pair and the smoothness term which is a regularization term used to impose spatial smoothness on the disparity-values d and the normals n. Here we will focus on the smoothness term E_{smooth}.

$$E_{smooth}(d, n) = \int_\Omega g(p)|\nabla d|_{\epsilon_d} + g(p)|\nabla d|_{\epsilon_n} \, dp$$

with $|\cdot|_\epsilon$ is the robust Huber norm

$$|x|_\epsilon = \begin{cases} \frac{|x|^1}{2\epsilon} & \text{if } |x| \leq \epsilon, \\ |x| - \frac{\epsilon}{2} & \text{otherwise} \end{cases}$$

And $g(p)$ is the introduced per-pixel weighting function

$$g(p) = e^{-\zeta |\nabla I_1|(p)|^\eta}$$

Similar to this, the authors of Monodepth1 encourage disparities to be locally smooth with an $L1$ penalty on the disparity gradients ∂d. They define the disparity smoothness loss as

$$C_{ds}^l = \frac{1}{N} \sum_{i,j} |\partial_x d_{ij}^l| e^{-\|\partial_x I_{ij}^l\|} + |\partial_y d_{ij}^l| e^{-\|\partial_y I_{ij}^l\|}$$

Derived from Monodepth1, the edge-aware smoothness loss in Monodepth2 is expressed as

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

3.2 Reprojection Loss

Because we reconstruct the image from multiple images, we have to deal with the problem that some objects are visible in the target image, by are not visible in the source images. If the network predicts the correct depth for such a pixel, the corresponding color in an occluded source image will likely not match the target image, inducing a high photometric error penalty. This effect can be reduced by masking some pixels but this method doesn't handle disocclusion.

Hence the new novel photometric loss can handle both these issues, instead of averaging the photometric error over all source images, we simply use the minimum. The improved per-pixel photometric loss is

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

3.3 Auto-masking Loss

Self-supervised monocular training often operates under the assumption of a moving camera and a static scene. However, this assumption fails when the scene has an object moving at ego velocity, which means the same velocity as our camera. This makes our model confused between the moving object and the sky for example (because both don't change the position). This problem can create 'holes' of infinite depth in the predicted test time depth maps. This

motivates the second contribution: a simple auto-masking method that filters out pixels that do not change appearance from one frame to the next in the sequence. This has the effect of letting the network ignore objects that move at the same velocity as the camera, and even ignore whole frames in monocular videos when the camera stops moving.

The authors introduce a per-pixel mask μ to the loss. They observe that pixels that remain the same between adjacent frames in the sequence often indicate a static camera, an object moving at equivalent relative translation to the camera, or a low texture region. Therefore, the authors set μ to only include the loss of pixels where the reprojection error of the warped image $I_{t'\to t}$ is lower than that of the original, unwarped source image I'_t

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

where [ ] is the Iverson bracket. In cases where the camera and another object are both moving at a similar velocity μ prevents the pixels which remain stationary in the image from contaminating the loss. Similarly, when the camera is static, the mask can filter out all pixels in the image.

3.4 The combined Loss Function

The final training loss is the combination of per-pixel smoothness and masked photometric losses.

$$L_{final} = \mu L_p + \lambda L_s$$

This loss is average over each pixel, scale, and batch.

4 Training and Evaluation

4.1 Training

4.1.1 Dataset

In this task, we utilized the KITTI dataset [9], made available by the Karlsruhe Institute of Technology and Toyota Technological Institute. This dataset offers a diverse and demanding collection of real-world images and sensor data obtained from a mobile platform. It serves as a valuable resource for a range of computer vision tasks, including object detection, segmentation, and depth estimation—tasks that are specifically addressed in this assignment.

4.1.2 Training

During the preprocessing phase, we resize the input images to ensure seamless compatibility with our model. Despite the images in the KITTI dataset already having a size of 640x192, it remains a prudent practice to resize the input to meet the specific requirements of our model.

The model underwent training on three sequences from the KITTI dataset, namely 2011_09_29_drive_0071, 2011_09_29_drive_0026, and 2011_09_26_drive_0096, which includes 1200 images in total. The training process encompassed 60 epochs, with a learning rate set at 0.0001. The learning rate experienced a tenfold reduction every ten epochs. The entire training procedure concluded in approximately 3.5 hours. Both the best-performing based on the loss and final models were saved to our drive for future utilization.

