ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Computer Vision: Assignment 4

November 18, 2021

Department of Computer Science, ETH Zürich

Chapter 1

Introduction

1.1 Environment Setup

For setting up the Python environment, we will use miniconda. For the best and easiest experience, we strongly recommend using a Linux distribution (such as Ubuntu). However, miniconda is also compatible with Windows and Mac.

Run the following commands from the root of the code directory to install and activate the environment.

```
conda env create -f env.yaml

conda activate cv-lab4

pip install plyfile open3d
```

1.2 Hand in

The deadline for this assignment is Friday, December 3, 23:59. Write a short report answering the questions mentioned in the tasks. Upload it together with your code (for multi-view stereo, also upload the trained model `model_000003.ckpt`) and images in a single zip file to moodle.

Throughout the assignment, do not modify the function interfaces or the already written complete code. We have added extra comments to the base code to guide you.

In case you run into any issues, please check the moodle forum and open a new topic if needed.

Chapter 2

Model Fitting

In this assignment, you will learn how to use RANSAC (RANdom SAmple Consensus) for robust model fitting. We will work on the simple case of 2D line estimation.

2.1 Line Fitting

With a ground truth linear model $y = kx + b$, we generate a point set `(x_noisy, y_noisy)` with the linear model and add noise to it. Besides, we also add outliers to the point set.

2.1.1 Least-squares Solution (3 pts.)

Fill in function `least_square` in `line_fitting.py`. The function returns the least-squares soultion, `k_ls` and `b_ls`, given x and y. (*hint*: you can use `numpy.linalg.lstsq`).

2.1.2 RANSAC (7 pts.)

Fill in function `ransac` in `line_fitting.py`. The function (loops for `iter=300` iterations) works as follows:

1. randomly choose a small subset, with `num_subset` elements, from the noisy point set (*hint*: you can use `random.sample` to choose a set of random indices)
2. compute the least-squares solution for this subset
3. compute the number of inliers and the mask denotes the indices of inliers (a point is an inlier if its distance to the line smaller than `thres_dist`), fill in function `num_inlier`

4. if the number of inliers is larger than the current best result, update the parameters `k_ransac`, `b_ransac` and also the `inlier_mask`

2.1.3 Results (5 pts.)

For k, b, **write down the ground truth, estimation from least-squares and estimation from RANSAC in the report**.

Save the plot and upload it in the zip file.

Chapter 3

Multi-View Stereo

Given a collection of images with known camera parameters, multi-view stereo (MVS) describes the task of reconstructing the dense geometry of the observed scene.

In this assignment, we will try solving the multi-view stereo problem with deep learning. First, we estimate the depth maps for each view of the scene. Then we reconstruct a point cloud with all the depth maps and the filtering techniques.

The pipeline of the method is shown in Fig. 3.1. First, the features are extracted for images. Second, given uniformly distributed depth samples, we warped the source features into the reference view. Third, we compute the matching similarity between reference view and each source view. Fourth, we integrate the matching similarity from all the source views. Fifth, we do the regularization on integrated matching similarity to output probability volume. Finally, we perform depth regression and get the final depth map.

3.1 Dataset

We use a part of DTU dataset (official website of DTU) for training, validation and test. The download link is here. The DTU dataset is an indoor multi-view stereo dataset with 124 different scenes and 7 different lighting conditions.

The lists of scenes for training, validation and test are in `lists/dtu`. For the scenes used for training and validation, pair files, camera files, RGB images, ground truth depth maps and masks (denote which pixels have valid ground truth depth value) are provided. For the scenes used for test, only pair files, camera files and RGB images are provided. Pair files record the best ten source views for each reference view. We have already implemented the functions to read it in.

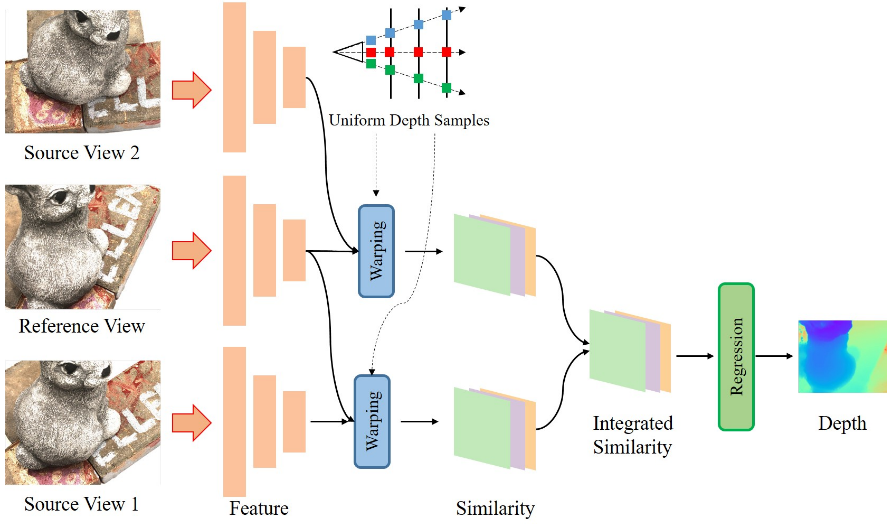

Figure 3.1: Detailed Structure of our method.

