Stereo Depth Estimation

Pahulmeet Singh

University of Bonn s6papahu@uni-bonn.de
https://github.com/Pahulmeet

Abstract. Depth estimation is a trivial concept for humans but a difficult task for machines. Understanding of depth is at the core of applications like 3D Scene construction, Augmented Reality and Autonomous Driving. Some of the approaches to solve this problem for machines include Radar/Lidar, Epipolar Geometry and deep learning(with monocular or stereo images)[6]. Stereo Depth estimation is based on how humans physiology solves this problem. In this approach, images from two cameras are used to triangulate and estimate distances[1]. Here, in this project we present a deep learning approach using 2D CNN and 3D CNN architecture to replicate the ground truth disparity of the stereo images. The model is tested on the KITI dataset with different models using the SmoothL1 loss and 3pixel loss as the performace metrics. The model structure aims to get features of the left and right images through the 2D-CNN and get the disparity through the 3D-CNN architecture.

1 Introduction

Depth estimation is a crucial concept in 3D vision and hence, it is important to find techniques that helps machines in understand it. It refers to a set of techniques and algorithms aiming to obtain a representation of the spatial structure of a scene[2].Depth perception is useful for scene understanding, scene reconstruction, virtual and augmented reality, obstacle avoidance, self-driving cars, robotics, and other applications. For this problem of depth estimation, Lidar has been a popular method for solving the problem but the price of hardware is high and LiDAR is sensitive to rain and snow, so there is a cheaper alternative: depth estimation with a stereo camera. This method is also called stereo matching. Deep learning has continuously been showing amazing achievements in computer vision and they help machines in the visual recognition tasks quite well. The Convolutional Neural Networks(CNNs) has numerous applications in this field and here, we want to employ this architecture for stereo depth estimation.

2 Method

To solve this problem, we follow the given baseline model pipeline provided in the introductory slides. This model consists of a Two-Dimensional Convolutional

Neural Architecture followed by a Three-Dimensional Convolutional Neural Architecture. The output from this deep learning model is finally passed on to a Disparity Regressor. The input consists of left and right images from the dataset passed to the 2D-CNN one after another thus, utilizing weight-sharing. Now, we get the output features from this 2D-CNN for both left and right images which in turn is now combined to form a Cost Volume Matrix. This new cost volume is passed to the next 3D-CNN and the output from this CNN is passed on to the regressor to get our final output image.

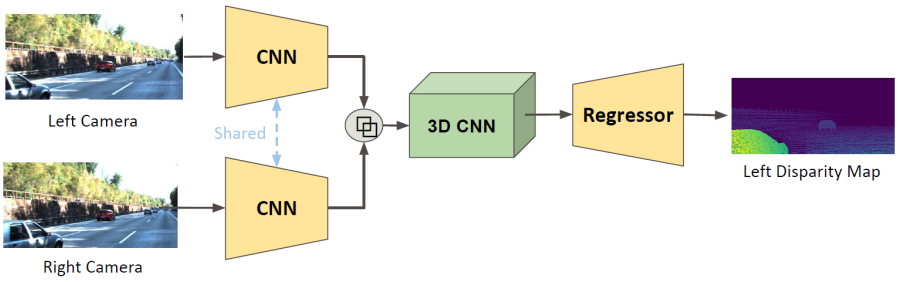

Fig. 1. Baseline model as stated in the introductory slides of the Project

As is evident from the baseline model, a number of different models can be constructed with different numbers of layers, kernel sizes, etc. This model construction is not limited to building the complete model from scratch but pre-trained models can be used as well. These pre-trained models are available for both 2D and 3D CNNs like Resnet[3], VGG[5], [4], and r3d_18[7], x3d_xs respectively. The KITTI dataset at hand is rather small and hence, data augmentations can be applied to this dataset. Moreover, pretraining on the SceneFlow dataset can be done as well. Next, we have optimizers, schedulers, and different parameters that need to be adjusted.

As for my experiments, I constructed various CNN models from scratch using Pytorch functionalities by changing the number of layers, kernel sizes, etc. Apart from this, models were constructed using pre-trained CNN models like Resnet[3], VGG[5] for the 2D CNN, and r3d_18[7] for the 3D CNN. However, there was a need to change some number of layers, kernels, and some changes to the building blocks to match the desired dimensions of the image dataset in hand. Moving ahead, some data augmentations were tested like ColorJitter, RandomSolarize, RandomAdjustSharpness, RandomAdjustContrast, and RandomEqualize. I did not try much of these as some other augmentation techniques might interfere with pixel positionings. For running these models, I used my local machine with NVIDIA RTX2060 with 6GB memory. I tried with Google Colab but it kept disconnecting and was comparatively slower and caused some other problems when reading the uploaded images from the drive like changing their order even though the shuffle was kept “False”.

