CS/RBE 549 Computer Vision, Fall 2021

Project Report

Team Justice League

Member	Signature	Contribution (%)
Rahul Allam	_____	33.33
Aakash Thorat	_____	33.33
Gaurav Bhosale	_____	33.33

Grading:		
	Approach	_____/15
	Justification	_____/5
	Analysis	_____/15
	Testing & Examples	_____/15
	Documentation	_____/10
	Difficulty	_____/10
	Professionalism	_____/10
	Presentation	_____/20

TABLE OF CONTENTS

LIST OF FIGURES

SUMMARY

In this project, we take upon monocular depth estimation in indoor scenes using deep learning models. Specifically, we opted for a light-weight encoder decoder architecture to generate dense depth map of an input RGB indoor image. Firstly, we discuss the importance of depth in robotics applications. Later, we formulate the problem and discuss about the NYU-Depth V2 dataset. We present the dataset pre-processing steps and comprehensively explain the model and the architecture. We justify the reason behind choosing this model/implementation and the reason behind it. Additionally, we present our training results and plot the optimization loss against number of training epochs. We present the dense depth maps generated by the model on indoor scenes and finally, we conclude the report by describing potential applications, pros, cons and future scope.

INTRODUCTION

Depth is a very crucial piece of information, required to interact with the world around. It is of immense importance as it has numerous applications in the robotics and automation industry. Human perception of depth is an overly complex phenomenon. The desire to give machines and robots vision has fueled the innovation in developing several methods and technologies that compute depth. The acoustic approach of using Sonar sensors for obstacle detection or estimating depth is very traditional and obsolete. The recent development of Lidars, Stereo Cameras, and Kinect sensors proved to be groundbreaking. But even these technologies fail when it comes to estimating depth information from a 2D RGB Image. Also, they are very heavy and power-consuming. This poses a limitation on deploying them for many micro-robotic applications. Thus, such problems can be tackled by using lightweight, low cost, and highly efficient monocular cameras, for example in micro aerial vehicles.

This problem of finding the Depth Information from a single 2D Image can be solved using CNNs. The earliest Work on Monocular Depth using machine learning was published in 2006 (Ashutosh Saxena, 2006) and since then there has been a lot of research going on to extract depth information from pictorial cues in the 2D images. This project aims at finding a dense depth map for any given 2-dimensional RGB Image using Convolutional Neural Networks. The network uses an encoder-decoder architecture for this purpose. The results are evaluated using various performance metrics described later in the report. The CNN model has been trained on NYU Depth V2 indoor images Dataset.

PROBLEM FORMULATION

The goal is to generate a dense depth map of the same size as the input RGB Image. The train set T has M training examples.

Given a training set $T = \{(I^i, D^i)\}^M$, where $I^i \in I$ and $D^i \in D$

The task is to learn a non-linear mapping $\Phi: I \rightarrow D$, where I is Image and D is Depth map.