The image size is 640 × 512. To reduce computation, we will estimate the depth map at resolution 160 × 128. The intrinsics provided in camera files are already scaled to the resolution 160 × 128 (so no need to scale it).

3.1.1 RGB Images (2 pts.)

Fill in function `read_img` in `datasets/data_io.py`, which returns the RGB image. Normalize the intensity into the range $[0, 1]$.

3.1.2 Camera Parameters (3 pts.)

Fill in function `read_cam_file` in `datasets/data_io.py` to return `intrinsics`, `extrinsics`, `depth_min` and `depth_max`. The camera files record extrinsic, intrinsic and depth range with the following format:

```
extrinsic

E00 E01 E02 E03

E10 E11 E12 E13

E20 E21 E22 E23

E30 E31 E32 E33

intrinsic

K00 K01 K02

K10 K11 K12

K20 K21 K22
```

`DEPTH_MIN DEPTH_MAX`

3.2 Network

3.2.1 Feature Extraction (5 pts.)

Given N input images of size $W \times H$, we use $\mathbf{I}_0$ and $\{\mathbf{I}_i\}_{i=1}^{N-1}$ to denote reference and source images respectively. We extract features, $\mathbf{F}_i \in \mathbb{R}^{W/4 \times H/4 \times 32}$, for images with a 2D CNN. The detailed structure is shown in Table 3.1. **Fill in class `FeatureNet` in `models/module.py`.**

Layer	Output Size
ConvBnReLU, filter=3x3, stride=1	$W \times H \times 8$
ConvBnReLU, filter=3x3, stride=1	$W \times H \times 8$
ConvBnReLU, filter=5x5, stride=2	$W/2 \times H/2 \times 16$
ConvBnReLU, filter=3x3, stride=1	$W/2 \times H/2 \times 16$
ConvBnReLU, filter=3x3, stride=1	$W/2 \times H/2 \times 16$
ConvBnReLU, filter=5x5, stride=2	$W/4 \times H/4 \times 32$
ConvBnReLU, filter=3x3, stride=1	$W/4 \times H/4 \times 32$
ConvBnReLU, filter=3x3, stride=1	$W/4 \times H/4 \times 32$
Conv, filter=3x3, stride=1	$W/4 \times H/4 \times 32$

Table 3.1: Details of the feature extraction structure. ConvBnReLU is the abbreviation of 2D convolution, batch normalization and ReLU.

3.2.2 Differentiable Warping (15 pts.)

We denote intrinsic matrices with $\{K_i\}_{i=0}^{N-1}$ and relative transformations with $\{[\mathbf{R}_{0,i}|\mathbf{t}_{0,i}]\}_{i=1}^{N-1}$ (from reference view 0 to source view i). For each pixel $\mathbf{p}$ in the reference feature, we generate D samples, $\{d_j\}_{j=1}^{D}$, that are uniformly distributed in the range `[DEPTH_MIN, DEPTH_MAX]`. We will evaluate the matching similarity at these depth values and then estimate the final depth.

1. For pixel $\mathbf{p}$ in the reference feature and a depth value d_j, **write down the equation of corresponding pixel $\mathbf{p}_{i,j} := \mathbf{p}_i(d_j)$ in the report**, which is the projection of $\mathbf{p}$ in source view i with depth value as d_j (the coordinates of pixels are given in homogeneous coordinates).

2. **Implement function `warping` in `models/module.py`.** For each pixel $\mathbf{p}$ in the reference feature and given depth values $\{d_j\}_{j=1}^{D}$, the function returns the corresponding source features at the projection locations of $\mathbf{p}$.

3.2.3 Similarity Computation and Regularization (15 pts.)

1. First, we use group-wise correlation to reduce the dimension and compute the similarity between reference feature and warped source feature.

Let $\mathbf{F}_0(\mathbf{p})$, $\mathbf{F}_i(\mathbf{p}_{i,j}) \in \mathbb{R}^{32}$ be the reference and warped source features. After dividing their feature channels evenly into G groups, the g-th group similarity $\mathbf{S}_i(\mathbf{p}, j)^g \in \mathbb{R}$ is computed as:

$$\mathbf{S}_i(\mathbf{p}, j)^g = \frac{1}{C/G} \left\langle \mathbf{F}_0(\mathbf{p})^g, \mathbf{F}_i(\mathbf{p}_{i,j})^g \right\rangle, \tag{3.1}$$

where $\mathbf{F}_0(\mathbf{p})^g$ and $\mathbf{F}_i(\mathbf{p}_{i,j})^g$ are the g-th feature group, $\langle \cdot, \cdot \rangle$ is the dot product. **Implement function `group_wise_correlation` in `models/module.py`.**

