Depth Estimation

While there are many algorithms and techniques for predicting the depth information of an image, in this task, we will only focus on depth estimations from camera images, like Monocular Depth Estimation (MDE) and Stereo Depth Estimation (SDE), and compare their strengths and weakness with some examples.

Monocular Depth Estimation

MDE is the task of predicting the depth information of a scene using only one image. Unlike the stereo system which required two images.
The main deep learning based approaches for this task is 1) a discriminative approach where the network tries to predict depth as a supervised learning objective, and 2) a generative approach like diffusion model where depth prediction is an iterative image generation task.

There are so many recent projects for these two approaches, like Midas, Marigold, DepthAnything, DepthPro, and DepthCrafter.
After quick comparison, I chose Depth Anything V2 because it is the most popular, outperforms Midas and Marigold, and has publicly available training code. In contrast, DepthCrafter and DepthPro lack published training code, with DepthCrafter being specialized in videos.

Preferable Properties	Fine Detail	Transparent Objects	Reflections	Complex Scenes	Efficiency	Transferability
Marigold [31]	✓	✓	✓	✗	✗	✗
Depth Anything V1 [89]	✗	✗	✗	✓	✓	✓
Depth Anything V2 (Ours)	✓	✓	✓	✓	✓	✓

Table 1: Preferable properties of a powerful monocular depth estimation model.

Depth Anything V2 is based on Depth Anything V1, so, let's explain the V1 framework first.

Depth Anything V1

From an architectural perspective, we can approach this problem similarly to image segmentation. However, instead of having predefined classes with a classification objective, as in segmentation, we deal with a continuous range of depth values, making it a regression task. So, we don't need novel architecture, we can use what worked with segmentation like UNet variants or encoder/decoder architecture.
The major proposal in this paper is introducing the various amounts of the unlabeled images to the model training pipeline.

Their work utilizes both small labeled datasets (D_l) and a larger unlabeled dataset (D_u). First they train a teacher model on the D_l, and then use it to assign pseudo labels for the D_u. Finally, they train a student model on both the labeled and pseudo labeled datasets.

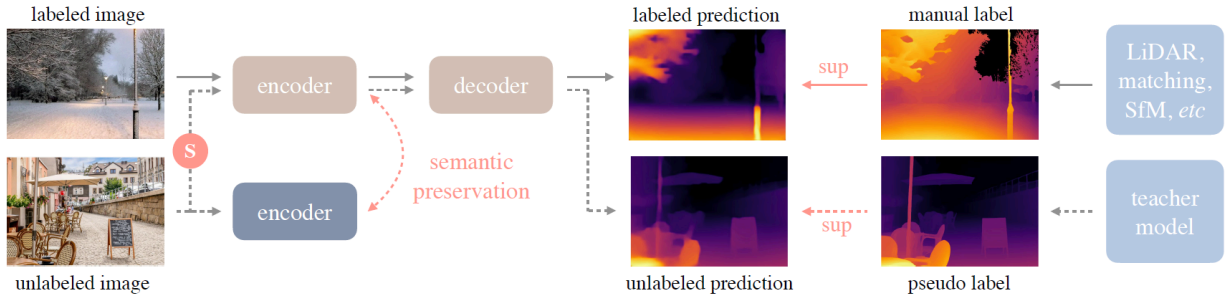


Figure 2. Our pipeline. Solid line: flow of labeled images, dotted line: unlabeled images. We especially highlight the value of large-scale unlabeled images. The S denotes adding strong perturbations (Section 3.2). To equip our depth estimation model with rich semantic priors, we enforce an auxiliary constraint between the online student model and a frozen encoder to preserve the semantic capability (Section 3.3).

And to make the student network more capable, they proposed to challenge the student with a more difficult optimization target by introducing strong perturbation on the unlabeled images and adding a feature alignment loss using the pretrained DINOv2 model. And only the student network is used during inferring time.

Perturbation

CutMix operation with 50% probability and it's applied only on the unlabeled images.

Feature alignment loss

$$\mathcal{L}_{feat} = 1 - \frac{1}{HW} \sum_{i=1}^{HW} \cos(f_i, f_i'), \qquad (9)$$

where cos(·, ·) measures the cosine similarity between two feature vectors. f is the feature extracted by the depth model S, while f′ is the feature from a frozen DINOv2 encoder.

