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

PROGRAMMING INTERGRATION PROJECT - CO3101

Assignment

Digging Into Self-Supervised Monocular Depth Estimation

Supervisor: Nguyen Tien Thinh
Class: CC01
Student(s): Nguyen Xuan Thanh - 2152285
Vo Tran Minh Hieu - 2152560
Ma Minh Hoang - 2152081
Duong Trong Phuc - 2152237

HO CHI MINH CITY, OCTOBER 2023

Abstract

The purpose of this report is to review the paper "Digging Into Self-Supervised Monocular Depth Estimation", which proposes a set of improvements for self-supervised learning of monocular depth estimation. The paper shows that a simple model with a minimum reprojection loss, a full-resolution multi-scale sampling method, and an auto-masking loss can achieve state-of-the-art results on the KITTI benchmark. The reports summarizes our understanding on image processing, neural network and deep learning; the methods and results of the paper, its strengths, limitations, and implications for future research. Last but not least, we discuss how to reimplement the model using the PyTorch framework.

Contents

1 Introduction

1.1 Definition

Depth estimation is the task of estimating a dense depth map given one or several RGB images. The color of each pixel in the depth map corresponds to its depth, i.e. the Euclidean distance between the camera and the corresponding location in 3D world. For monocular version of the problem, the goal can be formulated as given a single image I, learn a function f that predict the per-pixel scene depth $\hat{d} = f(I)$.

1.2 Applications

Understanding the appearance of a scene from a single image is a fundamental problem in computer perception. There are many applications in 3D reconstruction, augmented reality, autonomous driving, and robotics.

1.3 Visual Cues for Monocular Depth Estimation

Human excels at monocular depth estimation because our visual system is superior in terms of generalization and exploitation of pictorial cues. One of them is occlusion which happens when one object partially covers another farther away. The next cue is called perspective which implies parallel lines appear to meet in distance. Related to perspective are size and texture gradient as the same object can have different sizes and surface densities corresponding to its absolute position to the camera. Furthermore, objects get blurry and bluish as they move away from the camera. This makes up the atmospheric cue. Other than that, human also use patterns of light and shadows on the surface when perceiving depth. Finally, height cue describes the inverse proportion of object's height and depth.
The seven cues lead to an important conclusion: human has developed a rich, structural understanding of the world through past visual experience that included moving and observing a vast number of scenes. We then apply this prior knowledge when perceiving a new one, probably unseen before. Therefore, it's natural to mimic the human perception by training a model that observes sequences of monocular images and aims to explain its observations through predictions of camera motion and scene structure.

2 Learning-based method

2.1 Supervised Depth Estimation

A significant proportion of learning based methods treat depth prediction as a supervised regression problem, in which the models learn to predict depth maps directly from color input images and their corresponding ground truth depths. Consequently, they face the problem of collecting accurate, large and varied training datasets for supervised learning. Even sophisticated hardware, such as laser scanners, can give imprecise results because of movement and reflections.

2.2 Self-supervised Approaches

As an alternative, training depth estimation models can be viewed as an image reconstruction problem. Consider the task of novel view synthesis: given an image of the scene, synthesize a new image seen from a different camera pose. This can be done provided that we have a per-pixel depth of that image, together with camera pose and visibility in the nearby view. If

we reverse this process, that is, given a set of images as input, either in the form of stereo pairs or monocular sequences, if we can learn a function that is able to predict the appearance of a target image from the viewpoint of another image, then we have learned vital 3D features of the captured scene that aid the regressed approximation. This is the intuition of self-supervised training, whose requirements are simply sequences of images without manual labeling or camera motion information.

Figure 1: Picture of a man holding up the Leaning Tower of Pisa.

Here is an instance of ambiguous scenarios resulted from the projection of different depths onto the image plane. If we were able to view this scene from a completely different angle, this illusion immediately disappears and correct layout of the world can be trivially derived.

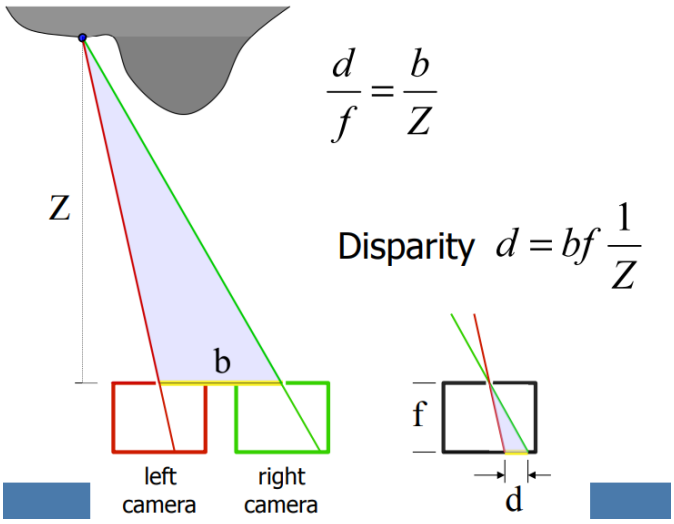
2.2.1 Self-supervised Stereo Training

2.2.1.a Binocular Disparity

A stereo-pair image contains two views of a scene side by side. One of the views is intended for the left eye and the other for the right eye. Because a given feature in the scene will have different position relative to each camera, if we superpose the two images, there will be a disparity in the location of that feature.

2.2.1.b Relationship between disparity and depth

Consider the case when both cameras are looking forward with their optical axes parallel. Let b, the baseline, be the amount of horizontal displacement between two cameras. Let f denotes the focal length and Z denotes the project depth along the z-axis of a point P in the scene to these cameras. Then,



In words, the horizontal disparity in this particular setup is equal to the ratio of the product of the baseline b and the focal length f to the depth. Given that we know b, we can compute d to recover Z. In nature, while the formula becomes much more complex, disparity remains to be the scaled inverse depth.

