Depth Estimation in monocular images

Nikhil Chapre
Vellore Institute of Technology
Chennai, India
nikhilc2209@gmail.com

Shaunak Deshpande
Vellore Institute of Technology
Chennai, India
shaunak.deshpande4@gmail.com

***Abstract*— This study is based on a Deep Learning approach to generate depth maps based on input 2d images. We train a CNN using the DIODE depth dataset and use it to generate depth maps and compare them with existing depth maps from the training set. This study uses dataset having both indoor and outdoor images from various scenes having varying depth.**

Keywords—deep learning, depth, RCNN, DIODE dataset, U-Net, SSIM

I. INTRODUCTION

Depth estimation in monocular images often poses to be a challenging task as without any perception of depth it is difficult to visualize distance of objects from the screen. It is easier in stereoscopic images because of multiple images from several camera angles. Since depth estimation in general has its various applications ranging from scene reconstruction for 3D printing or just for the sake of visualizations. To tackle this problem we try to use Supervised Learning to train models which can accurately identify the distance of objects from the screen(camera) and form depth maps for each pixel for easier visualization. Using convolutional neural networks we can form depth maps to assign different colours based on a colormap gradient scale. For this purpose we will make use of the DIODE indoor dataset which contains high quality indoor images with their respective depth maps and masks.

Using a supervised learning method will help us in understanding the difference between a actual depth and the depth map predicted by the model.

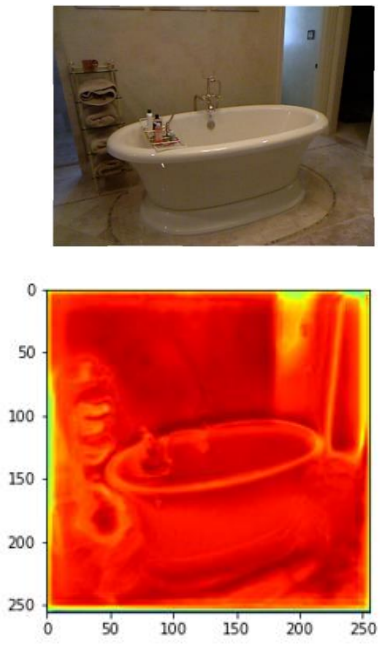

Fig 1. Depth map of an image from the DIODE dataset

Monocular depth estimation from images has also found various other applications ranging from training robots to understand scenes and grasp objects around them. Since robots made for smaller tasks having a single camera may not be able to differentiate distances between objects in a scene, using depth maps is a great approach. Though the RCNN architecture used is from U-Net, ResNet and DenseNets can also be used in certain other use cases. Also instead of passing images directly we can pass them in batches using a data pipeline.and tune the batch size manually. The architecture and need of such RCNN models has been discussed further along with their working.

During training it is necessary that we have a metric to analyse the performance of the model in different circumstances and the effect of tuning hyperparameters. We have used the inbuilt keras SparseCategoricalCrossEntropy method to evaluate losses as well proposed Structural Similarity Index Metric(SSIM) to get more information from out training metrics.

II. LITERATURE REVIEW

A. *Object Detection of Remote Sensing Airport Image Based on Improved Faster R-CNN: HAN Yongsai1, MA Shiping2, ZHANG Fei, LI Chenghao: The model used here was an efficient version of Region based Convolutional Neural Network. The use case was remote sensing of airport images and detecting objects in the same. A deeper basic ResNet was used, and add a new fully connected layer to the end detection network and combine the SoftMax classifier and 4 logistic regression classifiers for object detection according to the inter-class correlation of the object.*

*Satellite images of Airports were used. The faster RCNN model uses Region Proposal Net (RPN) which results in more accuracy, efficiency. This framework identified objects such as airports, civil aviation aircraft, fighter jets, transport planes, helicopters, tanks, and bridge objects. Each proposed ROI is convoluted with 3*3*n filters, pooling operation used on feature map obtained from convolution of last layer.*