3 Experimental Details

3.1 First Model Run

My experiments started with building a first basic baseline model that consisted of a five-layer 2D-CNN and a six-layer 3D-CNN but the number of out-channels in this model were kept quite small as at this point, the model was tested to check if the loss function and regressor were working fine while keeping the required dimensions intact as directed in the introductory slides. The SmoothL1 Loss function was used and the KITI dataset was passed through the model without any data augmentations. A disparity regressor was implemented from a GitHub repository. However, the first problem I faced was with the ground truth images as the pixel values were too large and I was unaware of this problem but the problem was resolved by dividing these values by 256 and thus converting the range close to the max disparity of 192. This model used an Adam optimizer with a learning rate of 3e-4 with a "ReduceLROnPlateau" Scheduler. The reason for using Adam optimizer was that many papes were using the same optimizer with a starting learning rate of 1e-4 or 3e-4 with a step scheduler. This baseline model was run for nearly 380 epochs and the reconstructed images started to show some progress.

3.2 Using Pre-trained models

Now, once I had this baseline model of mine running, I tried changing the number of layers and kernel sizes but the end result was similar and the model stagnated around a loss of 4.8 or 4.5 in the best-case scenario. Different models were run but they reached a plateau around 4.7 and then the scheduler would just start decreasing the learning rate more. At this point, I implemented models with pre-trained resnet18[3] or VGG[5] models but did not use the complete models but rather some building blocks followed by my 3D-CNN block. A pre-trained 3D-CNN namely, r3d_18[7] was also used with changes to the building blocks.

The best model out of these turned out to be the one that used the complete resnet18[3] model in the 2D CNN architecture with changes in strides from 2 to 1 in some layers to retain the desired dimensions. Also, the last 7th block of the resnet18[3] was changed to reduces the number of out-channels for memory limitation reasons. The 3D CNN was constructed based on the r3d_18[7] model i.e. the layers structures, parameters, and normalizations were based on that model. The reason for making such a model was as from what I understood is that, the 2D CNN is trying to learn the feature maps and I considered the pretrained resnet18[3] model a better fit for this as it must have previously learned features that might be helpful. The 3D CNN layer was trained from scratch as this layer. As for the dataset, I multiplied it by 3 by applying 3 types of data augmentations, namely, normalizing the dataset by the same values that were used to train the resnet18[3], using the ColorJitter with brightness and hue set to 0.3, and using RandomEqualise with a probability of 0.9. The batch size was kept at 1 and Adam optimizer with a learning rate of 3e-4 and betas 0.9 and

0.999 were used. "ReduceLROnPlateau" was used as a scheduler with a patience of 4 and factor 0.5.

3.3 The loss function

The reconstructed images started to look a lot better with this model but the SmoothL1 loss was still 4.2 after nearly 110 epochs. So, I had a little doubt about the "N" term in the given formula i.e., if "N" is equal to the total number of pixels or the number of non-zero pixels. For now, I have been considering the case to be the latter which is non-zero pixels only but if "N" was changed to total pixels of the target ground image then the loss was close to 2.03 but still off by 0.93 than the desired loss.

$$Smooth\ell 1 = \frac{1}{N}\sum_{i}^{N} li \tag{1}$$

$$li = \frac{1}{2\beta}(X_i - \hat{X}_i)^2 \quad abs(X_i - \hat{X}_i) < \beta \quad \& \quad li = abs(X_i - \hat{X}_i) - 0.5 * \beta \quad otherwise \tag{2}$$

3.4 Final Model

The model shown in tables 2 and 3 is the final model with 4730081 parameters that was initially run for 100 epochs on the KITI dataset but as mentioned above the dataset was replicated three times with data augmentation. Each epoch was taking close to 13 minutes to run and after 100 epochs, the original dataset without any augmentation was passed to the model for another 150 epochs i.e. only the first 150 training images and loss was computed on them.

4 Results

As can be seen in the figures2,3,4,5,6,7 the model is definitely learning the concept however, it has overfit on the trained data as is evident in the images. In testing images, we can see the car highlighted like in ground truth but the reconstructed image needs some more post-processing and also it misses multiple cars on some ocassions as well. The loss obtained from the SmoothL1 loss when only non-zero values considered for accounting "N" in Eq.1, the value is too far i.e. 3.89 (reqd. below 1.1) but when "N" is changed to all the pixels then it comes to around 1.99(still does not make the cut). The model was still learning but at a relatively lower pace as expected. Some better optimizations and especially pretraining was necessary to beat the 1.1 margin.

