CV Final Project: Depth Estimation in Crowd Simulation

Simone Caldarella
Artificial Intelligence Systems
DISI, University of Trento
simone.caldarella@studenti.unitn.it

Federico Pedeni
Artificial Intelligence Systems
CIMeC, University of Trento
federico.pedeni@studenti.unitn.it

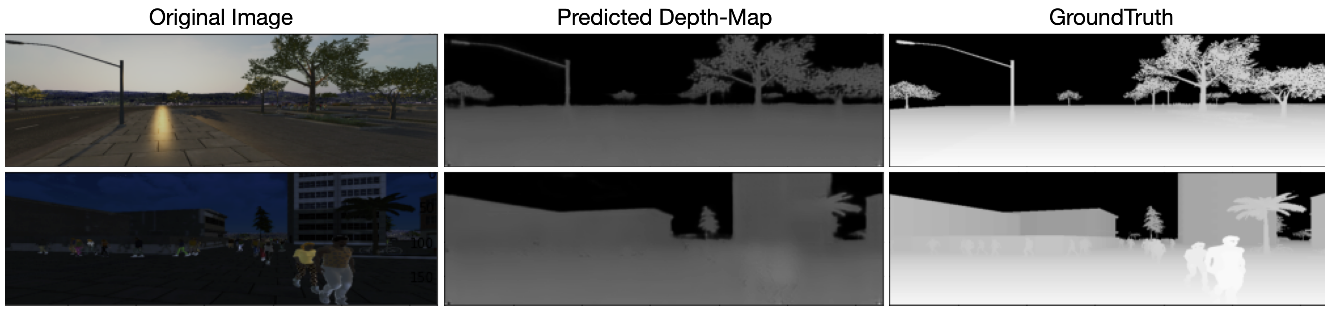

Figure 1. **Results obtained with our fine-tuning pipeline** in comparison with the depth groundtruth gathered from the simulated environment. The range shifting in the depth values could be caused by the low range of depths in the groundtruths.

Abstract

The goal of this project was to build and train a monocular depth estimation model applied in a simulated environment built with Unity. In the following sections it will be discussed the steps involved in the development of the model as long as the production of the ground-truths employed fine-tune the model in a supervised fashion.

1. Introduction

Depth estimation is a computer vision task which deals with measuring the distance of each pixel of a scene relative to a camera, producing a depthmap. The depth can be extracted from either monocular (single) or stereo (multiple views of a scene) images. While classical/older methods use multi-view geometry to find a match between stereo images with the goal of computing the disparity (and thus the distance), newer methods use deep learning models to estimate a depth-map from either a single image, by setting a regression loss in a supervised learning fashion, or employing a sequence of images, by learning to generate a novel view from it.

Notwithstanding the impressive efficiency of these newer approaches, researchers encountered the issue of data scarcity: a common issue for most of the fields deep learning. In Depth Estimation this issue becomes critical due to the necessity of data with high variability across all possible scenes in order to properly train the model in a regression fashion. Furthermore, the acquisition of depthmaps in populated environments is complicated and requires specific costly hardware and eventually it could yield to noisy data, with artifacts due to motion and illumination or to the behavior of the other actors.

For the above reasons, the cutting edge research in this field is trying to include simulated environments in the data gathering process: by using simulations in the data creation process, engineers gain uncountable advantages, ranging from the pixel-wise control of each sample to the possibility of automatic validation of the data, including the independence from costly equipment required for real data collection.

Following these considerations, in this project, we relied on a pre-trained unsupervised depth estimation model (Monodepth2 [1]), that we fine-tuned, in a supervised fashion, employing scene-depth image couples gathered from a simulated envornment in Unity [2].

2. Dataset

Our starting point was the Unity simulation of a city environment developed by the MMLab (Multimedia Signal Processing and Understanding Lab) of the University of

Trento[1]. The environment, whose scene resembles the city of Trento, is provided with a script that generates human-like avatars that simulate crowd movements. Within 8 differently located generation areas, a random number of avatars are created and start to move, randomly walking without bumping into eachother. However, even though they do move realistically, they do not get much far from their origin, an issue that we had to confront later when generating data.

The city is composed of around 20 physical buildings, several intersecting streets and many trees and bushes; it also contains a mountain-like background for higher realism and a mutable skybox for changing daytime along with smaller objects such as cars and bikes to add details to the simulation. Starting from this environment, we tried to use already existing cameras to save simultaneously rendered frames and depth maps. Cameras were adapted to resemble the point-of-view of already existing real-life depth datasets such as KITTI [3], mainly by fixing height and orientation of the cameras to resemble real cameras mounted on moving vehicles.

To extract the depthmaps, we develop and ad-hoc that renders each pixel's distance from the camera onto a hidden texture. Specifically, the distance is measured in absolute values between the far/near clipping planes of the camera (that define the field of view), then mapped in the range $[0, 1]$ and transformed into a grayscale image. In this way nearer objects appear brighter, while further ones progressively fade to black. This shader executes at a reserved point of the rendering pipeline, using a dedicated Renderer object that we added to the code of the simulator.

In order to capture images from different perspectives and locations, we automatized the acquisition process while making the cameras walk along predefined paths. We defined 7 paths composed of more than 20 nodes, each composed of straight lines connecting subsequent nodes and avoiding physical objects: we ran a simulation for each of these routes, making the camera move at constant speed and always oriented towards the next point, so that the motion always appears to be in the frontal direction; we also set the daytime to change continuously during each walk, to add stochasticity in the lighting conditions. Paths were designed to traverse more than one generation areas, so that they encountered many avatars with high probability: we consider this a key difference between our dataset and the other public ones, because in the real world it is difficult to use depth-recording-equipment inside a crowd of people.