B. B. *Real-Time Depth Estimation from 2D Images: Jack Zhu, Ralph Ma, They proposed model here is Transfer learning on real time images captured by camera, study the depth maps and estimate depths using heatmap on the NYU-dataset. They compared the transfer learning model on a pretrained and untrained network, compared performance differences. The lossy function used here is RMSE. Different pretrained model were used Mean, Make3d, Course Network, Course + Fine Network, Hierarchical VGG. Hierarchical VGG produced the best efficiency and the lowest RMSE values.*

C. *Fast Automatic Vehicle Annotation for Urban Traffic Surveillance: Detection and Annotation for Vehicles (DAVE) model was suggested which simultaneously runs 2 CNNs. 1st CNN is responsible for Vehicle detection. 2nd CNN is responsible for Annotation. A shallow Fully Convolutional Fast Vehicle Proposal Network was suggested for extracting all vehicles pose, color, type, simultaneously. These two nets are jointly optimized so that abundant latent knowledge learned from the deep empirical ALN can be exploited to guide training the much simpler FVPN. The proposed DAVE framework outperforms RCNN, Fast RCNN, Faster RCNN.*

D. *Depth Prediction from 2D Images: A Taxonomy and an Evaluation Study. They performed an Evaluation study by training the data on both supervised and unsupervised models. They compared multi view and Monocular models. The non-deep learning methods use stereo matching and have some disadvantages such as needing train data while runtime. The monocular is divided into Bootstrap and Iterative networks whose output is combined into a Refined network. The monocular 2D depth maps generated are refined and used as outputs.*

III. PROPOSED WORK

Like any other Deep learning model, we need a Lossy function to evaluate the accuracy and losses at each epoch of the model. Since we do not use a bootstrapped network and use iterative network as a subnetwork of Supervised Deep learning network, we need a good mathematical lossy function and aim to reduce the absolute value with each epoch.
We took inspiration from several papers and came up with a lossy function which is a combination of finding SSIM, that is, Structural Similarity Index Metric measures the image quality degradation. We compare the Predicted depth map and the Target depth map from the dataset. SSIM measures the losses that may be incurred from compression, mathematical computations and other losses.
Mathematical formula for SSIM:

$$\mathrm{SSIM}(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

Fig 2. SSIM function for loss metric

We also focus on edges by using the MARE metric. The Mean Absolute Relative Error measures weighted errors using gradients across X and the Y axis.
Mathematical formula for MARE:

$$\frac{1}{N}\sum_{p=1}^{N}\frac{|d_p^* - d_p|}{d_p^*}$$

Fig 3. MARE formula for loss metric

The advantage of using MARE over MAE, RMSE is that metric is not defined for null values meaning that we don't get 0 errors causing inconsistent values. Also considering the absolute values means that the positive and negative cancel each other out.
We finally combine the errors calculated by SSIM and MARE to get the loss in each epoch. Since we use tensor quantities, we use the tf.reducemean function to find mean.

IV. EXPERIMENTAL SETUP

Dataset: Dataset used is the Diode Dataset which has real high quality images of indoor and outdoor scenes as well as their depth map and masks required to build true depth maps for the RCNN to learn. The dataset has around 80-120 GB of training set images and 3 GB of validation set images. Training on both indoor and outdoor data enables CNN to perform well on any image and since 3D scene reconstruction was our main target it should perform well on test images which include indoor scenes.

Deep Learning Model: The model used in our case was based on U-Net architecture. U-Net is used to capture both high level details as well as low-level details in images. In the encoding part of the U-Net model we downscale the image and at each step we double the number of neurons in the hidden layer. This way we are able to capture the low-level details of the image such as texture difference or gradient separation. To do this we apply convolutional layer twice with a 3x3 kernel and then we apply max pooling to get feature maps which only have the most prominent features covered by the filter. After this we start upscaling the image again to its original size where we do the opposite by halving the number of neurons at each step of the convolutional layer and start concatenating the downscaled images and then apply the same convolutional layers twice to meet our original dimensions of the image.
In case of overfitting data we can also opt to include dropout layers at both the downscaling and upscaling block.

