Monocular Depth Estimation

Davide Saia, mat.374446
Master's Degree in Robotics LM-72
Engineering department
Prof. Gabriele Costante

Abstract

This is a supervised deep learning challenge. The task is about, train a model, that has to generate a depth map from given images in input. The input pictures are created in a virtual world, generated with the software Unreal Engine. The task is evaluated by two metrics, thanks to the use of a validation set given. In this report, I give an overview of what **depth estimation** is, and how I resolve the given task, explain how I built my custom deep net and which ideas I use to obtain good performance.

1 Introduction

Depth Estimation measures the **distance** of each pixel relative to the camera. Depth is extracted from either monocular (single) or stereo (multiple views of a scene) images. Traditional methods use multi-view geometry to find the relationship between the images. Newer methods can directly estimate depth by minimizing the regression loss, or by learning to generate a novel view from a sequence. Scene depth estimation plays an important role in computer vision, which enhances the perception and understanding of real three-dimensional scenes leading to a wide range of applications such as robotic navigation, autonomous driving, and virtual reality. The models are based on CNN, and they used a better combination of them and exploited the new GPU's technology, thanks to the innovation, the training phase is faster than in the past. It is important to the great work given from the past to reuse the pre-trained models like ResNet to build a custom model to execute the task. A general depth map has a lighter patches that means the objects are further, instead the darker patches means the objects are closer 1.

Figure 1: Output depth Map example

2 State of the art

Over the years the problem of depth estimation has with a lot of attempts, the performance going better and faster to obtain and use it in different fields, like robotics and driverless mobility. The most important models afloat the community are:

- MiDas
- ZoeDepth
- PatchFusion
- Marigold

each model is based in the most cases on the previous one or on a custom build to exploit the performance obtained, improve the ability to capture more details and reach better performance. The first model *MiDas* has 3 different versions and each version reached a better accuracy score, an example of output from a given input 2

Figure 2: MiDas examples output

The goals reached are incredible, the afterwards models like *ZoeDepht* have the goal to make inferences in meters, its backbone was made with Midas, and the performance is better, it has the problem with the quality of details that are lost in blurred blobs and walls and sharp corners get wobbly.[1] A *patch fusion* network that fuses a globally consistent coarse prediction with finer, inconsistent tiled predictions via high-level feature guidance, A Global-to-Local (G2L) module that adds vital context to the fusion network, discarding the need for patch selection heuristics, and a Consistency-Aware Training (CAT) and Inference (CAI) approach, emphasizing patch overlap consistency and thus eradicating the necessity for post-processing. It uses UnrealLStereo4K to generate input with intricated details.[2] It worked on the issues which ZoeDepth has and resolved them, this net used ZoeDepth and its particular attention to be geometrically consistent.
The last one Marigold is a stable diffusion model which means it learns how an image slowly degrades—diffuses—into noise in a series of iterations, more particularly it learns what is lost in each step. The problem of diffusion models is they slowly generate images from noise in cycles, in each iteration structure emerges from the noise; and details are slowly revealed. The results of this process are stunning, not just because of the level of detail but also the consistency.[3]

3 Theorical Background

In this challenge, I used a lot of the properties of convolutional layers, in particular, the convolutional layers allow to connect sparsely, this can help the net to learn more about the intrinsic and semantic meaning of pictures and compress the dimension, thanks to use the **stride** parameter.

$$n_{out} = \left[\frac{n_{in}+2p-k}{s}\right] + 1$$

n_{in} : number of input features
n_{out} : number of output features
k : convolution kernel size
p : convolution padding size
s : convolution stride size

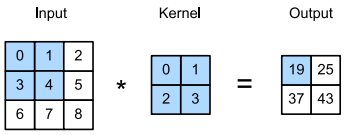

Figure 3: Convolutional layer example

On the encoder side, it is important to make a net with convolutional layers, or it is possible to use a pre-build net, which can be based on transfer learning. It is a technique for using a pre-trained model to make the training phase faster and use it for new tasks.[4] Another important technique which I used is fine-tuning; the trade-off to balance both techniques is complex, and it was the fundamental step during the training phase. Instead of transfer learning, the net can be manipulated, modifying its weights to better perform my specific task.[5] As for the decoder side, I decided to use convolutional transposed layers, which have the property of making upsampling, the net has to perform the opposite task that was done by the encoder 4.

$$n_{out} = (n_{in} - 1) \times s + k - 2p$$

n_{in} : number of input features
n_{out} : number of output features
k : transposed convolution kernel size
p : transposed convolution padding size
s : transposed convolution stride size

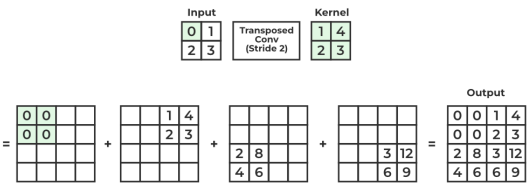

Figure 4: Transposed Convolutional example

An important parameter to modify to obtain better performance is the learning rate, in transfer learning the learning rate is important to start from a low values to increase the value with different technique, this hyperparameter have the task to take the net to convergence in faster or lower way in function of its values. To evaluate the better way to change the learning rate, I use the *CyclicLR* scheduler, it has the peculiarity of oscillating the value between boundaries, allowing exploration and exploitation.[6]

$$lr(t) = lr_{min} + (lr_{max} - lr_{min}) \times max(0, 1 - abs(2 \times cycle - 1 - \frac{t}{step\_size}))$$

lr_{min} : the minimum learning rate
lr_{max} : the initial maximum learning rate
$cycle$: the current cycle number
$step\_size$: the number of iterations in a half-cycle
t : the current iteration number

A deep neural network is trained on epochs and an epoche is divided in batch, the batch size is an hyperparameter, it depends on the dataset size and the quality of dataset items. With a small dataset, a small batch size gives to the model the chance to generalize more than a big batch size, because of the small size the model seen the net cannot overfitting, it is important to choose the parameter to not obtain the opposite situation, to avoid underfitting, and the problem of noise on the given input.[7]

4 Model

The model is based on Encoder-Decoder architecture, I used as a decoder Resnet50 and five convolutional layers to make upsampling and give in output the resultant depth map 5.