2.2.1.c The correspondence problem

In order to measure disparity, we need to determine for a point in the left image, the point in the right image that results from the projection of the same scene point. Therefore, the majority of self-supervision using stereo pairs computes the similarity between each pixel in one image and every other pixels in the other. In other word, they approximate the pixel disparities between the pair to use as additional source of training for depth estimation. Although there are many efficient and innovative published works that tackle this aspect of deep learning, it is not the topic to be reported here but a reference to be compared to.

2.2.2 Self-supervised Monocular Training

A less constrained form of learning uses monocular videos, in which consecutive temporal frames are turned into training signal. However, this introduces an issue nonexistent in stereo training, where we are usually provided with the baseline of two synchronized cameras. Here, in addition to predicting depth, we must estimate the ego-motion of camera between frames. The purpose of such task is that estimation of relative pose between two consecutive frames addresses the ambiguity of extremely large number of possible depth maps which can correctly reconstruct the novel view. The detail of how this can be achieved will be discussed in the subsequent section.

2.3 Camera Pose Estimation with Deep Learning

Camera pose estimation is the task of determining the translation and orientation of camera from an image. This is one of the fundamental problems in computer vision and is key to applications including augmented reality, autonomous navigation and robotics. Motivated by the success of deep Convolutional Neural Networks (dCNNs) in image classification, object detection and semantic segmentation, PoseNet was proposed by Kendall et al. as the starting-point architecture for regressing absolute pose with deep learning. The architecture had inspired a surge of deep

learning-based methods for pose estimation, including the one discussed in this report. Unlike PoseNet, we aren't concerned with the location of camera centre in world coordinate, but the relative transformation of the camera between two consecutive views of monocular videos.

3 Multiple View Geometry

3.1 Camera Model

3.1.1 The pinhole camera model

In order to create a two-dimensional image, image sensors gather light scattered from objects in the scene. Each photon arriving at the sensor produces energy according to its wavelength. The output of the sensor is the sum of all energy due to photons observed in some time window, i.e. the intensity of light.
To see a focused image, all the photons from approximately the same spot in the scene must arrive at approximately the same point in the image plane. The pinhole camera, consisting of a tiny pinhole opening O at the front and an image plane at the back, is able to ensure nearby photons in the scene passing through O will gather.

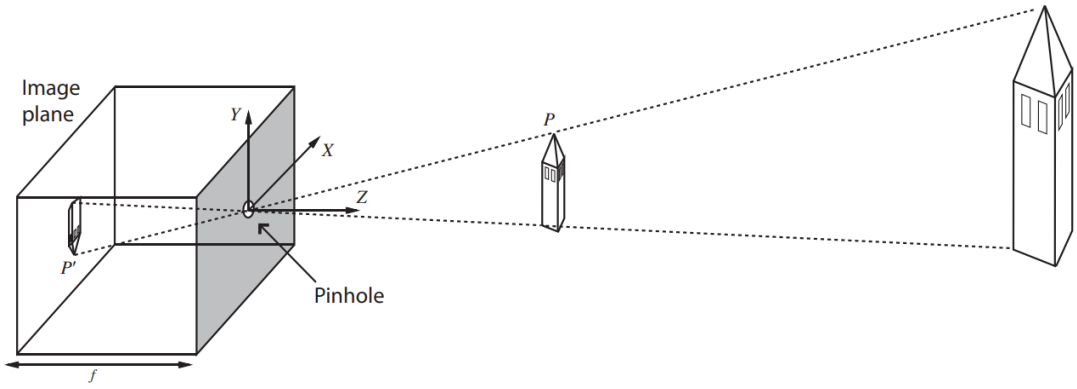


Figure 2: The pinhole camera illustrated as a large box. Image is projected upside down and large, distant objects look the same as smaller, nearby objects.

The image-formation process can be described under a three-dimensional coordinate system with the origin at the pinhole. Consider a point P in the scene, whose coordinates are (X, Y, Z) and projection in the image plane is P' with coordinates (x, y). Let f be the distance from the pinhole to the image plane. By similar triangles, we can derive

$$x = \frac{-fX}{Z}$$
$$y = \frac{-fY}{Z}$$

The Z reminds us of the size cue, and the minus signs indicates that the image is inverted compared with the scene. Without loss of generality, we will omit the minus signs.

3.1.2 Central projection using homogeneous coordinates

If the world and image points are represented by homogeneous vectors - regular vectors having an extra coordinate, the linear mapping between them using the notations $\bar{x} = \begin{pmatrix} \frac{fX}{Z} \\ \frac{fY}{Z} \\ 1 \end{pmatrix}$, $\bar{X} = \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$ can be represented as

$$\bar{x} = \frac{1}{Z} \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \frac{1}{Z} \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \bar{X}$$

If the principal point is not at the origin of the camera coordinate frame - the 2D coordinate on the image plane, then the image point P' will be at $(\frac{fX}{Z} + p_x, \frac{fY}{Z} + p_y)$ where (p_x, p_y) are the coordinates of the principal point. Write

$$K = \begin{pmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{pmatrix}$$

Then

$$Z\bar{x} = \begin{bmatrix} K & 0 \end{bmatrix} \bar{X}$$

K is called the camera intrinsic matrix as it includes the internal parameters of a camera and projects 3D points in the world onto the 2D image plane.
In general, location of P will be expressed in terms of a different Euclidean coordinate frame, which we call the world coordinate frame. The two coordinate frames are related via a rotation and a translation. If $\vec{n} = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$ is the vector representing the coordinates of P in the world coordinate frame, and $\vec{n}'$ represents it in the camera coordinate frame, then $\vec{n}' = R(\vec{n} - C)$ where C is the coordinates of the camera centre O in the world coordinate frame, and R is a 3×3 rotation matrix representing the orientation of the camera coordinate frame. In homogeneous form, the mapping can be rewriten as

$$Z\bar{x} = \begin{bmatrix} K & 0 \end{bmatrix} \begin{bmatrix} R & -RC \\ 0 & 1 \end{bmatrix} \bar{X} = KR \begin{bmatrix} I & -C \end{bmatrix} \bar{X}$$

As a result, the 3×4 homogeneous camera projection matrix is $M = \frac{1}{Z} KR \begin{bmatrix} I & -C \end{bmatrix}$.

