CS230

High Topology SEMI wafers - Height Variation Map Prediction

Muthukumar Kalyanaraman, Krishnachandiran Ravichandiran
mkalyana@stanford.edu, krishrav@stanford.edu
GitHub Project Link: https://github.com/krishnachandiran/CS230_Wafer_Height_Prediction

Abstract

In semiconductor industry, wafers have surface undulations (topological variations) typically up to <50um. Wafer types such as 3DIC have a lot of vertical IC structures with varying topology up to 800um. Optical inspection tools scanning these wafers for surface defects have smaller depth of focus at higher magnifications. For a wafer with surface variations, the images captured at higher magnification produces blurred images. Blurred images make traditional defect finding algorithms to perform poorly. To inspect 3D structures within a chip (die) at higher magnifications, currently there is a need to use complex optical engineering techniques, which is time consuming.

We propose to train a deep supervised CNN network with residual block that takes down-sampled high-resolution images obtained with focus assist from HW to predict the height variation map given a low-resolution image as input. The height variation map thus output can be used for various defect finding algorithms to boost throughput.

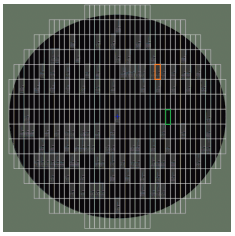
1 Introduction

In the field of semiconductor wafer defect inspection and metrology, to study the surface of wafer and to inspect them for any surface defects high precision objectives/camera with various magnifications are used. In KLA's wafer inspection tools, pre-capturing the focus map (which is a time-consuming process using real time Z-sensors and other engineering techniques) before the inspection is avoided to gain time or throughput. A real time proprietary Auto focus system to provide real time wafer depth information is used instead to inspect SEMI standard wafers.

While inspecting high topology wafers, the Auto focus systems are prone to focus failures particularly with higher magnification that have lesser Depth of focus (DOF). A focus assist map input at higher mags can make Auto Focus (AF) systems work reliably and accurately.

The below image shows how a wafer layout in a KLA Inspection system looks with a typical high-topology wafer having missing dies (or holes). The grid in white color is an artifact drawn over the output image to help guide the user to locate a die within a chip. The presence of one of the die (chip) on surface of wafer is marked by orange box, where as for the one where we don't have a chip is marked with a green colored box. When the AF system scans over a missing die at higher magnification, it struggles to find the real time focus and can error out.

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

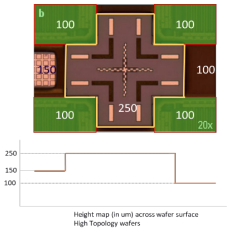


To achieve the end goal, we used an DL network that can determine the height variations in the wafer surface with an image captured at a lower magnification (faster to capture image in lower mags). This depth variation information thus predicted can assist the AF systems in real time to avoid focus errors. This also helps the inspections to detect defects on the surface of the wafers even at higher magnifications without errors or throughput loss.

2 Challenges

1. The main challenge is mining the data for training since wafer images are not available in the public domain due to IP reasons. Gathering the vast set of images of different types of high topology wafers is a challenge.

2. Another challenge is mapping focus values collected from high precision z-sensors to the training image



In the above image, as you can notice – wafer surface is uneven from left to right because of wafer design. Currently: the height map graph recorded by AF system takes the max of all Z height across 'y' for a particular 'x' position in the image and reports the same. For e.g., the green box with red outline above has a surface height as 100um; where as a structure along the same 'y' is 150um. The AF system along the y records the height as 150um (and not on a per pixel basis). Ideally, the z-sensor values collected with time as a scale need to be mapped to every (x, y) pixels in the image collected. If this DL solution can resolve the same, it's an added positive upshot.