Depth Anything V2

While Depth Anything V1 achieves state of the art results, it struggles with fine details, transparent objects, and reflections images.
So in the updated project, the authors suggest that it is the quality of the data that needs improvements, not the architecture itself.

Most real labeled datasets captured with sensors tend to be noisy, miss fine details, produce low-resolution depth maps, and face challenges with lighting conditions as well as reflective or transparent objects. Unlike real images, synthetic images generated with graphics engines are highly accurate, have high resolution outputs that capture fine details, and the depth of transparent and reflective surfaces can be easily obtained.

The problem with synthetic images though is that they are too clean compared to the real images and capture only a small distribution of the data in the real world, so, Depth Anything V2 introduce a new framework that makes use of both of them.

It's much the same pipeline and architecture as V1, but now the teacher model is trained on synthetic data instead of real data, and then it assigns pseudo labels on unlabeled data, and finally the student model is trained only on the pseudo labeled data from the teacher model.

Experiments

We will do some experiments with the left images and ignore the right for now.
When comparing images, the mean squared error (MSE), while simple to implement, is not highly indicative of perceived similarity. In some cases, two different images can have the same MSE value. Therefore, we will use the Structural Similarity Index (SSIM) as a more perceptually relevant alternative.
SSIM measures the similarity between two images, with values ranging from -1 to 1:
SSIM = 1 → Identical images.
SSIM close to 0 → Highly different images in structure and content.
SSIM < 0 → Structurally different in a way that is perceptually opposite.
Also, SSIM metric could return a diff array, which has the pixel by pixel difference that could be plotted to indicate visually which portions of the image are different. Bright areas indicate high similarities, and dark areas indicate high differences.

Disparity Estimation

Trying the model on the 466.png image

Original image

Challenges

- Textureless road.
- Reflective buildings.
- Repetitive patterns on the road and buildings.
- Sunlight illumination on the road and buildings.

Ground truth

Prediction

Calculating the SSIM score and the difference

```
score, diff = ssim(depth_pred_s, depth_true, full=True, data_range=255)
score
```

```
0.8008088277339043
```

Difference

The areas where the model fails are highlighted within the red rectangle. Which also could be noted from comparing the ground truth and the prediction visually.

Strengths

- The model accurately estimated the road depth despite the sun illuminating certain areas, making them appear brighter.
- Accurately estimated far buildings and even buildings with sun lighting on them.

Weaknesses

- Couldn’t accurately estimate thin and detailed objects on the side road that have holes
- Couldn’t accurately estimate panel, lamps, and tree edges.