3.2 Depth-image Based Rendering

Depth-image based rendering is the process of synthesizing "virtual" views of a scene from still or moving color images and associated per-pixel depth information. It consists of two steps. At first, the target image points are reprojected into the 3D world, using corresponding depth data. Then, these 3D space points are projected into the image plane of the second camera located at the desire viewing position. We will investigate several cases where two images are taken from different views as a result of changing camera parameters, moving or rotating its centre and a combination of all three operations. For simplicity, the world coordinate frame will be chosen to coincide with the first camera centre. This results in the first camera projection matrix is $M = \frac{1}{Z} K \begin{bmatrix} I & 0 \end{bmatrix}$ and the second one is $M = \frac{1}{Z} K' R \begin{bmatrix} I & -C \end{bmatrix}$.

3.2.1 Stationary camera centre

A 3D object and the camera centre define a set of rays, which intersect with a plane to form an image. Suppose this set of rays is intersected by two plane as follow

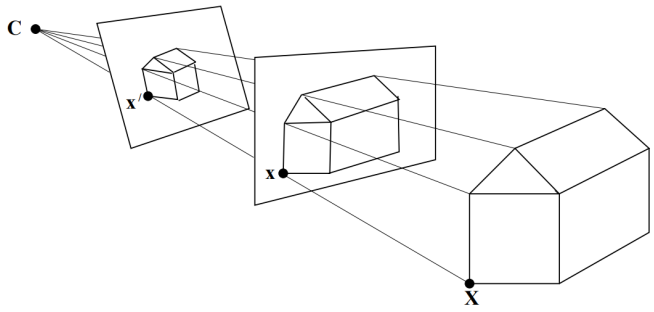


Figure 3: A ray between a 3-space point X and the camera centre C pierces the planes at x and x'. All image points are related by a planar homography: $x' = Hx$

The cameras have the same centre, so $M' = K'RK^{-1}M$. As a result, the corresponding image points are related by a planar homography $H = K'RK^{-1}$.

3.2.2 Translations of the camera

Suppose the motion of the cameras is a pure translation with no rotation and no change in the internal parameters. Then the inhomogeneous coordinates of the 3D point is

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = ZK^{-1} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = ZK^{-1}\bar{x}$$

Therefore,

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \frac{1}{Z} K \begin{bmatrix} I & t \end{bmatrix} \begin{pmatrix} ZK^{-1}\bar{x} \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} + \frac{Kt}{Z}$$

3.2.3 Camera under general motion

Given two arbitrary "virtual" cameras, we may rotate the one used for the first image so that it is aligned with the other. A further correction may be applied to account for any difference in the intrinsic matrices. Since these two corrections don't move the first camera centre, they result in a projective transformation H of the first image.

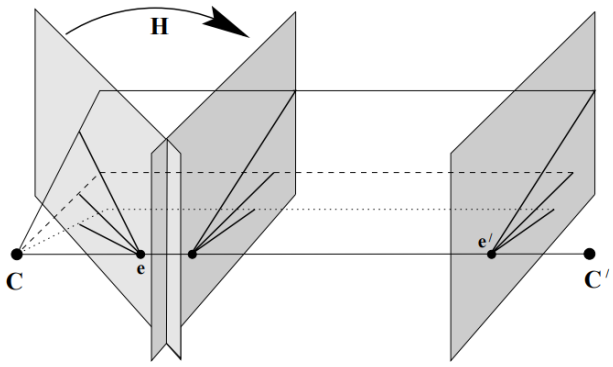


Figure 4: The first camera (on the left) may be rotated and corrected to form a pure translational motion.

Combine the two individual cases above, we acquire the mapping

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = K'RK^{-1} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} + \frac{K't}{Z}$$

The first term of the summation depends on the image position and covers the camera rotation and change of internal parameters; while the second term depends on the point's depth Z and takes account of camera translation.

4 Proposed Learning Method

4.1 Architecture Overview

The creators of monodepth2 feed to the system unlabeled image sequences capturing a scene from different viewpoints. Two models are coupled during training, one for single-view depth prediction, and one for multi-view camera pose estimation, with a loss based on reprojection error of nearby images to the target using the computed depth and pose.
Let denote $I_1, \ldots, I_N$ as a monocular image sequence with one of the frames I_t being the target view and the rest being the source views I_s $(1 \leq s \leq N, s \neq t)$. While the single-view depth network predicts a dense depth map D_t, the multi-view pose network estimates a 4x4 camera transformation matrix $T_{t \to s}$ that represents the relative pose for each source view I_s with respect to the target image I_t. Then, based on depth image-based rendering, the predicted depth D_t and matrix $T_{t \to s}$ will be combined with the prior intrinsic K to calculate for each source view I_s, its reprojection $\hat{I}_s$ to the target coordinate frame.

4.2 Differentiable Image Warping

The calculation of the reprojection $\hat{I}_s$ can be decomposed into 3 stages. First and foremost, the authors initialize a meshgrid of pixel coordinates of the target image and multiply it with the inverse intrinsic matrix K^{-1}, and later the predicted depth D_t in order to back-project I_t to a 3D point cloud. After that, the points are reprojected onto the source view by multiplying each with the product of K and the transformation matrix $T_{t \to s}$. The result of the two stages is that for

each homogeneous coordinates p_t of a pixel in I_t, there is a corresponding projected coordinates p_s onto the source view I_s. The final stage is to obtain the value of $\hat{I}_s$ at each location p_t.