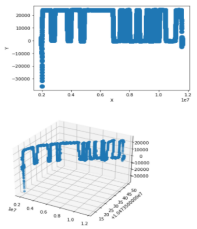
3 Related work

The above explained problem statement of finding the height variations on top of the wafer can also be logically converted into a problem of finding the depth of every point on the wafer with respect to the camera that is capturing it and distance of plane that holds the wafer. Hence, the problem can be mapped to the depth estimation of the object to the camera. There are many classical sensor-based methods, analytical methods and latest deep learning techniques were employed. Some non DL techniques are explained below in **Appendix - Related work**

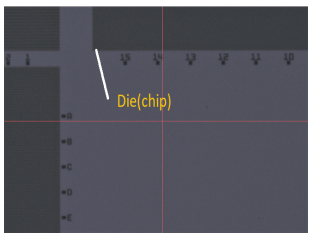
Deep learning: The interesting works that is related are from a paper by Iro Laina et al [5] on 'Deeper Depth Prediction with Fully Convolutional Residual Networks' which discussed depth prediction of Images. This paper introduces a fully CNN architecture to depth prediction, endowed with ***novel*** up-sampling blocks, that allows for dense output maps of higher resolution. This paper also proposes a more efficient scheme for up-convolutions and combine it with the concept of residual learning. Other notable works include Depth Estimation from Single Image Using CNN-Residual Network by Xiaobai Ma et.al [9], Estimating Depth from RGB and Sparse Sensing by Zhao Chen et.al.

4 Dataset and Features

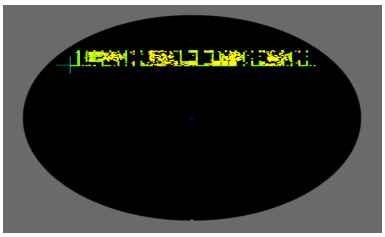
The raw image data set contains 2000+ of wafer inspection images with 16-bit gray scale images and corresponding Z-data (height data from KLA's AF systems) are captured at a higher magnification from the wafer scanning tool. The images are captured at different chip locations keeping the scan image width and height fixed. Data pre-processing involves segregating or clipping the images grabbed in a proper(equal) length along with sensor values collected. We make use of existing AF sensors out channel for getting the height information. The architecture and logic for capturing the Z-data from firmware did not exist before. *Specifically for this project, we had to inject some firmware level code to grab the data.* We were successful enough to grab the data from firmware which was spewing out AF data at 8000Hz. The data-grabbing is tuned to high frequency specifically for this project because we need the minute changes in nanosecond levels as the wafer scans happen at a very high speed. The plotted graph of Z-data for one left to right scan collected by this method is shown below:

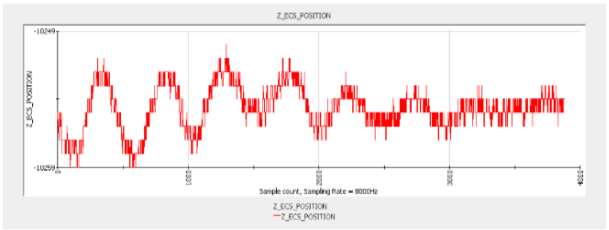


Sample Image: The below image is a clipped part image of the scan image used for training. Like the below image, many scanned images will be collected from the different wafer/tools and clipped using pre-processing.



The figure on the left below shows one row scan from left to right in a 300mm wafer with 3DICs while the figure on the right shows the corresponding z-data or height variation map grabbed from AF systems while performing the defect inspection.





Pre-processing: The wafer image is a gray scale on the left as its scanned is from a high magnification setting. As part of pre-processing, the image is down-sampled; sliced to match the network inputs[13] and served as Input vector (X) for training. Along with the image, HW sensor data from AF system is also collected(Y).

5 Methods

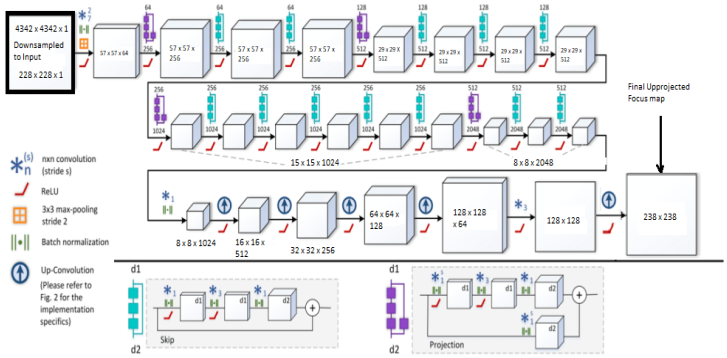
We use a modified version of FCRN (Fully Convolutional Residual Networks) to help predict the height variation map [13]. The base version of the FCNR model has been taken from the reference

cited. The FCRN layer shapes is customized to support out 16 bit greyscale channel and size instead of existing 3 channel and wafer images of 4342 * 4342 size instead of NYU dataset size.