Best Model using Pre-Trained Resnet[3] in 2D-CNN with changes					
Operation	In Channels	Out Channels	Kernel	Stride	Padding
Conv2D	3	64	(7,7)	(1,1)	(3,3)
BatchNorm2D	64	-	-	-	-
ReLU	-	-	-	-	-
MaxPool2D	-	-	(3,3)	(2,2)	-
Conv2D	64	64	(3,3)	(1,1)	(1,1)
BatchNorm2D	64	-	-	-	-
ReLU	-	-	-	-	-
Conv2D	64	64	(3,3)	(1,1)	(1,1)
BatchNorm2D	64	-	-	-	-
Conv2D	64	64	(3,3)	(1,1)	(1,1)
BatchNorm2D	64	-	-	-	-
ReLU	-	-	-	-	-
Conv2D	64	64	(3,3)	(1,1)	(1,1)
BatchNorm2D	64	-	-	-	-
Conv2D	64	128	(3,3)	(1,1)	(1,1)
BatchNorm2D	128	-	-	-	-
ReLU	-	-	-	-	-
Conv2D	128	128	(3,3)	(1,1)	(1,1)
BatchNorm2D	128	-	-	-	-
Conv2D	64	128	(1,1)	(1,1)	-
Conv2D	128	128	(3,3)	(1,1)	(1,1)
BatchNorm2D	128	-	-	-	-
ReLU	-	-	-	-	-
Conv2D	128	128	(3,3)	(1,1)	(1,1)
BatchNorm2D	128	-	-	-	-
Conv2D	128	256	(3,3)	(1,1)	(1,1)
BatchNorm2D	256	-	-	-	-
ReLU	-	-	-	-	-
Conv2D	256	256	(3,3)	(1,1)	(1,1)
BatchNorm2D	256	-	-	-	-
Conv2D	128	256	(1,1)	(1,1)	-
BatchNorm2D	256	-	-	-	-
Conv2D	256	256	(3,3)	(1,1)	(1,1)
BatchNorm2D	256	-	-	-	-
ReLU	-	-	-	-	-
Conv2D	256	256	(3,3)	(1,1)	(1,1)
BatchNorm2D	256	-	-	-	-
Conv2D	256	256	(3,3)	(2,2)	(1,1)
BatchNorm2D	256	-	-	-	-
Conv2D	256	256	(3,3)	(1,1)	(1,1)
BatchNorm2D	256	-	-	-	-
Conv2D	128	32	(3,3)	(1,1)	(1,1)

Table 1. Table showing the 2D CNN layers from Resnet[3] with some changes to kernels and last building block to retain desired image sizes. Adam optimizer with learning rate of 3e-4, betas(0.9,0.999) and ReduceLROnPlateau Scheduler with patience of 4 and factor of 0.4 was used.

Best Model 3D-CNN inspired from r3d_18[7] model					
Operation	In Channels	Out Channels	Kernel	Stride	Padding
Conv3D	64	64	(3,3,3)	(1,1,1)	(1,1,1)
BatchNorm3D	64	-	-	-	-
ReLU	-	-	-	-	-
Conv3D	64	64	(3,3,3)	(1,1,1)	(1,1,1)
BatchNorm3D	64	-	-	-	-
ReLU	-	-	-	-	-
Conv3D	64	128	(3,3,3)	(1,1,1)	(1,1,1)
BatchNorm3D	128	-	-	-	-
ReLU	-	-	-	-	-
Conv3D	128	128	(3,3,3)	(1,1,1)	(1,1,1)
BatchNorm3D	128	-	-	-	-
ReLU	-	-	-	-	-
Conv3D	128	32	(3,3,3)	(1,1,1)	(1,1,1)
BatchNorm3D	32	-	-	-	-
ReLU	-	-	-	-	-
Conv3D	32	32	(3,3,3)	(1,1,1)	(1,1,1)
BatchNorm3D	32	-	-	-	-
ReLU	-	-	-	-	-
Conv2D	32	1	(3,3)	(1,1)	(1,1)

Table 2. Table showing the 3D CNN layers inspired from r3d_18[7] model.Adam optimizer with learning rate of 3e-4, betas(0.9,0.999) and ReduceLROnPlateau Scheduler with patience of 4 and factor of 0.4 was used.