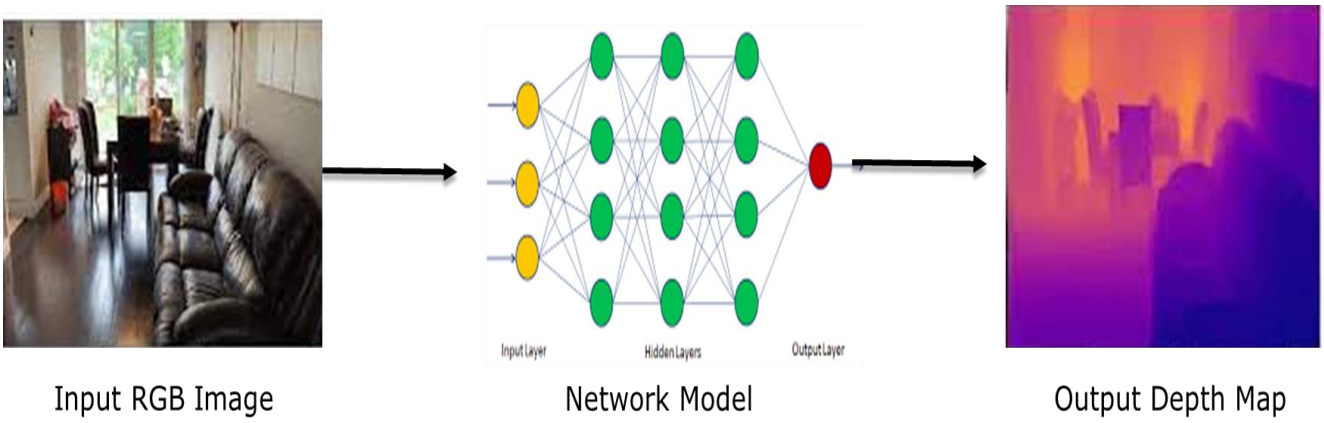

Figure 1 RGB image fed to a neural network to generate depth map

In figure 1, on the left, we have a 2D RGB image which is fed as an input to the CNN. The output as shown on the right side is the corresponding depth map generated by the CNN. The CNN's encoder-decoder architecture non-linearly maps the input to the output depth map.

How do Neural Networks perceive depth?

According to paper (Dijk, 2019), neural networks deduce depth information from images using the following cues:

- Position in the image
    - Objects which are higher in the image are above the horizon.
- Occlusion
    - Objects that are closer block those that lie behind them. Occlusion provides information on depth order, but not distance.
- Texture density
    - Textured surfaces further away are finer grained.
- Linear perspective
    - Straight, parallel lines in the physical world appear to converge in the image.
- Apparent size of objects
    - Objects that are further away from the camera appear.
- Shading and illumination
    - Surfaces appear brighter when they directly face the source of illumination.
- Focus blur
    - Objects that lie in front or behind the focal plane appear blurred.
- Aerial perspective
    - Very far away objects have less contrast and appear to have blue tint.

DATASET

The NYU-Depth V2 data set (silberman, n.d.) is made up of images from a variety of indoor scenes as recorded by both the RGB and Depth cameras from the Microsoft Kinect. It consists of pairs of RGB and Depth frames that have been annotated with dense depth labels for every image in the dataset. In addition to the projected depth maps, we have included a set of preprocessed depth maps whose missing values have been filled.

The dataset also consists of other information such as acceleration data, depths, images, instances, labels, class, scenes, and scene types. We use a subset of these images for our training. We use paired dataset on 10,000 images of RGB and depth.

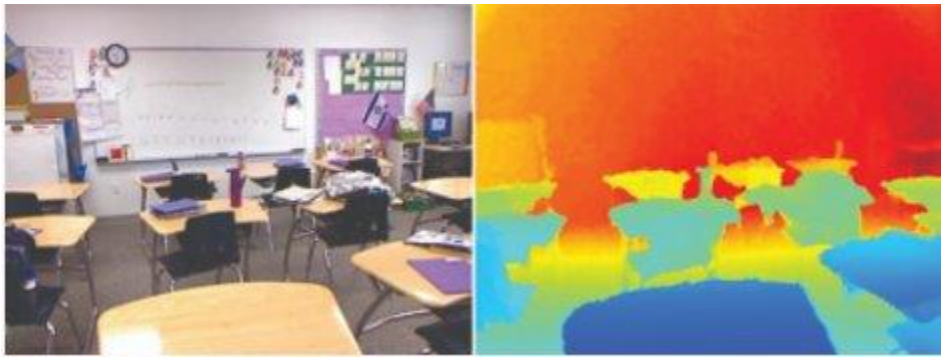

Figure 2 On the left we have RGB image of a classroom. On the right, we have Depth Image where red color represents depth farthest from the camera while blue is nearer.

The Datasets consist of indoor scenes like living room, bathroom, kitchen, classroom, libraries, etc.