The CNN + Residual model is the architecture proposed initially by [11] which uses ResNet50 [5] without the last fully-connect layer and pooling layer as feature extractor, and then uses up-projection blocks to up-sample the extracted feature. The FCRN network is composed of a first ResNet50 block with initialized pre-trained weights, and progresses with a sequence of convolutional and unpooling layers that make the network learn its upscaling. Dropout layers are then placed at the end alongside final convolution layers that yield the predicted result.

FCRN is one of the most used models for on-device depth prediction. It's interesting to note that the overall FCRN architecture is highly inspired by the U-Net scheme. Both use three downsampling and three upsampling convolutional blocks with fixed filter 3×3. Originally, U-Net was built with two convolutional layers in each block and the number of filters for all convolutional layers was kept constant. Conversely, in the FCRN implementation, the authors increased the number of subsequent layers to compensate for the loss of higher-resolution information caused by pooling.

We followed a similar approach as the above model is a proved one , but modified it to our need as shown below:



6 Experiments/Results/Discussion

Code and Experiments The initial data set has been collected for training, dev, and test from the target 3DIC wafers along with HW data from one lot of KLA wafers. The samples from the same are show in previous section. Data has to be pre-processed to do the down-sampling with height data synchronization to provide labeled output (for the training data). Challenges during the pre-processing have been captured in **Appendix - Challenges during Data Pre-processing**

Experiments and multiple iterations of training were done with different ways of feeding in the data to the network. Initially tried with full resolution of images and later with 16-bit images. Finally, 4342 * 4343 resolution images are downsampled to 228 * 228 size to decrease the training time and cost.

Experiments were done on the Z - data (Y - output) to make it normalized between 0 to 1 as float, as the Z (height data) range is from (0 to 25000). Finally, The Z-data which has the ground truth range from (0 to 25000) are also downsampled to (0 to 255) for faster convergence.

A separate pre-processing script '*mimReader.py*' for converting raw images to png format was used and '*syncPreprocesWaferData.py*' to process/sync/align the AF sensor data and with wafer image data and crop them into many pieces for train/test dataset. '*waferloader.py*' was used to read the wafer data and to downsample the dataset which was fed to *PyTorch*.

The pre-processed data is then trained using *train.py*. The pre-trained weights are loaded using helper methods in *weights.py* from *NYU_ResNet-UpProj.npy*. *fcrn.py* orchestrates the network architecture that is called from *train.py* as part of training the image data.

Baseline To speed up the work and to establish the baseline results using the model chosen, we have used the existing model training weights from pre-trained model on NYU Dataset. Since we have

modified the training data input layer(shape and channel), the pretrained weights cant be loaded as it is, so we have dropped few trained layers from pre-trained weight to match out input/output data.

Initially, the qualitative result is observed and gathered by taking the model train on the NYU dataset and used it to predict the depth map for the input from the Collected KLA Wafer dataset.

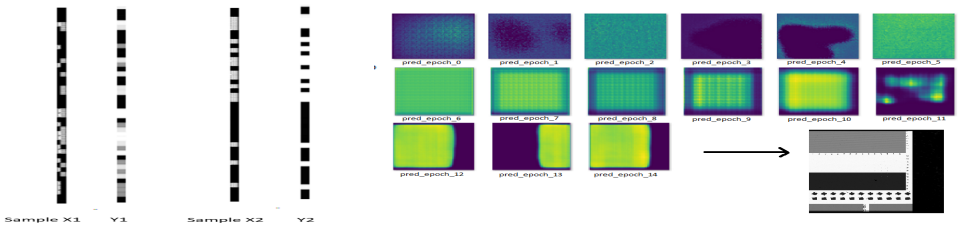
Network and Hyper-Parameters The modified FCRN network is tabulated in the **Appendix - Network Details**.