2. In our method, we integrate the matching similarity from source views by simply taking the average. Then we regularize it, $\mathbf{S} \in \mathbb{R}^{W/4 \times H/4 \times D \times G}$, with a 2D U-Net. This results in $\bar{\mathbf{S}} \in \mathbb{R}^{W/4 \times H/4 \times D \times 1}$. For 2D U-Net, the detailed structure is shown in Table 3.2. **Fill in class `SimlarityRegNet` in `models/module.py`.**

Input	Layer	Output
$\mathbf{S}$	ConvReLU, filter=3x3, stride=1	C_0 $(W/4 \times H/4 \times D \times 8)$
C_0	ConvReLU, filter=3x3, stride=2	C_1 $(W/8 \times H/8 \times D \times 16)$
C_1	ConvReLU, filter=3x3, stride=2	C_2 $(W/16 \times H/16 \times D \times 32)$
C_2	ConvTranspose2d, filter=3x3, stride=2	C_3 $(W/8 \times H/8 \times D \times 16)$
$C_3 + C_1$	ConvTranspose2d, filter=3x3, stride=2	C_4 $(W/4 \times H/4 \times D \times 8)$
$C_4 + C_0$	Conv2d, filter=3x3, stride=1	$\bar{\mathbf{S}}$ $(W/4 \times H/4 \times D \times 1)$

Table 3.2: Details of the 2D U-Net for matching similarity regularization. ConvReLU is the abbreviation of 2D convolution and ReLU. ConvTranspose2d is a 2D transposed convolution. ConvTranspose2d.

Note that the 2D U-Net is only used to aggregate information in the image space (not along depth dimension). So you may need to reshape $\mathbf{S}$ before feeding it in the network.

3.2.4 Depth Regression (3 pts.)

We apply *softmax* on $\bar{\mathbf{S}}$ along the depth dimension to produce a probability volume $\mathbf{P} \in \mathbb{R}^{W/4 \times H/4 \times D}$. The regressed depth value $\mathbf{D}(\mathbf{p})$ at pixel $\mathbf{p}$ is found as the expectation:

$$\mathbf{D}(\mathbf{p}) = \sum_{j=0}^{D-1} d_j \cdot \mathbf{P}(\mathbf{p}, j), \tag{3.2}$$

where $\mathbf{P}(\mathbf{p}, j)$ is the probability for pixel $\mathbf{p}$ and depth value d_j.

Fill in function `depth_regression` in `models/module.py`.

3.2.5 Loss Function (2 pts.)

Fill in function `mvs_loss` in `models/module.py`. We compute the `l1_loss` with estimated depth and ground truth. Note that we need to use the provided mask since not all the pixels have ground truth depth.

3.2.6 Photometric Confidence

We define the photometric confidence as the probability sum of 4 nearest samples around the estimation. The reason behind is that for those falsely matched pixels, their probability distributions are scattered and cannot be concentrated to one peak, which results in a low photometric confidence. This is already implemented in `models/net.py`. We will use photometric confidence to filter out unreliable estimations (confidence below a threshold $\tau = 0.8$) during 3D reconstruction.

3.3 Training (5 pts.)

We set the number of depth values $D = 192$ and the number of input images to $N = 3$, which means that two best source views will be used to estimate the depth of reference view. We train our model with Adam ($\beta_1 = 0.9, \beta_2 = 0.999$) for 4 epochs with a learning rate of 0.001. We set batch size to 2. The training takes about 10 hours.

The part for training is already implemented in `train.py`. Run `bash train.sh` to start training. You can use tensorboard to monitor the training process that is stored in the checkpoint file. **Take a screenshot of the tensorboard to show the convergence of loss on both training dataset and validation dataset.**

3.4 Test (10 pts.)

We set the number of input images to $N = 5$ to utilize more multi-view information. The process is mainly implemented in `eval.py`.

We sequentially estimate the depth maps (`save_depth`), filter out unreliable estimations with geometric consistency filtering (`reproject_with_depth` and `check_geometric_consistency`) as well as photometric consistency filtering (`check_photometric_confidence`), and reconstruct the point clouds (`filter_depth`). Run `bash eval.sh` to do evaluation.

1. **Explain what geometric consistency filtering is doing in the report.**

2. For all the scenes, **visualize (`visualize_ply.py`) and take screenshots of the point clouds in Open3D.**

3.5 Questions (4 pts.)

Briefly answer the following questions in the report.

1. In our method, we sample depth values, $\{d_j\}_{j=1}^{D}$, that are uniformly distributed in the range `[DEPTH_MIN, DEPTH_MAX]`. We can also sample depth values that are uniformly distributed in the *inverse* range `[1/DEPTH_MAX, 1/DEPTH_MIN]`. Which do you think is more suitable for large-scale scenes?

2. In our method, we take the average while integrating the matching similarity from several source views. Do you think it is robust to some challenging situations such as occlusions?

3.6 Summary

Congratulations! With this assignment, you are already familiar with the core aspects of most learning-based stereo, multi-view stereo and optical flow methods! The core ideas behind can be briefly summarized as: sampling hypotheses, warping, computing matching similarity/cost, estimating disparity/depth/flow.