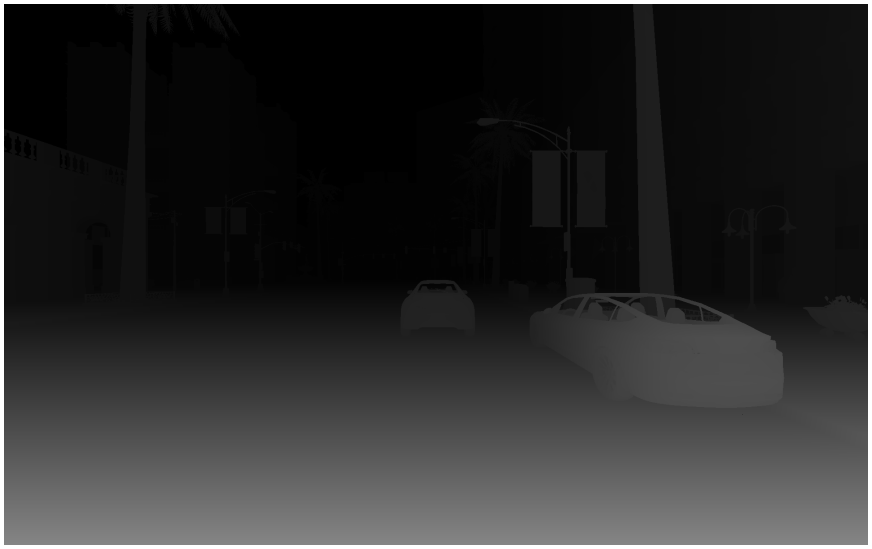
Trying the model on the 515.png image

Original image

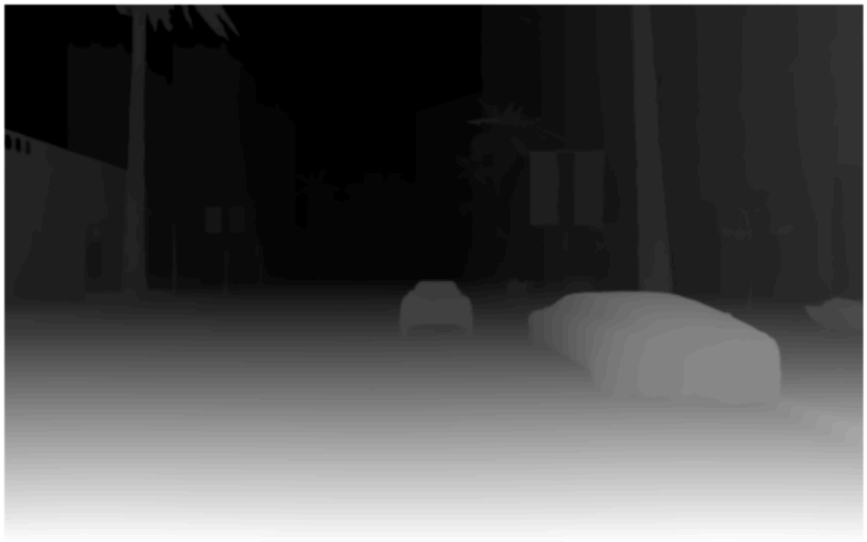
Challenges

- Same as the previous image but without the sunlight on the road.
- More details on the buildings and more sunlight on them and on the trees.
- More textureless objects like the cars.
- Transparent objects like the car's windows.

Ground truth



Prediction



Calculating the SSIM score and the difference

```
score, diff = ssim(depth_pred_s, depth_true, full=True, data_range=255)
score
```

```
0.704897427856405
```

Difference

The areas where the model fails are highlighted within the red rectangle. Which also could be noted from comparing the ground truth and the prediction visually.

Strengths

- Model accurately estimated the road and building depth.

Weaknesses

- As in the previous image, the model could not estimate the fine details and tree edges.
- Some differences at the car's windows.

Depth Estimation

Trying the model on the 466.png image

Original image

Ground truth

Prediction

Calculating the SSIM score and the difference

```
score, diff = ssim(depth_pred_s, depth_true, full=True, data_range=255)
score
```

```
0.41731586829731643
```

Difference

Strengths

- Model is good with near object like the road and provide good depth gradients

Weaknesses

- It struggles a lot with object edges, especially distant ones.

Conclusion

The model performs well overall, providing depth predictions that closely match the ground truth. It is reliable enough for applications in robotics and autonomous vehicles, where accurately detecting nearby objects is critical. However, its performance could be further improved by incorporating a better edge-aware loss function and training on more diverse data that includes small objects, fine details, and repeated patterns. This would enhance its ability to handle thin structures and complex textures, leading to more precise depth estimation in challenging scenarios.
And as for the depth, the model was initially trained for disparity estimation and later fine-tuned for depth estimation. If depth is the primary objective, training the model from scratch using depth values could lead to better performance and more accurate predictions.

Resources

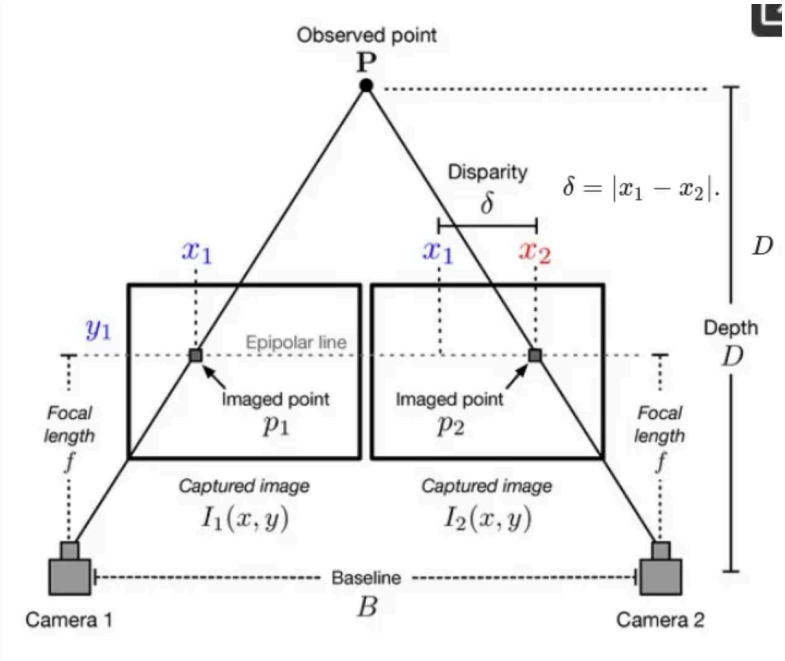
Depth Anything: Unleashing the Power of Large-Scale Unlabeled Data
https://arxiv.org/abs/2401.10891