The convolutional block, consisting of a convolutional layer, batch normalization, and activation function. This is a PyTorch implementation adapted from [13] to tune to our data set. The following are the hyper-parameters that we used:

- Network Architecture: FCRN (ResNet50) with pre-trained weights
- Added: Up-projection layer to preserve the feature map for hi-res images
- No. of Images for training: 3000
- No. of Images for dev: 200
- No. of Images for test: 70
- Image size: 228 * 228
- batch size: 35
- epochs: 10
- learning rate: 1.0e-5
- momentum: 0.9
- weight decay: 0.0005

Results

- A significant part of the project work involved in pre-processing the data to align the image data (X) with (Y) data taking into consideration the machine boundary conditions
- The final result of synchronization of X and Y dataset could be seen in the below image (image on the left)
- Initial iterations with high resolution image, high range z-data, without pre-trained weights etc., did not converge all that well with high loss throughout multiple epochs
- Finally: after the changes in the data as well as to the network, we are able to find a good result as shown in the image below. As seen in the image below, after the 12th epoch we are able to good convergence of the network (image on the right)



Discussions Some more tuning of network architecture is required to improve the training accuracy and try changing the batch size, adding regularization improve the inference accuracy are to be carried out. We plan to train the network with variety of different types of wafer data rather than just one to get better results. For want of time and IP, we could not arrive at conducive results; however we want to pursue this further and solve this interesting Computer vision problem.

7 Contributions

Krishnachandiran was involved in data pre-processing and setting network up for the training. Muthukumar was involved in tuning the hyper-parameters for the network chosen and providing up with different ideas.

8 Conclusion/Future Work

To get a faster output, specifically for the project: we had to some compromise and do changes to few areas different from what we initially planned. In future, we plan to try the following:

1. Grab 3 channel images from the tool and train
2. Making the network model to fix bounding boxes to detect the die start and end positions
3. Use the high resolution z-data itself as it for training or atleast normalize it, instead of losing precision information as did for the project
4. The data we used for this project is 5X, we wanted to experiment this with high resolution like 20X , 35X which gives near to pixel level data for training
5. For IP reasons, we are able to use only single type of wafer data to do training. The network need to be trained with different types of wafer and layer (in case when using this project within the company while adopting it as a solution)

This is an interesting problem to solve with AI based solution for KLA and can significantly boost the business in the chosen field of Computer vision/Semi conductor space. With varied data from different customer wafers and by tuning the network along with significant time investment, we expect to produce positive results solving for the business.

References

[1] Chaoqiang Zhao, Qiyu Sun, Chongzhen Zhang, Yang Tang, Feng Qian. *Monocular Depth Estimation Based on Deep Learning: An Overview*

[2] Alex Fu, Yannick Meier, Elena Chen, Nithin Poduval. *Depth Prediction with CNN on Sequential RGB Inputs*

[3] Jack Zhu, Ralpha Ma. *Real-Time Depth Estimation from 2D Images*

[4] Xinjing Cheng, Peng Wang and Ruigang Yang. *Learning Depth with Convolutional Spatial Propagation Network*

[5] Iro Laina, Christian Rupprecht, Vaseleios Belagiannis, Federico Tombari, Nassir Navab. *Deeper depth Prediction with Fully connected Residual Network*

[6] Shir Gur, Lior Wolf *Single Image Depth Estimation Trained via Depth from Defocus Cues*

[7] J. Surh, H.- G. Jeon, Y. Park, S. Im, H. Ha, and I. S. Kweon *Noise robust depth from focus using a ring difference filter*. In IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), 2017

[8] S. Zhuo and T. Sim. *Defocus map estimation from a single image*. Pattern Recognition, 44(9):1852–1858, 2011.

[9] Xiaobai Ma,Depth, Zhenglin Geng, Zhi Bie *Estimation from Single Image Using CNN-Residual Network*

[10] Zhao Chen, Vijay Badrinarayanan, Gilad Drozdov, Andrew Rabinovich *Estimating Depth from RGB and Sparse Sensing*

[11] Igor Vasiljevic, Nick Kolkin, Shanyi Zhang, Ruotian Luo, Haochen Wang, Falcon Z. Dai, Andrea F. Daniele, Mohammadreza Mostajabi, Steven Basart, Matthew R. Walter, Gregory Shakhnarovich - *DIODE: A Dense Indoor and Outdoor DEpth Dataset*

[12] Nathan Silberman, Pushmeet Kohli, Derek Hoiem, Rob Fergus - *NYU Depth Dataset V2*

[13] Depth Estimation Models with Fully Convolutional Residual Networks (FCRN) - https://neptune.ai/blog/depth-estimation-models-with-fully-convolutional-residual-networks-fcrn

9 Appendix

Challenges during Data Pre-processing Good amount of time was consumed in the data collection and writing the logic for data pre-processing. Few challenges to mention: we had the image data without any physical coordinates, but the z-data grabbed at 8000Hz along with encoder counts (encoder counts are physical measurement in micron scale. 32 encoder counts = 1 micron). We know that camera resolution is 1 pixel = 1.2913298 um. So while plotting the z-data along the image, either flooring or rounding the pixel value leads to

bigger or smaller dimension of Y. Finally plotted the Z-data into Image gradient and compressed the image to match the size of X input, which syncs the data better.