Capturing one screenshot and its depth map per each frame, we ended up with 7422 paired samples, in different daylight conditions and from different perspectives, with many screenshots taken inside avatar crowds. Even though our method could yield more samples, we decided to limit the dataset size because further iterations would have produced images with the same structural and color features.

[1]https://github.com/mmlab-cv/Crowd-Simulator

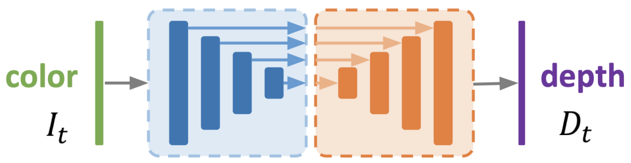

Figure 2. Monodepth2 Depth Network we employed for the fine-tuning.

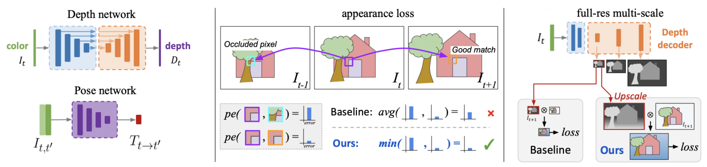

Figure 3. Overview of the full pipeline developed to train the Monodepth 2 model.

3. Method

Here, we describe the unsupervised depth estimation model that we used as pre-trained network and our fine-tuning pipeline, along with the issues we faced during the development.

3.1. Pre-Trained model: Monodepth2

As for the starting point of our fine-tuning procedure, we decided to rely on an unsupervised depth estimation pre-trained model, Monodepth2 [1], in order to leverage the advantages of the extensive training process accomplished by Godard et al., more difficult to obtain when training models from scratch.

In Monodepth 2, they try to tackle the limitation of acquiring *per-pixel groundtruth* depth training data by employing self-supervised learning. Specifically, they propose a set of improvements over standard self-supervised methods which show State-of-the-Art results (in 2019) on the KITTI benchmark [3]:

- *minimum reprojection loss* to handle occlusions;
- *full-resolution multi-scale sampling* method that reduces visual artifacts;
- *auto-masking loss* to ignore training pixels that violate camera motion assumptions.

The four different approaches applied jointly in the work, can be seen in the Fig. 3, where the components are:

1. *Depth Network*, a U-Net used to predict the depth, which employs a ResNet18 backbone;

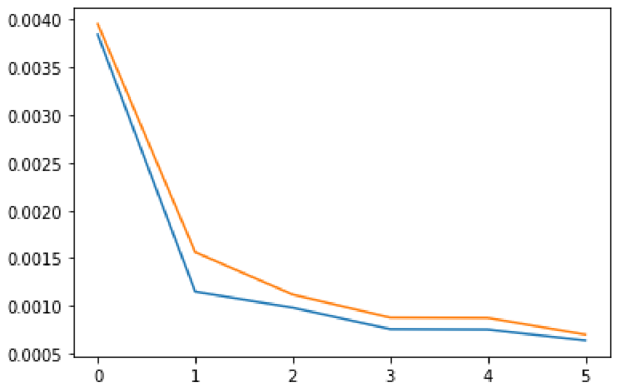

Figure 4. Orange Represents validation loss while blue represents train loss.

2. *Pose Network*, used to predict the pose between a pair of frames;

3. *Per-Pixel Minimum Repojection*, which ensures to give more penalty when correspondences between points in a sequence of images are bad, avoiding to take in account occluded pixels;

4. *Full-Resolution Multi-Scale* module, used to compute a different loss for depth generated from different intermediate layers, thus enhancing the level of details in the final results.

From this pipeline, we focused only on the already trained *Depth Network* (Fig. 2), following the assumption that the advantages given by all the components employed in the training would still had an impact to the fine-tuning starting point.

3.2. Fine-Tuning on Sintetic Dataset

As mentioned before, we imported the Depth-Network specifically trained with both image sequences and stereo images, at a fixed size of 640×192 on the KITTI dataset. Then, after having pre-processed and downsampled the data due to size constraints, we trained the model as a regression task (using MSE loss), computing the loss only with the full size image, thus without relying on all the other intermediate representation. Applying a shallow grid search on the most relevant hyper-parameters, we trained the model on 20 epochs, with a learning rate equal to $3e - 4$, using a batch size of 64.

In the image 4 can be seen an example of a run with train loss and validation loss, while an example of the results, obtained by our pipeline, can be seen in the Fig. 1. Despite the apparently good results, the closeness between validation loss trend and train loss trend could be a red flag in this field. Specifically, we suggest that, notwithstanding the different cameras employed in the sample gathering along with the camera movements introduced in the collection phase, the sintetic dataset presents a way lower entropy (and thus lower variability in the scene), if compared to real life scenes. This leads to a low variability also between the validation and the train split, which, as a consequence, makes the validation less useful to obtain a feedback on the overfitting phenomenon.

4. Conclusion

To summarize the main points of our work, we firstly managed to develop an ad-hoc script in order to extract the depth of a virtual scene, in Unity, from a selected camera. Secondly, we built an algorithm able to collect several *original image-depth map* pairs, from different moving cameras, in different lightning conditions. Thirdly, after a scraping in the literature to find a model that would suit up our needs, we created the fine-tuning script and trained the model in a supervised fashion, obtaining visually satisfactory results.

References

[1] Clément Godard, Oisin Mac Aodha, Michael Firman, and Gabriel J. Brostow. Digging into self-supervised monocular depth prediction. October 2019. 1, 2

[2] John K Haas. A history of the unity game engine. 2014. 1

[3] Jonas Uhrig, Nick Schneider, Lukas Schneider, Uwe Franke, Thomas Brox, and Andreas Geiger. Sparsity invariant cnns. In *International Conference on 3D Vision (3DV)*, 2017. 2