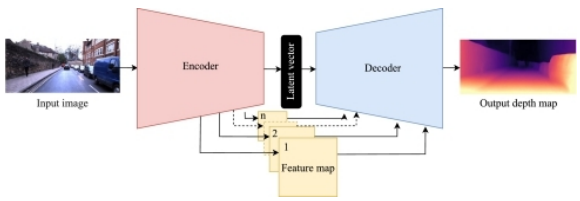

Figure 5: Example of Depth Estimation Net

4.1 Backbone

In my model, I used ResNet50 backbone, the importance of usage a pre-trained net is multiple and in particular this net gives the possibility to use it as a prebuilt **encoder**. I deleted the last two layers that are fully connected layer to make an inference of class, in this case, is not necessary. The architecture of the network is composed of five convolutional main stages 6.

The network takes in, a picture 224x224 as input, and after convolutional stages, the output channels are 2048 with the image height/width 7x7. On the first three convolutional layers I freeze the weights and allow the net to follow the transfer learning technique, to train the net faster. I choose to unfreeze the last layer to allow the net to learn about the new task and set an adaptive learning rate to improve the performance on the given dataset.

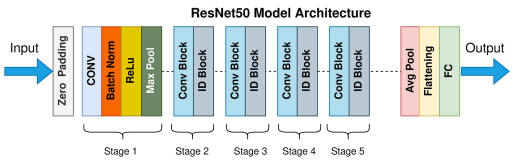

Figure 6: ResNet50 architecture

4.2 Decoder

The decoder side is composed by five transposed convolutional layers to make upsampling and give in output a desire depth map. The decoder net give in input 2048 feature maps, given by the output of ResNet, the next layer increase the resolution, but decrease the number of feature map. At the last transposed layer the output is just a feature map which is the prediction of the net.

5 Results

The net I made is created to make inference from a rgb pictures and give in output depth map. The ability of the net to estimate depth map, is measured by two metrics, which are:

- RMSE: represents the square root of the average squared differences between predicted and observed outcomes

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

- SSIM: a pixel-based metrics, it is based on the variance and the average of pixels and some parameter of correction that evaluate the pixel value similarities between preditcted and ground truth pictures

$$\mathrm{SSIM}(\mathbf{x}, \mathbf{y}) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{\left(\mu_x^2 + \mu_y^2 + C_1\right)\left(\sigma_x^2 + \sigma_y^2 + C_2\right)}.$$

The first metrics it is important to be the lowest as possible and the second one the highest. The model weights evaluated are given from *epoch 20*, at this epoch I obtained the following metrics values:

- $RMSE$: 2.57
- $SSIM$: 0.46

These values are calculated on the validation set.

6 Conclusion

The net obtained good performance on validation set thanks to the combination of multiple techniques: transfer learning and fine-tuning. The ideas to decrease the batch size and the choice of cyclic scheduler to set the learning rate between a range, help the net to improve performance. For future deploy I will try to implement data augmentation to make a better generalization from a given dataset, another ideas is to change the architecture of the net, test an approach on stable-diffusion net or transformer net which are state of the art and used on several fields. Both used attention layers to exploit the info obtained from a previous pictures and diffuse it along the net to make better and more general prediction.

References

[1] Bhat SF, Birkl R, Wofk D, Wonka P, Müller M. ZoeDepth: Zero-shot Transfer by Combining Relative and Metric Depth; 2023. Available from: https://arxiv.org/abs/2302.12288.

[2] Li Z, Bhat SF, Wonka P. PatchFusion: An End-to-End Tile-Based Framework for High-Resolution Monocular Metric Depth Estimation; 2023. Available from: https://arxiv.org/abs/2312.02284.

[3] Ke B, Obukhov A, Huang S, Metzger N, Daudt RC, Schindler K. Repurposing Diffusion-Based Image Generators for Monocular Depth Estimation; 2024. Available from: https://arxiv.org/abs/2312.02145.

[4] Zhuang F, Qi Z, Duan K, Xi D, Zhu Y, Zhu H, et al.. A Comprehensive Survey on Transfer Learning; 2020. Available from: https://arxiv.org/abs/1911.02685.

[5] Friederich S. Fine-Tuning. In: Zalta EN, Nodelman U, editors. The Stanford Encyclopedia of Philosophy. Winter 2023 ed. Metaphysics Research Lab, Stanford University; 2023. .

[6] Smith LN. Cyclical Learning Rates for Training Neural Networks; 2017. Available from: https://arxiv.org/abs/1506.01186.

[7] Radiuk P. Impact of Training Set Batch Size on the Performance of Convolutional Neural Networks for Diverse Datasets. Information Technology and Management Science. 2017 12;20:20-4.