Network Details

Layer (type) Output Shape Params

- Conv2d-1 [35, 64, 114, 114] 3,136
- BatchNorm2d-2 [35, 64, 114, 114] 128
- ReLU-3 [35, 64, 114, 114] 0
- MaxPool2d-4 [35, 64, 57, 57] 0
- Conv2d-5 [35, 64, 57, 57] 4,096
- BatchNorm2d-6 [35, 64, 57, 57] 128
- ReLU-7 [35, 64, 57, 57] 0
- Conv2d-8 [35, 64, 57, 57] 36,864
- BatchNorm2d-9 [35, 64, 57, 57] 128
- ReLU-10 [35, 64, 57, 57] 0
- Conv2d-11 [35, 256, 57, 57] 16,384
- BatchNorm2d-12 [35, 256, 57, 57] 512
- Conv2d-13 [35, 256, 57, 57] 16,384
- BatchNorm2d-14 [35, 256, 57, 57] 512
- ReLU-15 [35, 256, 57, 57] 0
- Bottleneck-16 [35, 256, 57, 57] 0
- Conv2d-17 [35, 64, 57, 57] 16,384
- BatchNorm2d-18 [35, 64, 57, 57] 128
- ReLU-19 [35, 64, 57, 57] 0
- Conv2d-20 [35, 64, 57, 57] 36,864
- BatchNorm2d-21 [35, 64, 57, 57] 128
- ReLU-22 [35, 64, 57, 57] 0
- Conv2d-23 [35, 256, 57, 57] 16,384
- BatchNorm2d-24 [35, 256, 57, 57] 512
- ReLU-25 [35, 256, 57, 57] 0
- Bottleneck-26 [35, 256, 57, 57] 0
- Conv2d-27 [35, 64, 57, 57] 16,384
- BatchNorm2d-28 [35, 64, 57, 57] 128
- ReLU-29 [35, 64, 57, 57] 0
- Conv2d-30 [35, 64, 57, 57] 36,864
- BatchNorm2d-31 [35, 64, 57, 57] 128
- ReLU-32 [35, 64, 57, 57] 0
- Conv2d-33 [35, 256, 57, 57] 16,384
- BatchNorm2d-34 [35, 256, 57, 57] 512
- ReLU-35 [35, 256, 57, 57] 0
- Bottleneck-36 [35, 256, 57, 57] 0
- Conv2d-37 [35, 128, 57, 57] 32,768
- BatchNorm2d-38 [35, 128, 57, 57] 256
- ReLU-39 [35, 128, 57, 57] 0
- Conv2d-40 [35, 128, 29, 29] 147,456
- BatchNorm2d-41 [35, 128, 29, 29] 256
- ReLU-42 [35, 128, 29, 29] 0