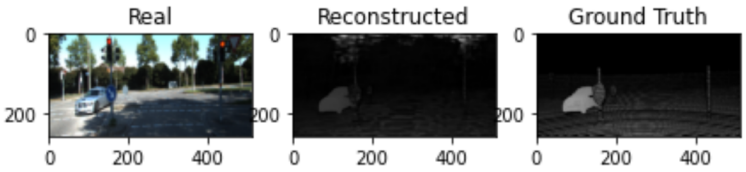

Fig. 2. Results on training data

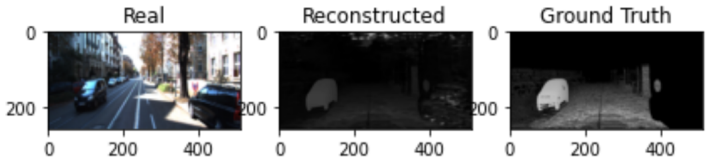

Fig. 3. Results on training data

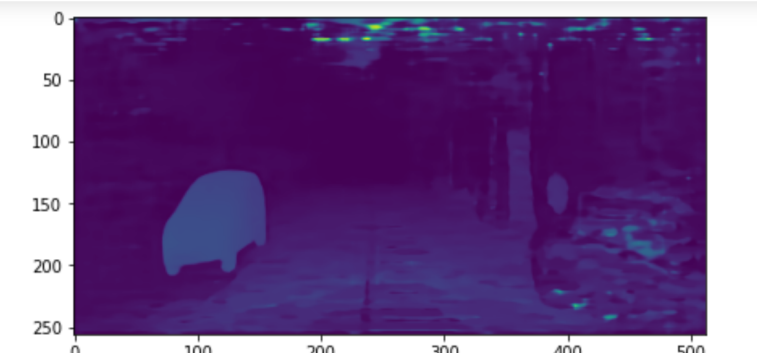

Fig. 4. Results on training data

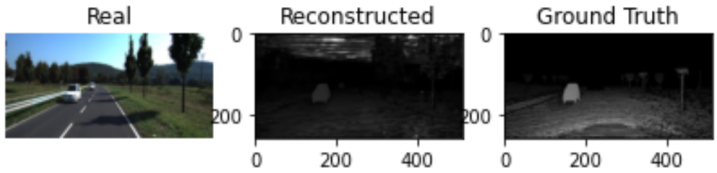

Fig. 5. Results on testing data

Fig. 6. Results on testing data

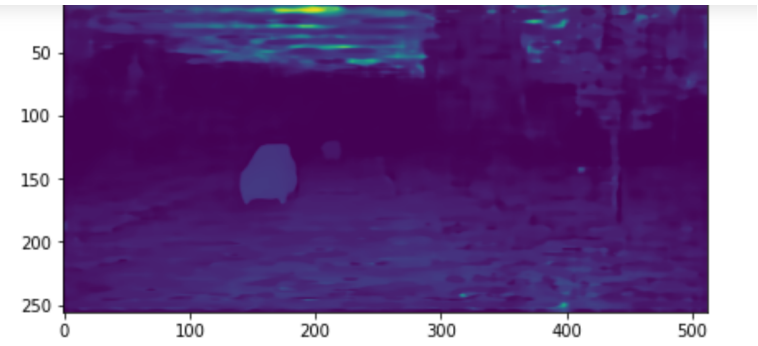

Fig. 7. Results on testing data

5 Conclusion

Utilizing deep learning in the field of depth estimation provides an alternative to traditional technique and expensive equipment and gives nice results as well. In this project, even though the losses obtained through the model were not as good but the model is learning to understand the concept at hand. Maybe, Pre-training would have solved this problem as the dataset size was too small. However, swithching to pretrained models like Resnet[3], VGG[5] and r3d_18[7] helped in designing better model architecture than starting from scratch.

6 Contributions

The project was completed by me alone. This includes getting to know about Stereo Depth Estimation, working with KITI and the sceneflow dataset,disparity, coding different models, data augmentations, experiments on different models and this report.

References

1. Depth estimation using monocular and stereo cues.
2. Depth estimation – an introduction pablo revuelta sanz, belén ruiz mezcua and josé m. sánchez pena.
3. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.
4. Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018.
5. Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.
6. Nikolai Smolyanskiy, Alexey Kamenev, and Stan Birchfield. On the importance of stereo for accurate depth estimation: An efficient semi-supervised deep neural network approach, 2020.
7. Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. A closer look at spatiotemporal convolutions for action recognition. *CoRR*, abs/1711.11248, 2017.