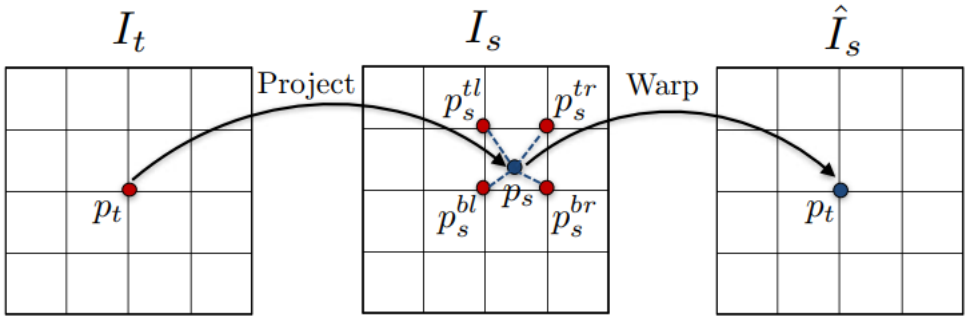


Figure 5: Illustration of the differentiable image warping process. For each point p_t in the target view, we first project it onto the source view based on the predicted depth and camera pose, then use bilinear interpolation to calculate the value of $\hat{I}_s$.

4.3 Bilinear Interpolation

When pixels in the target view are moved around to match its location in the source view, it can results in continuous values for their coordinates, as illustrated below.

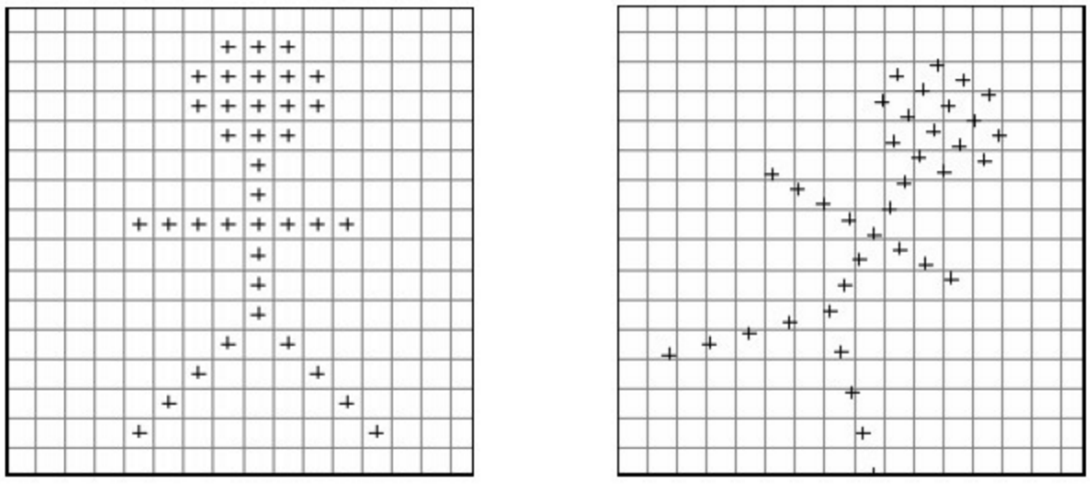


Figure 6: The rotation places some points at locations that are not centered in the squares. This means that they would not have a corresponding pixel coordinate in the image.

Bilinear interpolation can be briefly described as a resampling method that uses the distance weighted average of the four nearest pixel values in order to find the appropriate color intensity values of that pixel.

$$\hat{I}_s(p_t) = I_s(p_s) = \sum_{i \in \{t,b\}, j \in \{l,r\}} w^{ij} I_s(p_s^{ij})$$

where w^{ij} is linearly proportional to the distance between p_s and p_s^{ij}, and $\sum_{i,j} w^{ij} = 1$.

4.4 Loss Functions

4.4.1 Reconstruction Loss

The training aims to minimize the photometric reprojection error, which compares the target image I_t and its reconstruction $\hat{I}_s$ and can be formulated as

$$L_p = \sum_s pe(I_t, \hat{I}_s) = \sum_s \sum_p |I_t(p) - \hat{I}_s(p)|$$

The authors use L1 and SSIM for photometric error function pe

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

4.4.2 Structural similarity index measure (SSIM)

SSIM is a perceptual metric that quantifies image quality degradation caused by processing such as data compression or by losses in data transmission. Specifically, it measures the similarity between two images. The SSIM index is first calculated locally, using a square window which moves pixel-by-pixel over the entire image. Let x and y denote the image regions inside the window at each iteration. The formula is based on three relatively independent comparison measurements: luminance (l), contrast (c) and structure (s). The calculation of each includes the Gaussian-weighted pixel sample mean, variance, and covariance within the windows.

$$SSIM(x, y) = l(x, y)^\alpha c(x, y)^\beta s(x, y)^\gamma$$

Finally, a single overall measure is obtained

$$SSIM(X, Y) = \frac{1}{M} \sum_{i=1}^{M} SSIM(x_j, y_j)$$

where X and Y are the two images, x_j and y_j are the image contents at the j^{th} local window and M is the number of windows as a function of image dimensions.
Structural dissimilarity (DSSIM) may be derived from SSIM as

$$DSSIM(X, Y) = \frac{1 - SSIM(X, Y)}{2}$$

This makes up the left term in our photometric error function, with chosen weight $\alpha = 0.85$.

4.4.3 Per-Pixel Minimum Reprojection Loss

The issues with averaging the photometric error over all source images are pixels that are visible in the target image, but not in some of the source images. Even though the network predicts the correct depth for such a pixel, the corresponding color in an occluded source image will likely not match the target, inducing a high photometric error penalty. The proposed solution is to use minimum

$$L_p = \min_s pe(I_t, \hat{I}_s)$$

4.4.4 Disparity Smoothness Loss

Disparity smoothness is another metric to be considered because it encourages the predicted disparities to maintain piecewise smoothness and eliminates discontinuities wherever possible. The key observation is that since the disparities and the pixel intensity of an image are two bivariate functions, their gradients at each image point are 2D vectors with the components given by the derivatives in the horizontal and vertical directions. The loss simply penalizes disparity gradients using the original image gradients, where the weights are high at edges/boundaries, and low over smooth surfaces.