- Conv2d-43 [35, 512, 29, 29] 65,536
- BatchNorm2d-44 [35, 512, 29, 29] 1,024
- Conv2d-45 [35, 512, 29, 29] 131,072
- BatchNorm2d-46 [35, 512, 29, 29] 1,024
- ReLU-47 [35, 512, 29, 29] 0
- Bottleneck-48 [35, 512, 29, 29] 0
- Conv2d-49 [35, 128, 29, 29] 65,536
- BatchNorm2d-50 [35, 128, 29, 29] 256
- ReLU-51 [35, 128, 29, 29] 0
- Conv2d-52 [35, 128, 29, 29] 147,456
- BatchNorm2d-53 [35, 128, 29, 29] 256
- ReLU-54 [35, 128, 29, 29] 0
- Conv2d-55 [35, 512, 29, 29] 65,536
- BatchNorm2d-56 [35, 512, 29, 29] 1,024
- ReLU-57 [35, 512, 29, 29] 0
- Bottleneck-58 [35, 512, 29, 29] 0
- Conv2d-59 [35, 128, 29, 29] 65,536
- BatchNorm2d-60 [35, 128, 29, 29] 256
- ReLU-61 [35, 128, 29, 29] 0
- Conv2d-62 [35, 128, 29, 29] 147,456
- BatchNorm2d-63 [35, 128, 29, 29] 256
- ReLU-64 [35, 128, 29, 29] 0
- Conv2d-65 [35, 512, 29, 29] 65,536
- BatchNorm2d-66 [35, 512, 29, 29] 1,024
- ReLU-67 [35, 512, 29, 29] 0
- Bottleneck-68 [35, 512, 29, 29] 0
- Conv2d-69 [35, 128, 29, 29] 65,536
- BatchNorm2d-70 [35, 128, 29, 29] 256
- ReLU-71 [35, 128, 29, 29] 0
- Conv2d-72 [35, 128, 29, 29] 147,456
- BatchNorm2d-73 [35, 128, 29, 29] 256
- ReLU-74 [35, 128, 29, 29] 0
- Conv2d-75 [35, 512, 29, 29] 65,536
- BatchNorm2d-76 [35, 512, 29, 29] 1,024
- ReLU-77 [35, 512, 29, 29] 0
- Bottleneck-78 [35, 512, 29, 29] 0
- Conv2d-79 [35, 256, 29, 29] 131,072
- BatchNorm2d-80 [35, 256, 29, 29] 512
- ReLU-81 [35, 256, 29, 29] 0
- Conv2d-82 [35, 256, 15, 15] 589,824
- BatchNorm2d-83 [35, 256, 15, 15] 512
- ReLU-84 [35, 256, 15, 15] 0
- Conv2d-85 [35, 1024, 15, 15] 262,144
- BatchNorm2d-86 [35, 1024, 15, 15] 2,048
- Conv2d-87 [35, 1024, 15, 15] 524,288
- BatchNorm2d-88 [35, 1024, 15, 15] 2,048
- ReLU-89 [35, 1024, 15, 15] 0

- Bottleneck-90 [35, 1024, 15, 15] 0
- Conv2d-91 [35, 256, 15, 15] 262,144
- BatchNorm2d-92 [35, 256, 15, 15] 512
- ReLU-93 [35, 256, 15, 15] 0
- Conv2d-94 [35, 256, 15, 15] 589,824
- BatchNorm2d-95 [35, 256, 15, 15] 512
- ReLU-96 [35, 256, 15, 15] 0
- Conv2d-97 [35, 1024, 15, 15] 262,144
- BatchNorm2d-98 [35, 1024, 15, 15] 2,048
- ReLU-99 [35, 1024, 15, 15] 0
- Bottleneck-100 [35, 1024, 15, 15] 0
- Conv2d-101 [35, 256, 15, 15] 262,144
- BatchNorm2d-102 [35, 256, 15, 15] 512
- ReLU-103 [35, 256, 15, 15] 0
- Conv2d-104 [35, 256, 15, 15] 589,824
- BatchNorm2d-105 [35, 256, 15, 15] 512
- ReLU-106 [35, 256, 15, 15] 0
- Conv2d-107 [35, 1024, 15, 15] 262,144
- BatchNorm2d-108 [35, 1024, 15, 15] 2,048
- ReLU-109 [35, 1024, 15, 15] 0
- Bottleneck-110 [35, 1024, 15, 15] 0
- Conv2d-111 [35, 256, 15, 15] 262,144
- BatchNorm2d-112 [35, 256, 15, 15] 512
- ReLU-113 [35, 256, 15, 15] 0
- Conv2d-114 [35, 256, 15, 15] 589,824
- BatchNorm2d-115 [35, 256, 15, 15] 512
- ReLU-116 [35, 256, 15, 15] 0
- Conv2d-117 [35, 1024, 15, 15] 262,144
- BatchNorm2d-118 [35, 1024, 15, 15] 2,048
- ReLU-119 [35, 1024, 15, 15] 0
- Bottleneck-120 [35, 1024, 15, 15] 0
- Conv2d-121 [35, 256, 15, 15] 262,144
- BatchNorm2d-122 [35, 256, 15, 15] 512
- ReLU-123 [35, 256, 15, 15] 0
- Conv2d-124 [35, 256, 15, 15] 589,824
- BatchNorm2d-125 [35, 256, 15, 15] 512
- ReLU-126 [35, 256, 15, 15] 0
- Conv2d-127 [35, 1024, 15, 15] 262,144
- BatchNorm2d-128 [35, 1024, 15, 15] 2,048
- ReLU-129 [35, 1024, 15, 15] 0
- Bottleneck-130 [35, 1024, 15, 15] 0
- Conv2d-131 [35, 256, 15, 15] 262,144
- BatchNorm2d-132 [35, 256, 15, 15] 512
- ReLU-133 [35, 256, 15, 15] 0
- Conv2d-134 [35, 256, 15, 15] 589,824
- BatchNorm2d-135 [35, 256, 15, 15] 512
- ReLU-136 [35, 256, 15, 15] 0