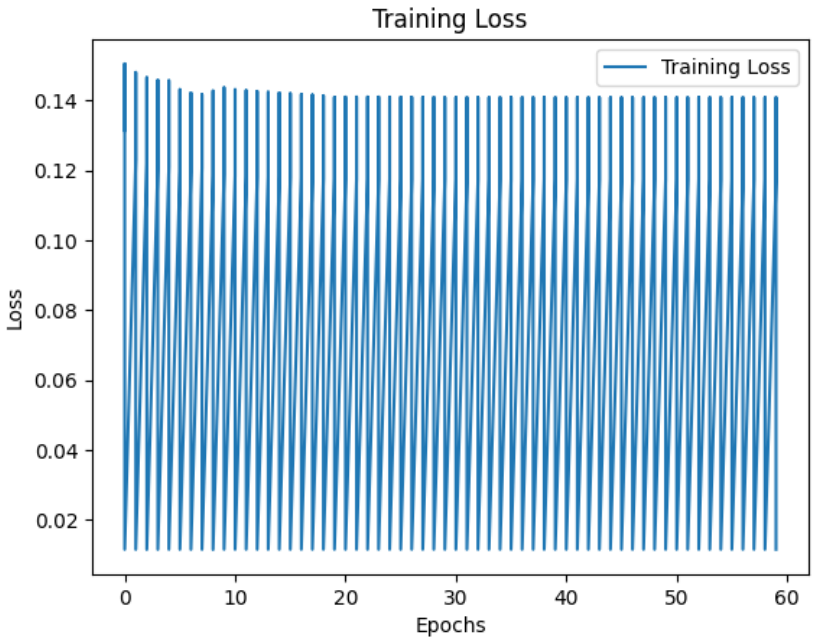

Figure 10. Loss fluctuates over epochs

If you wish to review our code, you can access it via the following link. For access to the

dataset and the trained models, please kindly visit Dataset and Model link..

4.2 Evaluation

The following image is a simple depth estimation of the input RGB image:

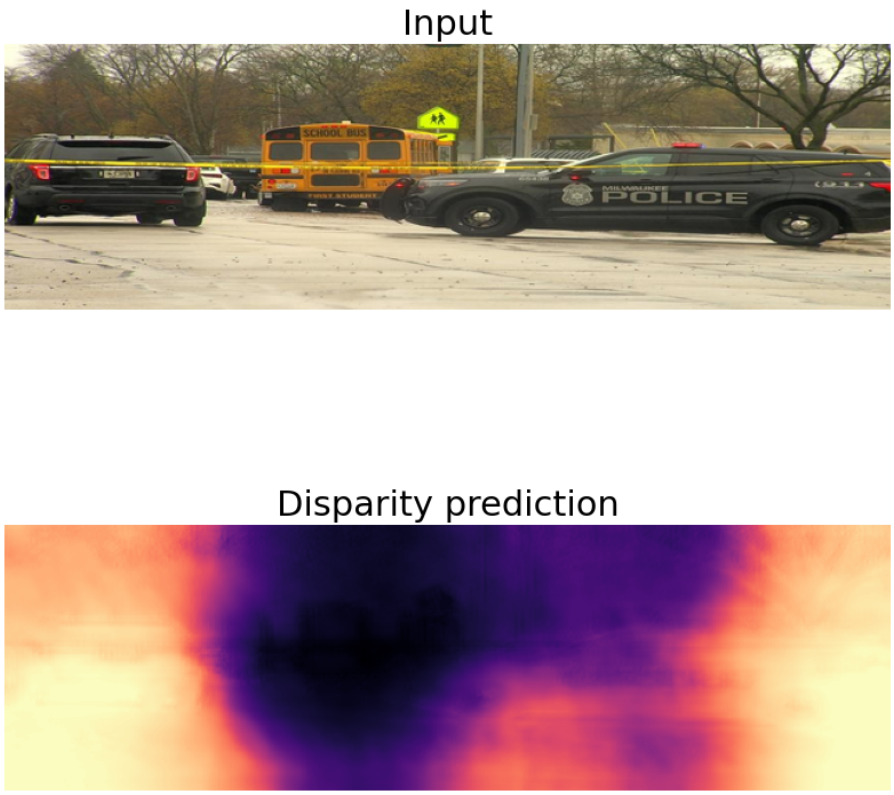

Figure 11. Depth Estimation using trained model

The generated output reveals two distinct orange regions corresponding to the positions of the two police cars; however, beyond these instances, the depth estimation produced by our model falls short. Notably, the model struggles to discern the depth of the school bus. Furthermore, it exhibits a more concerning limitation, an inability to delineate the outlines of objects. This deficiency is evident in the model's incapacity to differentiate between trees and police cars, leading to both being assigned identical depth values in the prediction.