DATA PRE-PROCESSING

Each image pair in the training data is encoded as a .h5 file, which contains both RGB image and corresponding depth map of size 640 x 480. We convert this data into respective NumPy arrays, apply a group of image transformation on both RGB

and depth maps, where we perform operations, such as resize, center crop to reduce the spatial size of the RGB images to 224 x 224 x 3 and depth maps to 224 x 224 x 1. Post this, the images are converted into tensors and are fed to the encoder architecture.

METHODOLOGY

In this section, we describe the architecture for our network followed in this project as well as justification for such choices.

NETWORK ARCHITECTURE

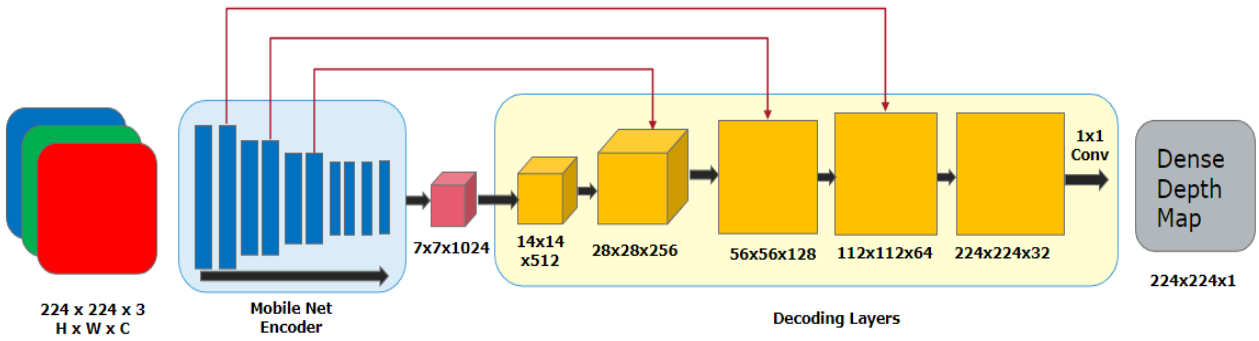

Figure 3 The encoder decoder architecture

Our network is a fully convolutional encoder-decoder architecture as shown in figure 3.

The encoder performs depth-wise separable convolutions and extracts features from the input image. The output of the MobileNet is fed to the decoder, where they are upsampled at each stage and finally give the output depth map. The feature maps extracted at each layer of the encoder are then fed to the corresponding decoder of appropriate feature map size using skip connections.

These feature maps from the encoders enable the decoder to get spatial information lost during the propagation and help in recovering fine-grained details. Finally, we use 1x1 convolution at the end to get the Dense Depth Map. The lightweight and low latency nature of the architecture helped us achieve 30 fps real-time dense depth map playback, which can be used in applications such as mobile robotics.

Encoder Network

To get low latency, we use a state-of-the-art efficient network, MobileNet (Andrew G. Howard, 2017), as our encoder of choice. MobileNet makes use of depth-wise decomposition, which are more efficient than normal convolutions. Below, we present the particulars of the encoder architecture, which captures the filter shape, stride and input size of each layer.

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$

Figure 4 MobileNet architecture (dw represents depthwise separable convolutions, s represents stride)

In Figure 3 and 4, a 224 x 224 x 3 tensor is being fed to the network, which gets processed by the network. As we the process the tensor through the layers of the encoder, the spatial size of the image is decreased while the representative power/depth/number of channels of feature map increases. Finally, after we process the tensor through all encoder layers, we reach the bottleneck, represented in red, where we get a 7 x 7 x 1024 feature map. The network performs average pooling before it feeds the output to the decoder architecture.

Depthwise Separable Convolutions

Depthwise separable convolutions (Chollet, 2017) help the encoder encode the data faster than traditional convolutions. Below, in figure 5, we present a pictorial representation to compare standard convolution and depthwise convolution.