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

4.5 Training Procedure

For each target image I_t, two frames temporally adjacent to it are chosen as source frames, i.e. $I_s \in \{I_{t-1}, I_{t+1}\}$. The target image is input to the depth network to produce a disparity map. Then, I_t and the previous frame I_{t-1} are input to the pose network to give the first pose $T_{t\to t-1}$. The same process is repeated for the target frame and the succeeding one to find the second ego motion $T_{t\to t+1}$. Then, the reprojections $\hat{I}_{t-1}$ and $\hat{I}_{t-1}$ are calculated as described above. Self-supervision presents as we compare the reprojected target images with the real one using photometric loss as shown and optimize these using a minimization which finds the best per-pixel match between reconstructions.

5 Transfer Learning

In practice, convolutional neural networks are rarely trained from scratch because this requires a dataset of sufficient size, a lot of computation power and days of waiting. Instead, they are usually pretrained on a very large dataset, ImageNet for instance. The network is then used for the task of interest, typically in 3 major transfer learning scenarios:

- The last fully-connected layer, which outputs class scores in classification task, is removed. This leaves the rest of the network as a fixed feature extractor for the new dataset.
- Fine-tune the weights of a few layers by continuing the backpropagation on the new dataset. This makes the feature representations in the base model more relevant to the desire task.
- The pretrained models stays intact and is used for the same task as it is designed but with a narrower scope.

6 U-Net Architecture

6.1 Image Segmentation in Deep Learning

Segmentation is the process of breaking an image into regions of pixels with similar visual properties such as brightness, color, and texture. Unlike image classification, which try to classify all pixels into one class, image segmentation attempts to label each pixel. Therefore, it is useful when the image contains multiple objects with equal importance.

Figure 7: An image containing both cars and people.

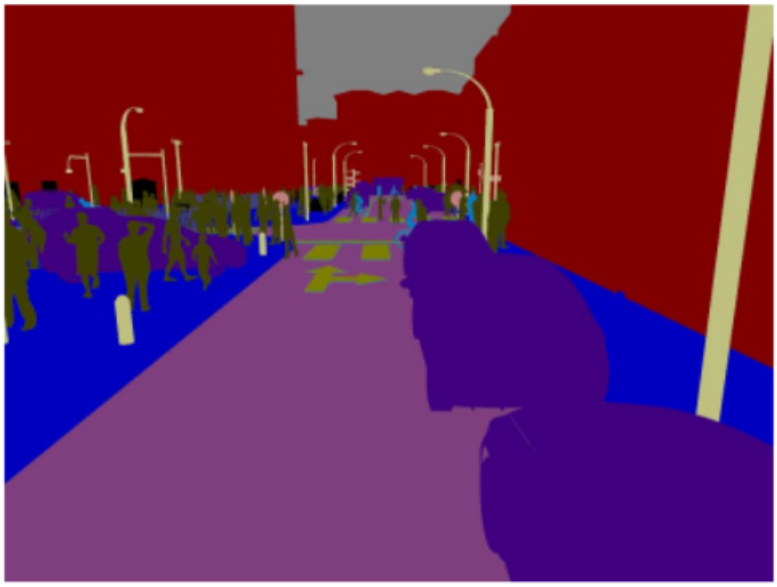


Figure 8: Corresponding output from an image segmentation algorithm. While people are coded olive, cars and buildings are given blue and red respectively.

There are two approaches to segmentation. One consecutively detects object in the scene and applies a color coded mask around that object. The other classifies the objects belonging to the same class with a single label (color). The U-Net mainly aims at the latter approach, called semantic segmentation.

6.2 Intention

U-Net was introduced in the paper "U-Net: Convolutional Networks for Biomedical Image Segmentation". As the title suggest, it is intended to be a semantic segmentation model for medical imaging. This is shown in the paper as different instances of the HeLa cells are indicated by different colors. What's stand out is that the architecture overcomes the large dataset requirement through data augmentation to use the available annotated samples more efficiently. However, we will not discuss this part in the report to focus entirely on the architecture, which enable us to represent both deep abstract features as well as local information.