- Conv2d-137 [35, 1024, 15, 15] 262,144
- BatchNorm2d-138 [35, 1024, 15, 15] 2,048
- ReLU-139 [35, 1024, 15, 15] 0
- Bottleneck-140 [35, 1024, 15, 15] 0
- Conv2d-141 [35, 512, 15, 15] 524,288
- BatchNorm2d-142 [35, 512, 15, 15] 1,024
- ReLU-143 [35, 512, 15, 15] 0
- Conv2d-144 [35, 512, 8, 8] 2,359,296
- BatchNorm2d-145 [35, 512, 8, 8] 1,024
- ReLU-146 [35, 512, 8, 8] 0
- Conv2d-147 [35, 2048, 8, 8] 1,048,576
- BatchNorm2d-148 [35, 2048, 8, 8] 4,096
- Conv2d-149 [35, 2048, 8, 8] 2,097,152
- BatchNorm2d-150 [35, 2048, 8, 8] 4,096
- ReLU-151 [35, 2048, 8, 8] 0
- Bottleneck-152 [35, 2048, 8, 8] 0
- Conv2d-153 [35, 512, 8, 8] 1,048,576
- BatchNorm2d-154 [35, 512, 8, 8] 1,024
- ReLU-155 [35, 512, 8, 8] 0
- Conv2d-156 [35, 512, 8, 8] 2,359,296
- BatchNorm2d-157 [35, 512, 8, 8] 1,024
- ReLU-158 [35, 512, 8, 8] 0
- Conv2d-159 [35, 2048, 8, 8] 1,048,576
- BatchNorm2d-160 [35, 2048, 8, 8] 4,096
- ReLU-161 [35, 2048, 8, 8] 0
- Bottleneck-162 [35, 2048, 8, 8] 0
- Conv2d-163 [35, 512, 8, 8] 1,048,576
- BatchNorm2d-164 [35, 512, 8, 8] 1,024
- ReLU-165 [35, 512, 8, 8] 0
- Conv2d-166 [35, 512, 8, 8] 2,359,296
- BatchNorm2d-167 [35, 512, 8, 8] 1,024
- ReLU-168 [35, 512, 8, 8] 0
- Conv2d-169 [35, 2048, 8, 8] 1,048,576
- BatchNorm2d-170 [35, 2048, 8, 8] 4,096
- ReLU-171 [35, 2048, 8, 8] 0
- Bottleneck-172 [35, 2048, 8, 8] 0
- Conv2d-173 [35, 1024, 8, 8] 2,097,152
- BatchNorm2d-174 [35, 1024, 8, 8] 2,048
- Conv2d-175 [35, 512, 8, 8] 4,719,104
- Conv2d-176 [35, 512, 8, 8] 3,146,240
- Conv2d-177 [35, 512, 8, 8] 3,146,240
- Conv2d-178 [35, 512, 8, 8] 2,097,664
- Conv2d-179 [35, 512, 8, 8] 4,719,104
- Conv2d-180 [35, 512, 8, 8] 3,146,240
- Conv2d-181 [35, 512, 8, 8] 3,146,240
- Conv2d-182 [35, 512, 8, 8] 2,097,664
- BatchNorm2d-183 [35, 512, 16, 16] 1,024