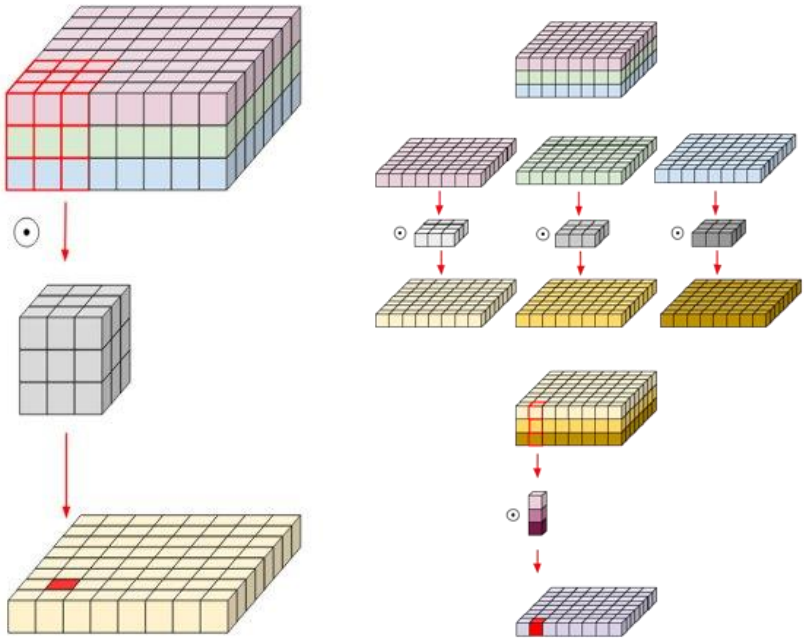

Figure 5(Left) Convolution and (Right) Depth-wise Separable Convolution

In figure 5, on the left, you can see the whole filter being convolved with the 3 channel RGB input to compute the convolved output. However, on the right, each channel of the 3-channel filter is being convolved individually with each channel of the input 3 channel image to get a 3 convolved feature maps, only to apply a point wise convolution to compute the final feature map. Both convolutions result in the

same answer but depthwise separable convolution is 100 times faster than traditional convolution.

Additionally, we present the math to show that depthwise separable convolutions is 100 times faster than traditional convolution. To do this, we compute number of multiplications (NOM) for both types of convolutions to compare them.

For standard convolution, the number of multiplications is as follows:

$$NOM_{standard\ convolution} = N \times D_p^2 \times D_k^2 \times M$$

Where,

> ***N** is a number of filters, **D_p** is HxW of the image, **D_K** is kernel, **M** Multiplication of single kernel*

Depth-wise Separable Convolution can be performed using N $D_p \times D_p$ depthwise layers and a 1×1 pointwise layer. Since each filter in a depth-wise layer only convolves with a single input channel, the complexity of a depth-wise layer is much lower than that of a standard convolutional layer. Hence, its depth-wise Separable Convolution is Depth-wise Convolution + Pointwise convolution.

For Depth-wise convolution, the number of multiplications is as follows:

$$NOM_{depthwise\ convolution} = M \times D_k^2 \times D_p^2$$

For Point-wise convolution, the number of multiplications is as follows:

$$NOM_{pointwise\ convolution} = M \times D_p^2 \times N$$

Through summation of NOM for both Depth-wise convolution and Point-wise convolution, we get

$$NOM_{depthwise\ seperable\ convolution} = M \times D_p^2 \times (D_k^2 + N)$$

Finally, we take ratio of NOM of standard convolution and depthwise separable convolutions.

$$Ratio(R) = \frac{NOM_{standard\ convolution}}{NOM_{depthwise\ seperable\ convolution}}$$

$$= \frac{1}{N} + \frac{1}{D_k^2}$$

$$R = 0.010004$$

This implies depthwise separable convolution is 100 times faster. This in essence helps us encode the image much faster, which helps in achieving faster frame rates.