6.3 Architecture details

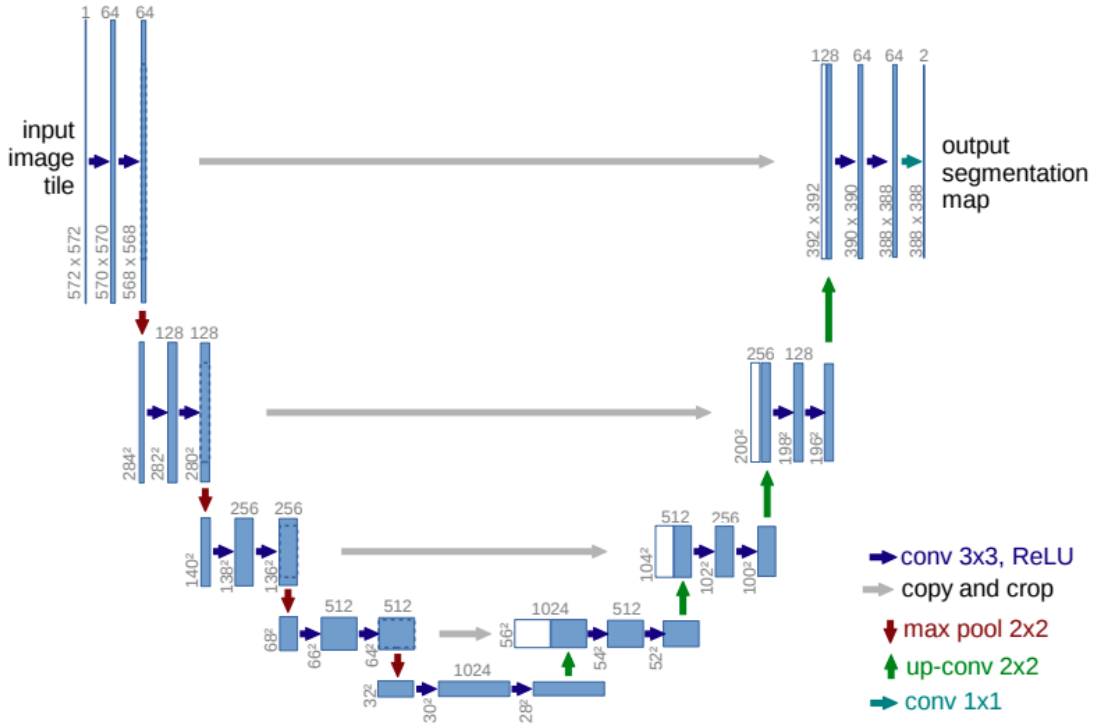


Figure 9: The original U-Net architecture. Each blue box represents a multi-channel feature map, with the number of channels on top and size at the lower left edge. White boxes are cropped and concatenated feature maps. The arrows denote different operations.

The original U-Net consists of a contracting path (left side) and an expansive path (right side). The contracting path works as a typical convolutional network because it repeatedly performs two 3x3 unpadded convolutions followed by a rectified linear unit (ReLU) and a 2x2 max pooling operation that reduces the size of the feature maps. On the other hand, the expansive path uses upsampling operators to propagate context information to higher resolution but halves the number of feature channels. The grey arrows between two paths indicate skip connections whose sole purpose is to combine features from the contracting path with the upsampled output. Finally, a convolution layer assembles component feature vectors to form class scores.

Depth estimation network of monodepth2 bases its foundation on U-Net architecture, but with two separate components called encoder and decoder working in place of the contracting and expansive paths respectively. However, the principles of an U-shaped, symmetric data pipeline and supporting skip connections remain unchanged.

7 Deep Residual Learning

7.1 Problems with Deeper Networks

The success of deep learning in image-related tasks has led to researchers building deeper convolutional neural networks with more layers to seek breakthroughs in complex problems and datasets. However, in 2015, evidences revealed that stacking more layers is not the method of learning better networks. Here is the list of detrimental behaviors of deep network with hundreds of millions of parameters.

- The notorious problem of vanishing/exploding gradients appears when there are too many layers and hinders the convergence of stochastic gradient descent with backpropagation. Fortunately, this problem has been addressed by including intermediate normalization layers.
- More parameters means higher probability for the network to overfit the training data. Dropout and other regularization methods help to reduce this phenomenon.
- When deeper networks start to converge, accuracy gets saturated and then starts to degrade rapidly.
- Deep neural networks are difficult and costly to optimize.

To address the degradation and optimization problem, residual neural networks or ResNets for short have been extensively used as the backbone for image processing component of many projects.

7.2 Residual Learning

Consider a few stacked layers of the network. Let x denotes the input to the first and $H(x)$ denotes the underlying mapping to be fit. Instead of training these layers to approximate $H(x)$, residual learning explicitly let them approximate a residual function $F(x) = H(x) - x$, then recast the mapping into $F(x) + x$. This works under the hypothesis that it is easier to optimize the residual mapping than to optimize the original mapping.

7.3 Identity Mapping by Shortcuts

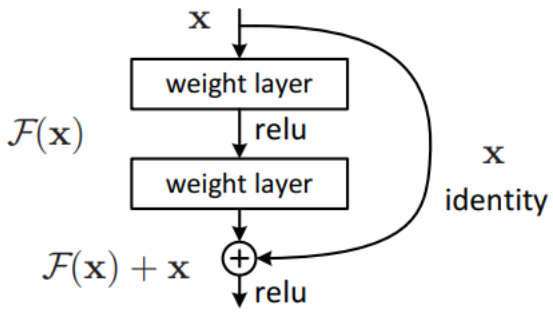


Figure 10: Residual learning building block.

A building block is formulated as

$$y = F(x, W_i) + x$$

In addition to the input vector x and output vector y, F represents the residual mapping to be learned. For example, the above figure has $F = W_2\sigma(W_1x)$ in which σ denotes ReLU and the biases are omitted. The operation $F + x$ is performed by a shortcut connection and element-wise addition. A perk of the shortcut is that it adds neither extra parameter nor computation complexity.
If the dimensions of x and F aren't equal, in which convolution operations reduce the dimensions of feature maps, we can perform a linear projection using a set of weights W_s integrated in the shortcut connections.

$$y = F(x, \{W_i\}) + W_sx$$

7.4 Network Architectures

The original paper mentions 5 ResNet models. They are ResNet18, ResNet34, ResNet50, ResNet101, and ResNet152. Though they share similarities in the organization of building blocks, we will focus on the detail implementation of ResNet18 as it is primarily used by the author of monodepth2.

layer name	output size	18-layer
conv1	112×112	7×7, 64, stride 2
conv2_x	56×56	3×3 max pool, stride 2 $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$
	1×1	
FLOPs		1.8×10^9

Figure 11: The ResNet18 architecture.