Here are the metrics of evaluation of our model:

abs_rel	sq_rel	rmse	rmse_log	a1	a2	a3
& 25250.291	& 4065.044	& 4.531	& 4.524	& 0.069	& 0.124	& 0.165

Figure 12. Metrics of model trained on KITI

Method	Train	Abs Rel	Sq Rel	RMSE	RMSE log	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Eigen [9]	D	0.203	1.548	6.307	0.282	0.702	0.890	0.890
Liu [36]	D	0.201	1.584	6.471	0.273	0.680	0.898	0.967
Klodt [28]	D*M	0.166	1.490	5.998	-	0.778	0.919	0.966
AdaDepth [45]	D*	0.167	1.257	5.578	0.237	0.771	0.922	0.971
Kuznietsov [30]	DS	0.113	0.741	4.621	0.189	0.862	0.960	0.986
DVSO [68]	D*S	0.097	0.734	4.442	0.187	0.888	0.958	0.980
SVSM FT [39]	DS	0.094	0.626	4.252	0.177	0.891	0.965	0.984
Guo [16]	DS	0.096	0.641	4.095	0.168	0.892	0.967	0.986
DORN [10]	D	**0.072**	**0.307**	**2.727**	**0.120**	**0.932**	**0.984**	**0.994**
Zhou [76]†	M	0.183	1.595	6.709	0.270	0.734	0.902	0.959
Yang [70]	M	0.182	1.481	6.501	0.267	0.725	0.906	0.963
Mahjourian [40]	M	0.163	1.240	6.220	0.250	0.762	0.916	0.968
GeoNet [71]†	M	0.149	1.060	5.567	0.226	0.796	0.935	0.975
DDVO [62]	M	0.151	1.257	5.583	0.228	0.810	0.936	0.974
DF-Net [78]	M	0.150	1.124	5.507	0.223	0.806	0.933	0.973
LEGO [69]	M	0.162	1.352	6.276	0.252	-	-	-
Ranjan [51]	M	0.148	1.149	5.464	0.226	0.815	0.935	0.973
EPC++ [38]	M	0.141	1.029	5.350	0.216	0.816	0.941	0.976
Struct2depth '(M)' [5]	M	0.141	1.026	5.291	0.215	0.816	0.945	0.979
Monodepth2 w/o pretraining	M	0.132	1.044	5.142	0.210	0.845	0.948	0.977
Monodepth2	M	**0.115**	**0.903**	**4.863**	**0.193**	**0.877**	**0.959**	**0.981**
Monodepth2 (1024 × 320)	M	**0.115**	**0.882**	**4.701**	**0.190**	**0.879**	**0.961**	**0.982**
Garg [12]†	S	0.152	1.226	5.849	0.246	0.784	0.921	0.967
Monodepth R50 [15]†	S	0.133	1.142	5.533	0.230	0.830	0.936	0.970
StrAT [43]	S	0.128	1.019	5.403	0.227	0.827	0.935	0.971
3Net (R50) [50]	S	0.129	0.996	5.281	0.223	0.831	0.939	0.974
3Net (VGG) [50]	S	0.119	1.201	5.888	0.208	0.844	0.941	**0.978**
SuperDepth + pp [47] (1024 × 382)	S	0.112	0.875	**4.958**	**0.207**	0.852	0.947	0.977
Monodepth2 w/o pretraining	S	0.130	1.144	5.485	0.232	0.831	0.932	0.968
Monodepth2	S	**0.109**	**0.873**	4.960	0.209	**0.864**	**0.948**	0.975
Monodepth2 (1024 × 320)	S	**0.107**	**0.849**	**4.764**	**0.201**	**0.874**	**0.953**	0.977
UnDeepVO [33]	MS	0.183	1.730	6.57	0.268	-	-	-
Zhan FullNYU [73]	D*MS	0.135	1.132	5.585	0.229	0.820	0.933	0.971
EPC++ [38]	MS	0.128	0.935	5.011	0.209	0.831	0.945	**0.979**
Monodepth2 w/o pretraining	MS	0.127	1.031	5.266	0.221	0.836	0.943	0.974
Monodepth2	MS	**0.106**	**0.818**	**4.750**	**0.196**	**0.874**	**0.957**	**0.979**
Monodepth2 (1024 × 320)	MS	**0.106**	**0.806**	**4.630**	**0.193**	**0.876**	**0.958**	**0.980**

Figure 13. Metrics of models trained on KITTI using Eigen split.