Decoder Network

The objective of the decoder is and upsample the output of the encoder to form a dense prediction. An aspect of the decoder is the up-sample operation used with strided convolution/ fractional convolution/deconvolution.

Convolution is then by nearest-neighbor interpolation that doubles the number of spatial resolutions. This results in a fast decoder which outputs the decoder size. The decoder consists of five blocks each of the module having a strided convolution, upsampling followed by Batch Normalization and Relu activation.

The decoder increases the spatial resolution by increasing width and height. The input to the decoder from the bottleneck layer is 7x7x1024 which is then upsampled and convolved using strided convolution. This is upsampled through modules up to 224x224x32 on which we finally apply 1x1 convolution to get 224x224 x1 depth map output.

Skip Connections

Encoder networks like above contain many layers to gradually reduce the spatial resolution, where we extract low level features to complex features as we go deeper in the network. Skip connections here are long skip connections which allows us to get localization features which are lost to the convolutions and down sampling through various layers. Skip connections like these were previously used in architectures such as U-Net (Olaf Ronneberger, 2015), showing that they can be beneficial in networks producing dense outputs like ours. We include skip connections from the MobileNet encoder to the outputs of the middle three layers in the decoder.

JUSTIFICATION

In small robotics platforms, multiple programs (such as localization, mapping, control, and other potentially other perception tasks) all run in parallel. Each program demands a certain amount of computational resources, and consequently, the CPU is not dedicated to the depth estimation task. Hence, our goal was to implement a lightweight depth estimation network which can be used in applications with computational restraints and resource limitations. Hence, we used MobileNet, which is a fast encoder due to its inherent depthwise separable convolutions. The network has 3 million trainable parameters, which is 13 times less in comparison to other depth estimation architectures (Wonka). Despite this, it performs pretty well while not compromising much on performance.

EXPERIMENTATION

In figure 6, we show the training trials and plot the RMSE loss against epochs.

Epoch	Learning Rate	Loss	Batch Size
70	0.01	L1	8
70	0.01	L1	16
70	0.01	L1	32
70	0.01	L2	8
70	0.01	L2	16
70	0.01	L2	32
200	0.01	L1	256

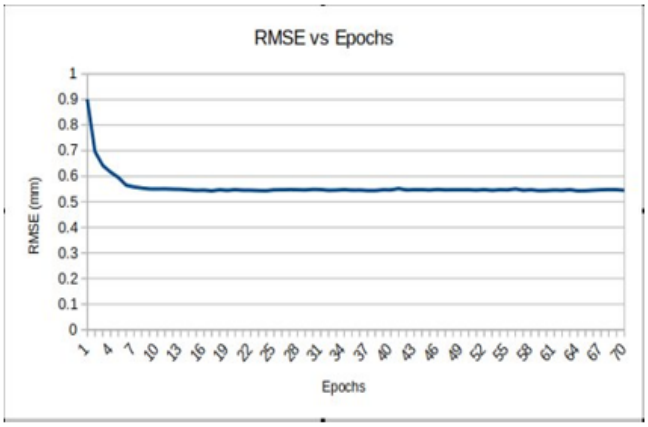

Figure 6 (Left) Experiments and (Right) RMSE vs Epochs

We train the network with multiple hyper-parameters such as learning rate, loss type and batch size. We try two different losses such as L1 loss (mean absolute

error) and L2 loss (mean square error), which penalizes the loss highly. We experiment using different batch sizes to see if our model generalizes better. We also trained our model with batch size 256 to gauge any shift in performance.

On the right, we plot the root mean square error (RMSE) against the number of epochs. We see that the RMSE error would not decrease beyond 0.55m and this is the best loss the model/architecture can achieve. Additionally, even though the model was trained for 70 epochs, the model's RMSE reached 0.55m by the 10^{th} epoch. Post that, the loss saturates and will not decrease any further.