The building blocks of ResNet18 are named conv1, conv2_x, etc. Here, the additional character _x denotes the number of time the same block is repeated. It can be seen that the only change occurs across conv2_x to conv5_x is in the number of input and output channels. Therefore, the Pytorch implementation of ResNet generalizes these blocks as one class called Basic Block.
Each Basic Block contains two convolutional layers, both are followed by batch normalization if not specified. In between the two layers, there is a ReLU activation. Each convolutional layer mostly contains a number of 3x3 padded filters such that for the same output feature map size, it uses stride of 1 and preserves the amount of filters. However, when we downsample the output with stride of 2, the number of filters doubles.
Another component of Basic Block is the shortcut connection, which copies (identity maps) the input to a local variable before the forward pass so that it is later added with the layers output. An important thing to note is that when downsampling happens (at the first convolutional layer of conv3_1, conv4_1 and conv5_1), the shortcut apply 1x1 convolution to match the dimensions. All ResNet architectures share the first convolutional layer, which has 64 7x7 filters of stride 2, and the following max pooling layer. Furthermore, the networks end with a global average pooling layer and a 1000-way fully-connected layer with softmax activation. In total, ResNet18 has around 11 million trainable parameters, a modest number compared to other deep neural networks on the field such as DispNet or its brother ResNet50.

8 Depth Estimation Network

As previously mentioned, the depth estimation model is based on the general U-Net architecture, with ResNet18 as the encoder. As a result, the encoder is modified not to provide 1000 class scores but to record the output of each group's last block, where groups are named conv1, conv2, etc. In other words, 5 sets of feature vectors will be the return value of the encoder, and also the input given to depth decoder.

Depth Decoder						
layer	**k**	**s**	**chns**	**res**	**input**	**activation**
upconv5	3	1	256	32	econv5	ELU [7]
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	iconv3	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

Figure 12: The depth decoder architecture. **k** denotes the kernel size, **s** the stride, **chns** the number of output channels. **res** is the downscaling factor relative to the input image and **input** corresponds to the layer's input where ↑ stands for 2x nearest-neighbor interpolation.

Due to the symmetry characteristic of U-Net, the decoder layers are also split into 5 groups to reverse the contracting effect and to utilise skip connections. Each group consists of 2 convolutional layers whose activation functions are ELUs (Exponential Linear Unit). In constract to ReLUs, for input $x < 0$, ELUs return negative value as $\alpha(e^x - 1)$ with $\alpha > 0$. This allows them to push mean unit activations closer to zero and fasten learning. Also, instead of the traditional zero padding, monodepth2 implements reflection padding in the decoder.
The first layer of each group halves the number of channels, so it is followed by an 2x umsampling operation that simply performs nearest-neighbor interpolation. After that, the second one concatenates the immediate result with extracted feature from encoder's block at the same level, then convolves and activates ELU. This layer retains the number of output channels, though. Except the lowest group of layers (the one that has number 5 assigned to it), each group uses an additional layer that compresses all channels to 1 and outputs sigmoid values σ. These are disparity predictions at four different scales (disp4 to disp1), in which the spatial resolution doubles at each of the subsequent scales. If needed, they will be convert to depth $D = \frac{1}{a\sigma+b}$, where

a and b are calculated to constrain D between 0.1 and 100 units.

9 Pose Estimation Network

The input to the pose estimation network is the target view I_t together with either its preceding or succeeding frame $I_{t'}$. Therefore, the ResNet18 encoder is modified to accept a pair of color images (or six color channels). As a result, convolutional weights in the first layer are of shape 6×64×3×3, instead of the default of 3×64×3×3. When pretrained weights are loaded for the encoder, the first filter tensors are duplicated to allow for a six-channel input image. Nevertheless, all weights in this new expanded filter are divided by 2 so that the output is in the same numerical range as the original ResNet.

Pose Decoder						
layer	**k**	**s**	**chns**	**res**	**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	-

Figure 13: The pose decoder architecture.

Only the last feature vector from the multi-view ResNet encoder is consumed by the pose decoder. As shown in the table, each of the leading three convolutional layers has 256 output channels and is followed by ReLU activation. The last one reduces the number of channels to 12, calculates the mean of each to produce a pair of relative pose $T_{t\to t'}$ and $T_{t'\to t}$. We extract the first of these, i.e. $T_{t\to t'}$, which is simply a combination of 6 real numbers, 3 will be turned into the translation matrix T and the rest, representing the axis-angle, will become the rotation matrix R. The product of R and T is a 4x4 matrix that transforms the camera view of the target frame I_t to the one of the source frame $I_{t'}$.

10 Re-implementation on KITTI dataset

We evaluate the pretrained model by using The KITTI Vision Benchmark Suite data set for Depth Prediction Evaluation. In particular, we use the manually selected validation and test data sets. Please visit the site https://github.com/bmV3Qmll/monodepth2Replicate for code reviewing and in-depth code comparison. The Google Colab notebook is available at https://colab.research.google.com/drive/1DXYgWVrgpvofyAptGTJpfHiCSOesrd2g#scrollTo=Mhy9tFOOMvdW

10.1 Model selection

The size of all images and their corresponding ground truth depth map is 1216x352. As a result, we chose the 1024x320 monocular model as a best fit. Because the model is not yet available the first time we evaluated, we need to download it using the utility provided by the author.

```
import utils
utils.download_model_if_doesnt_exist("mono_1024x320")
```

10.2 Data loader

The data loader in the original code is curtailed to precisely fit the raw KITTI data, which is not what we wanted to test. Therefore, we introduce another class of KITTI data loader named KITTIBenchmarkDataset that fit the benchmark testing data set. Viewers can inspect the added data loader at the end of the kitti_dataset.py file. Here, we give the explanation of how the code is derived. There are three parameters that are directly attached to the testing data. They are the data path, the folder containing model weights and the split file. The first option is merely where we unzip the dataset concatenated with the specific folder of the images. The second option is where the given script download the model to, which is fixed at "models/mono_1024x320". Finally, we chose the benchmark test files at the corresponding splits sub-folder.
Before the evaluation, we also need to prepare the ground truth depth maps. As a result, we add options to export depth maps for this KITTI data set as well as the following NYUv2 data set in the export_gt_depth.py file. Following the author, we save the exported depth in the same folder of data split files.