- ReLU-184 [35, 512, 16, 16] 0
- Conv2d-185 [35, 512, 16, 16] 2,359,808
- BatchNorm2d-186 [35, 512, 16, 16] 1,024
- BatchNorm2d-187 [35, 512, 16, 16] 1,024
- ReLU-188 [35, 512, 16, 16] 0
- UpProject-189 [35, 512, 16, 16] 0
- Conv2d-190 [35, 256, 16, 16] 1,179,904
- Conv2d-191 [35, 256, 16, 16] 786,688
- Conv2d-192 [35, 256, 16, 16] 786,688
- Conv2d-193 [35, 256, 16, 16] 524,544
- Conv2d-194 [35, 256, 16, 16] 1,179,904
- Conv2d-195 [35, 256, 16, 16] 786,688
- Conv2d-196 [35, 256, 16, 16] 786,688
- Conv2d-197 [35, 256, 16, 16] 524,544
- BatchNorm2d-198 [35, 256, 32, 32] 512
- ReLU-199 [35, 256, 32, 32] 0
- Conv2d-200 [35, 256, 32, 32] 590,080
- BatchNorm2d-201 [35, 256, 32, 32] 512
- BatchNorm2d-202 [35, 256, 32, 32] 512
- ReLU-203 [35, 256, 32, 32] 0
- UpProject-204 [35, 256, 32, 32] 0
- Conv2d-205 [35, 128, 32, 32] 295,040
- Conv2d-206 [35, 128, 32, 32] 196,736
- Conv2d-207 [35, 128, 32, 32] 196,736
- Conv2d-208 [35, 128, 32, 32] 131,200
- Conv2d-209 [35, 128, 32, 32] 295,040
- Conv2d-210 [35, 128, 32, 32] 196,736
- Conv2d-211 [35, 128, 32, 32] 196,736
- Conv2d-212 [35, 128, 32, 32] 131,200
- BatchNorm2d-213 [35, 128, 64, 64] 256
- ReLU-214 [35, 128, 64, 64] 0
- Conv2d-215 [35, 128, 64, 64] 147,584
- BatchNorm2d-216 [35, 128, 64, 64] 256
- BatchNorm2d-217 [35, 128, 64, 64] 256
- ReLU-218 [35, 128, 64, 64] 0
- UpProject-219 [35, 128, 64, 64] 0
- Conv2d-220 [35, 64, 64, 64] 73,792
- Conv2d-221 [35, 64, 64, 64] 49,216
- Conv2d-222 [35, 64, 64, 64] 49,216
- Conv2d-223 [35, 64, 64, 64] 32,832
- Conv2d-224 [35, 64, 64, 64] 73,792
- Conv2d-225 [35, 64, 64, 64] 49,216
- Conv2d-226 [35, 64, 64, 64] 49,216
- Conv2d-227 [35, 64, 64, 64] 32,832
- BatchNorm2d-228 [35, 64, 128, 128] 128
- ReLU-229 [35, 64, 128, 128] 0
- Conv2d-230 [35, 64, 128, 128] 36,928

- BatchNorm2d-231 [35, 64, 128, 128] 128
- BatchNorm2d-232 [35, 64, 128, 128] 128
- ReLU-233 [35, 64, 128, 128] 0
- UpProject-234 [35, 64, 128, 128] 0
- Dropout2d-235 [35, 64, 128, 128] 0
- Conv2d-236 [35, 1, 128, 128] 577
- ReLU-237 [35, 1, 128, 128] 0
- Upsample-238 [35, 1, 228, 228] 0

Total params: 63,565,377 Trainable params: 63,565,377 Non-trainable params: 0

Input size (MB): 6.94 Forward/backward pass size (MB): 15892.93 Params size (MB): 242.48 Estimated Total Size (MB): 16142.35

Related work

Sensors based: The traditional methods of estimation of the depth of any object from a single point is done through various methods like laser scanning and various sensors like sonar, radar, IR etc.,

Stereo Vision The concept of **Triangularization** with stereoscopic vision cameras was one of the well-known methods of estimating the depth with two or more cameras and their spatial coordinates

Analytical methods:

Depth from focus variations: This method involves the capturing of the same scene with different focal distances. The problem is solved analytically with cues and estimated depth based on a variety of blur models, such as the Ring Difference Filter like work of J. Surh.et.al [7]

Depth from defocus cues: Previous method allows for camera parameters to change for eg., changing the lens to vary the focal length. But this method is used in the constraint where camera parameters are not allowed to change. Zhuo et al. [8] for eg, estimated the amount of spatially varying defocus blur at edge locations.