RESULTS

In this section, firstly, we talk about the validation metrics we used to evaluate the performance of our model and present our metrics for each of our training trials. Further, we present the dense depth maps generated by the architecture and discuss our inferences and deductions.

MODEL VALIDATION

In figure 7, we present the root mean square error (RMSE) values and $\delta 1$ accuracy (the percentage of predicted pixels where the relative error is within 25%) for each of our training trails.

Epoch	Learning Rate	Loss	Batch Size	RMSE (meters)	δ1 accuracy
70	0.01	L1	8	0.66705	74.13
70	0.01	L1	16	0.57857	81.91
70	0.01	L1	32	0.54496	85.34
70	0.01	L2	8	0.77256	59.32
70	0.01	L2	16	0.60855	75.38
70	0.01	L2	32	0.52806	84.37
200	0.01	L1	128	0.596	82.08

Figure 7 RMSE (meters) and δ1 accuracy for all training trials

In Figure 7, our best model achieved 0.544 RMSE and 85.34% δ1 accuracy, which was trained for 70 epochs, with 0.01 learning rate, L1 (Mean Absolute Error) loss type and 32 batch size. To put this into perspective, RMSE encapsulates the fact that, while making predictions on unseen images, the model predicts a depth, which on average, is half a meter off from the target depth. This is impressive, considering that we only have 1 image to compute the dense depth and how lightweight the network is.

Additionally, with increase in batch size, we see a decrease in RMSE score and an increase in δ1 accuracy. This conforms with normal trend because with an increase in batch size, the model gets to see more data before updating parameters, which results in better generalization.

However, it is well known that too large of a batch size will lead to poor generalization and the model would overfit the data. This decrease in performance can be observed in the last training trail with batch size 128, where we see an increase in RMSE and decrease in accuracy (something we do not want!).

Smaller batch sizes allow the model to learn before seeing all the data. But the downside to this is the model might not converge to the global optima. Therefore, under no computational constraints, it is advised that one starts at a small batch size to reap the benefits of faster training and steadily grow the batch size, also reaping the benefits of faster convergence.

DEPTH PREDICTION ON INDOOR SCENES

In figure 8, we present the dense depth maps generated by our model on various indoor scenes.