10.3 Evaluation

The evaluation begins by determining whether the cuda is available for running the code in NVIDIA GPU, if not we set the device to CPU. Then, the weights for the depth encoder and decoder are loaded. One important task is that we need to use the right data loader described in the above subsection. After that, images are iterated for depth prediction, which in turn be compared with the exported ground truth depth maps for computing 7 different evaluation metrics.

11 Fitting the model to NYUv2 dataset

The main feature of the NYU-Depth V2 data set is 1449 densely labeled pairs of aligned RGB and depth images. Using these data, we aimed to train and evaluate a monocular model for 640x480 frames. The Google Colab notebook is available at https://colab.research.google.com/drive/15Kh9Pa5a7Yx0vMm6XAfXf7mcNNHJfaKS#scrollTo=HVVtUA4XdafE

11.1 Pre-processing

The labeled data set is provided as a Matlab .mat file with two variables we want to focus on:

- **images** - 480x640x3x1449 matrix of RGB images
- **depths** - 480x640x1449 matrix of in-painted depth maps. The values of the depth elements are in meters. The depth maps have been modified from projected ones as missing values have been filled in using the colorization scheme in http://www.cs.huji.ac.il/~yweiss/Colorization/.

Since we can't access the data directly from .mat file, we extract them into .jpg files, with grayscale for the depth maps.

```
for i = 1:1449
    imwrite(images(:,:,:,i), sprintf('./NYUv2/images/img_%d.jpg', i));
    imwrite(mat2gray(depths(:,:,i)), sprintf('./NYUv2/depths/gt_%d.jpg', i));
end
```

11.2 Prepare split files

We split the data into training, validating and testing files with the ratio of 0.64 : 0.16 : 0.2 by writing a python script that use split function from scikit-learn module. These files contain shuffle indexes and are copied to the splits folder of monodepth2.

```
import numpy as np
from sklearn.model_selection import train_test_split

n = np.arange(1, 1449)
train, test = train_test_split(n, test_size=0.2, random_state=31)
train, val = train_test_split(train, test_size=0.2, random_state=31)

with open('./NYUv2/splits/train_files.txt', 'w') as f:
    for i in train:
        f.write(f'images/ {i} 0\n')

with open('./NYUv2/splits/val_files.txt', 'w') as f:
    for i in val:
        f.write(f'images/ {i} 0\n')

with open('./NYUv2/splits/test_files.txt', 'w') as f:
    for i in test:
        f.write(f'images/ {i} 0\n')
```

11.3 Writing a new data loader

The data loader for NYUv2 is presented in a separate file named nyuv2_dataset.py

11.3.1 Intrinsic matrix

The NYU-Depth V2 data set's toolbox includes the "camera_params.m" file containing the camera parameters for the Kinect used to capture the data. Therefore, we acquire from it the focal lengths and principal point misalignment and normalize them by image width and height to construct the intrinsic matrix.

```
% Calibrated using the RGBDemo Calibration tool:
%   http://labs.manctl.com/rgbdemo/
%

% The maximum depth used, in meters.
maxDepth = 10;

% RGB Intrinsic Parameters
fx_rgb = 5.1885790117450188e+02;
fy_rgb = 5.1946961112127485e+02;
cx_rgb = 3.2558244941119034e+02;
cy_rgb = 2.5373616633400465e+02;
```

Listing 1: First few lines of the camera_params.m file

11.3.2 Getting image and depth map

The NYUv2 data loader follows the principles of the predefined KITTI data loader. We only need to change how the loader uses the command line options and split files to get the desire path to data.

11.3.3 Preprocessing

In each iteration, there is a 50% chance of performing horizontal flips, and color augmentations that set random brightness, contrast, saturation, and hue jiter with respective ranges of ± 0.2, ± 0.2, ± 0.2, and ± 0.1 to three consecutive images fed to the network.

11.4 Training the model

We train the model from scratch as the width and height ratio of image data aren't close to the provided model. The training procedure can be briefly summarised as

1. **Trainer initialization**. In this step, the code selects which device to use (GPU or CPU) and initialize depth and pose estimation networks from scratch as well as selecting the appropriate data loader.

2. **Training with Adam optimizer**. The default setting is to run 20 epochs with a batch size of 12 at a learning rate of 10^{-4} for the first 15 epochs then 10^{-5} for the remainder.

3. **Running a single epoch**. For each frame (together with its preceding and following frames), it's fed to the depth estimation network for the initial prediction, which is strengthened by the pose network. The outputs are then used to generate image depth maps for computing losses versus the ground truth depth. Using these losses, the optimizer is able to perform stochastic gradient descent in order to fit the training parameters.

4. **Saving the model**. For each epoch i, the model is saved to ' /tmp/{model_name}/models/weights_i', in which 'model_name' is user-defined.

11.5 Evaluating the model

The process is identical to what have been done in the KITTI data set, except that we have to change the data loader and specify the epoch 19 weights of the previously trained model.

References

[1] Clément Godard, Oisin Mac Aodha, Michael Firman, and Gabriel Brostow
Digging Into Self-Supervised Monocular Depth Estimation

[2] Alican Mertan, Damien Jade Duff, and Gozde Unal
Single Image Depth Estimation: An Overview

[3] Stuart J. Russell and Peter Norvig
Artificial Intelligence: A Modern Approach, Third edition

[4] Richard Hartley and Andrew Zisserman
Multiple View Geometry in Computer Vision, Second Edition

[5] Olaf Ronneberger, Philipp Fischer, and Thomas Brox
U-Net: Convolutional Networks for Biomedical Image Segmentation

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun
Deep Residual Learning for Image Recognition

[7] Clement Godard, Oisin Mac Aodha, and Gabriel J. Brostow
Unsupervised Monocular Depth Estimation with Left-Right Consistency

[8] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G. Lowe
Unsupervised Learning of Depth and Ego-Motion from Video