Software/Hardware: To run our Keras models on DIODE dataset, we used Nvidia CUDA Development toolkit 11.2 on Laptop RTX 3060 which has around 3840 CUDA cores. The CPU used is AMD Ryzen 5800H which features eight SMT-enabled Zen 3 cores running at 3.2 GHz (base clock speed) to 4.4 GHz (Boost). This system has 16GB RAM and the operating system used is Windows 10, The cuDNN version used is v8.1.1 which is compatible with our CUDA Development toolkit 11.2. The storage used is 512GB NVMe SSD which enables faster I/O operations.

V. RESULTS

A. *TRAINING DATA OUTPUTS*

Following are some output images recorded while training the data on default hyperparameters.

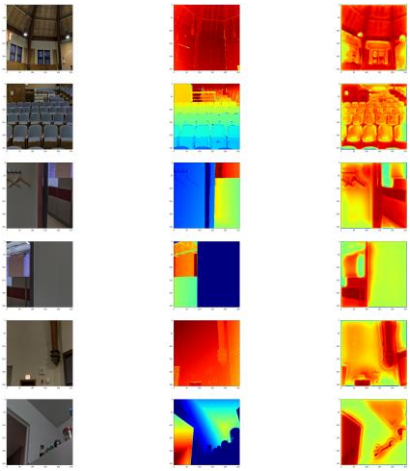

Fig 4. Training data output from U-Net model

As discussed before on the left we have we have our real images from the training set, in the middle we have our target depth maps which were there in the training data and on our right we have our predicted depth map output by the RCNN.

B. Evaluating losses per epoch

We run 10 epochs, even though 200 epochs are realistic for an accurate prediction, but lack of appropriate hardware for deep learning limited us to 10 epochs, results of the same are tabulated as follows.

TABLE 1

Epochs	Execution Time	Tensorflow Loss	Custom Loss
1/10	556ms/step	0.8211	0.7228
2/10	219ms/step	0.4998	0.6806
3/10	215ms/step	0.4045	0.3248
4/10	221ms/step	0.3710	0.3111
5/10	217ms/step	0.3444	0.2827
6/10	220ms/step	0.3291	0.2677
7/10	216ms/step	0.3266	0.2602
8/10	226ms/step	0.3408	0.2575
9/10	216ms/step	0.3133	0.2450
10/10	230ms/step	0.2753	0.2385

Fig 5. Tabular metrics from the trained model

We can easily observe the difference in losses at each epoch, tuning the learning rate and number of epochs can help us in fetching better results and minimizing losses even further.

VI. SUMMARY

Our extensive research and individual work has led to good results which can further be improved and worked upon by taking use of much improved hardware to train our model better. Our work on improving the present loss metrics helped us in improving our models overall performance. Though there were some cases where the model failed to deliver good results, which can easily be improved by feeding more training data to our RCNN model or using a different architecture like ResNets or DenseNets under different circumstances.

VII. REFERENCES

[1] Object Detection of Remote Sensing Airport Image Based on Improved Faster R-CNN: HAN Yongsai1, MA Shiping2, ZHANG Fei, LI Chenghao

[2] *B.* Real-Time Depth Estimation from 2D Images: Jack Zhu, Ralph Ma

[3] Fast Automatic Vehicle Annotation for Urban Traffic Surveillance: Detection and Annotation for Vehicles (DAVE)

[4] Depth Prediction from 2D Images: A Taxonomy and an Evaluation Study.

[5] Filippo Aleotti, Fabio Tosi, Matteo Poggi, and Stefano Mattoccia. Generative adversarial networks for unsupervised monocular depth prediction. In ECCV Workshops, 2018.

[6] Vincent Casser, Soeren Pirk, Reza Mahjourian, and Anelia Angelova. Depth prediction without the sensors: Leveraging structure for unsupervised learning from monocular videos. In AAAI, 2019.

[7] Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich, and Dacheng Tao. Deep ordinal regression network for monocular depth estimation. In CVPR, 2018.

[8] Miaomiao Liu, Mathieu Salzmann, and Xuming He. Discrete-continuous depth estimation from a single image. In CVPR, 2014.