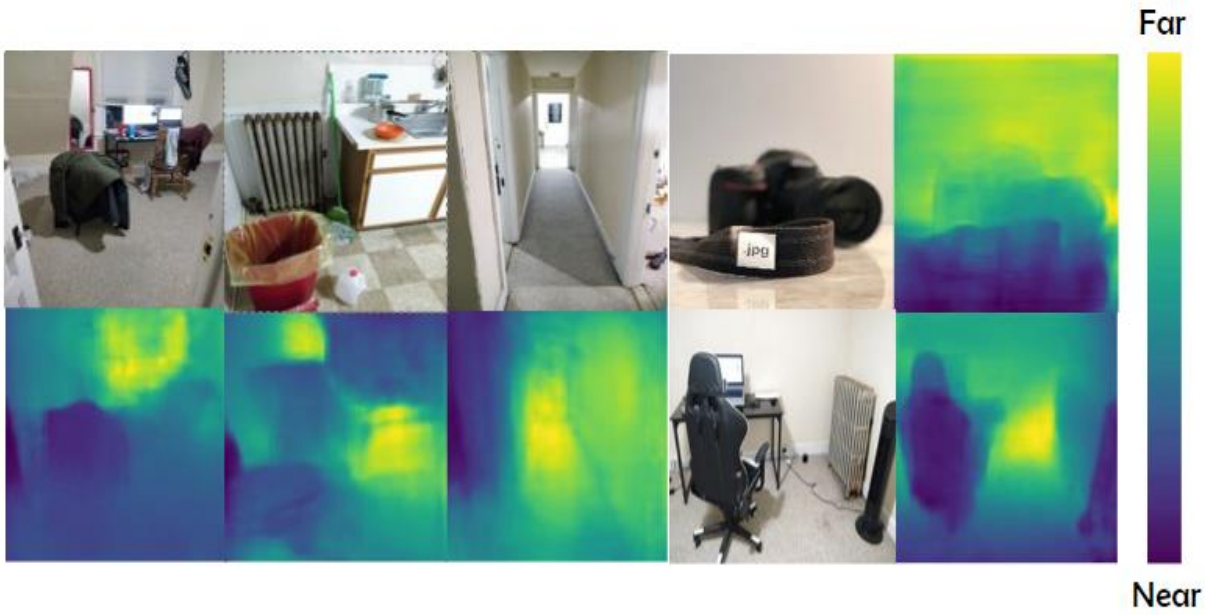

Figure 8 RGB images and their corresponding depth maps generated by the model

In the depth maps generated above in figure 8, the objects closest to the vantage point are represented in blue while the objects farthest from the vantage point are represented in yellow. The depth range between these extremes is represented by the colors on the Viridis color palette. It can also be seen that the generated depth maps are accurate. This is also due to additive skip connections from the layers of the encoder network to the layers of the decoder network, which allows image details from high resolution feature maps in the encoder to be merged into features within the decoder.

Additionally, the quicker computation of depth-wise convolutions in the encoder architecture results in a faster model, which on average takes 25 ms to generate a depth map on a single 2D image. To put this into context, the model can generate/play real-time depth information up to 33 FPS, to be precise. The following are the links to the source RGB video (captured from my phone camera) and the generated dense depth video: Source video, Generated depth map video

CONCLUSION

To conclude, we have presented a light-weight CNN based deep learning model to perform monocular depth estimation. We discussed the significance of depth information in real-world applications, and further dug into the advantages using monocular depth estimation. We also explained the network architecture and what features CNN looks for while computing depth. We trained our model on the NYU Depth V2 dataset, plotted the training loss and validated it based on RMSE and $\delta 1$ accuracy. We also presented the depth maps generated by the model and touched upon some of the advantages of the network.

Additionally, we would like to discuss the pros of the implementation. Depth wise separable convolution, performed in the encoder, which are 100 times faster normal convolutions, allows the network to perform real-time depth map estimation up to 30 fps. The Network is small, fast and efficient! It takes 25ms on average to estimate depth on a single RGB image/frame. Robotic applications with limitation of computational constraints can use this model while not having to compromise much on the depth accuracy. Skip connections help the decoding layers reconstruct more detailed dense depth output.

In the future, we will work towards achieving even higher fps real-time playback by either augmenting or pruning the network. The model currently has close to 3.5 million trainable parameters and this can certainly further prune down to achieve even faster frame rates while not sacrificing accuracy. Like everything in life, it's a balancing act! Moreover, we would also like to compare with other CNN based stereo methods and indoor datasets so we can see how our implementation compares with the state-of-the-art implementations.

REFERENCES

Andrew G. Howard, M. Z. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *CORR.*

Ashutosh Saxena, S. H. (2006). *Learning Depth from Single Monocular Images.* Stanford University.

Chollet, F. (2017). Xception: Deep Learning with Depthwise Separable Convolutions. *10.1109/CVPR.2017.195*, (pp. 1800-1807).

Dijk, T. v. (2019). How Do Neural Networks See Depth in Single Images?

Olaf Ronneberger, P. F. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. *DBLP:journals/corr/RonnebergerFB15*, *abs/1505.04597.* Retrieved from http://arxiv.org/abs/1505.04597

silberman. (n.d.). Retrieved from https://cs.nyu.edu/~silberman/datasets/nyu_depth_v2.html

Wonka, I. A. (n.d.). High Quality Monocular Depth Estimation via Transfer Learning. *CoRR.* Retrieved from http://arxiv.org/abs/1812.11941

APPENDIX

1. **PyTorch codebase:** https://github.com/Allamrahul/WPI_CV_Proj.git