Depth Anything V2
https://arxiv.org/abs/2406.09414

Depth Anything V2 repo
https://github.com/DepthAnything/Depth-Anything-V2

Stereo Depth Estimation

SDE works by placing two cameras at a fixed distance to capture two images simultaneously from slightly different perspectives, allowing for depth information extraction by comparing the differences between the two images. Same as how our brains use two images from our two eyes to estimate the depth information.



Given a left and a right image, we start with the left image and attempt to locate each object's corresponding position in the right image. This is achieved by scanning a small template window from the left image across all possible horizontal positions in the right image. The window in the right image that yields the best similarity score is considered the matching window.

Since the two cameras are aligned vertically and offset only in the horizontal direction, the search is limited to horizontal shifts. Once a match is found, the difference in horizontal coordinates between the corresponding points in the two images is known as disparity.

For example, the following figure shows that if an object in the left and right images with horizontal coordinates ul and yr, respectively, they will have a disparity of abs(xl - xr).

Given the disparity, the horizontal distance between the two cameras which is called the baseline (B), and the focal length which is a known camera parameter (f), we can calculate the depth (D) using the following formula:
D = B * f / disparity

Depth is inversely proportional to the disparity, and the disparity is proportional to the baseline. It could be interpreted as follows; if the distance between the two cameras are close, the difference between the two images will be small, and hence, the disparity will be small.

So, which metric could we use as a score between the windows? The metric used is determined by the method, and we have the following three method:

Local Stereo Matching: Matches image patches independently using a small window, and uses Sum of Squared Difference (SSD), Sum of Absolute Difference (SAD), or Normalized Cross-Correlation (NCC).

Global Stereo Matching: Uses optimization techniques to enforce smoothness constraints across the entire image, uses an energy function and tries to minimize it considering both matching accuracy and smoothness constraints.

Semi-Global Stereo Matching: Balances between local and global methods.

We will continue with the Semi-Global Stereo Matching (SGBM) method from OpenCV.

Disparity Estimation

Trying the model on the 466.png image

Original left and right images

Ground truth

Prediction

Calculating the SSIM score and the difference

```
score, diff = ssim(disparity_map, disp_true, full=True, data_range=(np.max(disp_true) - np.min(disp_true)))
score
```

```
0.5835988525096149
```

Difference

Depth Estimation

Trying the model on the 466.png image

Ground truth

Prediction

Calculating the SSIM score and the difference

```
score, diff = ssim(depth_map, depth_true, full=True, data_range=(np.max(depth_true) - np.min(depth_true)))
score
```

```
0.5744475300772707
```

Challenges with Stereo matching methods:

- The image must have a texture. If an image with a textureless surface, then all windows from the right image will have the same score with the left window and it wouldn't be possible to locate the object.
- The mage must have complex shapes. If an image has a repeated pattern, then again, multiple windows from the right image will have the same score.
- A lot of parameters to fine tune. For example, if the window size is small → more details but sensitive to noise, and if it has a large →smother disparity map but poor localization, it takes multiple small objects in the same window.

Resources

Stereo Vision: Depth Estimation between object and camera
https://medium.com/analytics-vidhya/distance-estimation-cf2f2fd709d8

OpenCV Python Depth Map Stereo Vision for Depth Estimation (Algorithm and Code)
https://www.youtube.com/watch?v=gffZ3S9pBUE