In contrast to the outcomes reported in the source material, our model's evaluation metrics, particularly Absolute Relative Error (abs_rel), Squared Relative Error (sq_rel), and Logarithmic Root Mean Squared Error (rmse_log), are considerably poorer compared to the performance achieved by various methods, including the original implementation of the developers of MonoDepth2. Additionally, the percentage of pixels with predicted depth-to-ground truth depth ratios below specific thresholds (1.25, 1.25^2, 1.25^3, corresponding to threshold metrics a1, a2, a3) is much closer to zero. Regrettably, we must acknowledge that we were unsuccessful in training a functional model.

5 NYUv2 - Indoor Depth Estimation

While the KITTI dataset offers a diverse range of raw data, it mostly includes outdoor images. To address this limitation and enhance the model's adaptability to indoor scenes, we incorporated another dataset focused exclusively on indoor images—the NYUv2 dataset. However, despite using the identical network architecture and loss function, the model's performance on the NYUv2 dataset falls significantly short compared to its performance on the KITTI dataset. Further details regarding this lack of performance will be explored in the subsequent sections.

5.1 Dataset

In the field of computer vision, the NYU Depth v2 (NYUv2) dataset is a commonly used benchmark, especially for depth estimation applications. This dataset, created by New York University, offers a wide range of indoor scenes that were photographed using both RGB and depth sensors. The NYUv2 collection is made up of photos from a variety of indoor and outdoor settings as well as offices and residential spaces. Since each image has pixel-by-pixel depth annotations, it's an invaluable tool for developing and testing depth estimate algorithms. The significance of the dataset in promoting research and development in the field of indoor scene interpretation and depth perception is attributed to its precision in-depth annotations and diversity.

5.2 Training

We scaled the input photos to fit the parameters of our model, following the same procedure as with the KITTI dataset. The first 800 photographs from the split file, which included a variety of interior scenes including living rooms, kitchens, bedrooms, and home offices, were used for training. Retaining the architecture and loss functions from the prior example, we set the model's parameters to 60 epochs and a learning rate of 0.00001, with a tenfold decrease every ten epochs.

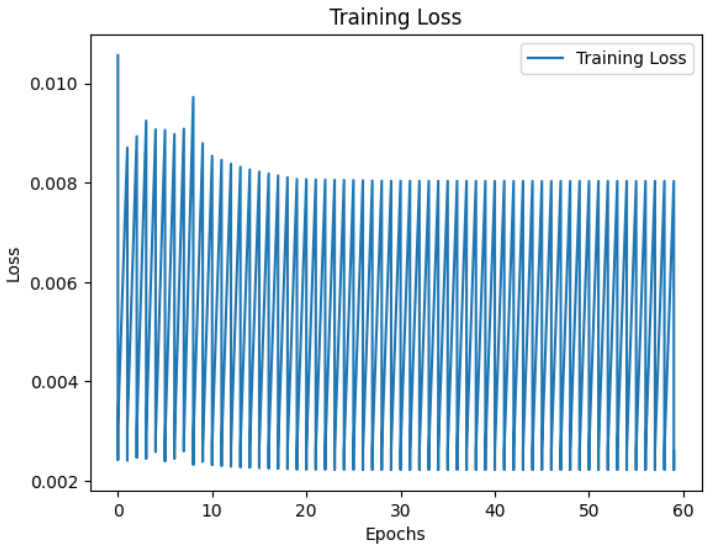

Figure 14. Training loss on NYUv2

The training loss exhibits fluctuations in each epoch, showing a decreasing trend in the maximum loss up to the 20th epoch. However, persistent fluctuations remain throughout the entire training process. Despite our efforts to identify the cause of this phenomenon, we were unable to find a suitable solution.
If you wish to review our code, you can access it via the following link. For access to the dataset and the trained models, please kindly visit Dataset and Model link.

5.3 Evaluation

The following image is a simple depth estimation of the input RGB image:

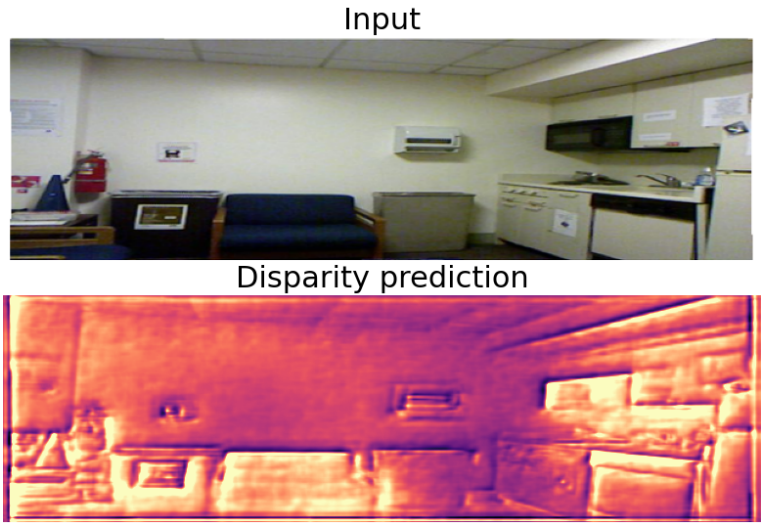

Figure 15. Depth Estimation using trained model

The output does not show encouraging results, as the graphic makes clear. The large region that is covered by orange suggests that there is a great deal of ambiguity in the model. It has trouble telling apart objects, much less determining how deep they are.

In addition, these metrics help to enhance the precision of our earlier observations:

```
abs_rel |   sq_rel |     rmse | rmse_log |       a1 |       a2 |       a3 |
  0.998  & 254.218  & 254.609  &   6.498  &   0.000  &   0.000  &   0.000  \\
```

Figure 16. Metrics of models trained using NYUv2

Every metric shows negative behavior, which is consistent with our expectations of poor model performance. The metrics that indicate a major departure from accurate depth predictions are the Squared Relative Error (sq_rel), Root Mean Squared Error (RMSE), and Logarithmic Root Mean Squared Error (rmse_log)—all of which are noticeably increased. The percentage of pixels with predicted depth-to-ground truth depth ratios below particular thresholds (1.25, 1.25^2, 1.25^3, respectively) represented by the threshold metrics (a1, a2, a3) are all registered as zero. This result clearly shows that the model is completely unable to perform its depth estimation task.

After discussions, we came up with a few reasons that might be able to explain this miserable situation:

- **Implementation Challenges:** Although monodepth2, one of the best depth estimation models, serves as an inspiration for our implementation, it is important to acknowledge that there can be undiscovered flaws in our code.
- **Limited Training Resources:** The model was trained using Google Colab's free tier, which placed limits on the amount of time that could be used. With more comprehensive training resources that enable training on the whole NYUv2 dataset, better outcomes could be possible.
- **Hyperparameter Tuning:** Carefully adjusting hyperparameters is frequently necessary to achieve optimal model performance. However, due to time and resource restrictions, the chosen hyperparameters might not be the most optimal ones.
- **Environmental Discrepancies:** The KITTI and NYUv2 datasets represent distinct environments—outdoor and indoor, respectively. Because of this point, specialized approaches are needed to handle the unique problems that each dataset presents.

We've learned a lot in our attempt to re-implement the Monodepthv2 depth estimation model. In the future, we hope we will have a chance to investigate this task more thoroughly with access to a larger dataset and more powerful computational resources, which will enable us to deepen our knowledge and produce better outcomes.s.

References

[1] Qiumei Zheng, Tao Yu, and Fenghua Wang. "DCU-NET: Self-supervised monocular depth estimation based on densely connected U-shaped convolutional neural networks". In: *Computers & Graphics* 111 (2023), pp. 145–154. ISSN: 0097-8493. DOI: https://doi.org/10.1016/j.cag.2023.01.016. URL: https://www.sciencedirect.com/science/article/pii/S0097849323000237.

[2] Tinghui Zhou et al. "Unsupervised Learning of Depth and Ego-Motion from Video". In: *CoRR* abs/1704.07813 (2017). arXiv: 1704.07813. URL: http://arxiv.org/abs/1704.07813.

[3] Zhou Wang et al. "Image quality assessment: from error visibility to structural similarity". In: *TIP* (2004).

[4] D. A. Silverstein and J. E. Farrell. "The relationship between image fidelity and image quality". In: *IEEE* (1996), pp. 881–884.

[5] Hang Zhao et al. "Loss functions for image restoration with neural networks". In: *Transactions on Computational Imaging* (2017).

[6] P. Heise et al. "Pm-huber: Patchmatch with huber regularization for stereo matching". In: *ICCV* (2013).

[7] C. Barnes et al. "PatchMatch: a randomized correspondence algorithm for structural image editing". In: *ACM Transactions on Graphics (TOG)* (2009).

[8] M. Bleyer, C. Rhemann, and C. Rother. "PatchMatch Stereo - Stereo Matching with Slanted Support Windows". In: *BMVC* (July 2011).

[9] Andreas Geiger et al. "Vision meets Robotics: The KITTI Dataset". In: *International Journal of Robotics Research (IJRR)* (2013).