Monocular Depth Estimation for Embedded and Mobile Platforms

Dhritiman Sagar
Stanford SCPD
dhrits@stanford.edu

Abstract

In recent years Deep Convolutional Neural Networks have been increasingly used to solve interesting problems in the field of computer vision. One such problem is Monocular Depth Estimation or estimating per-pixel depth given single images. Over the past few years, research papers like MegaDepth[23] and Monodepth[11] have produced extremely accurate depth maps from single images. However, the challenge with most of these approaches is that they're not optimized for mobile or embedded platforms. The FastDepth[33] paper from 2019 attempted to solve the problem of monocular depth estimation on mobile devices. This project attempts to recreate the FastDepth results, while improving upon them with a different architecture, loss function and training methodology.

1. Introduction

Depth estimation from images is a very important discipline in the field of computer vision and graphics. Recent years have seen a wide range of research which has approached the problem from different angles. The most commonly used and state-of-the-art methods rely on deep convolutional neural networks to solve this problem by training them on very large publicly available depth datasets[28][23][10][9]. The major problem with all these approaches is that they are extremely compute intensive. Most trained networks are greater than 20MB in size and have millions of floating point parameters. Thus, while they provide extremely accurate results, they're not suitable for real-time depth inference on commodity, embedded or mobile hardware.

Real-time, fast and power-efficient depth estimation has quickly become the holy-grail for many applications. In particular, mobile augmented reality applications, self-flying or self-driving automatons and even IOT-style home devices can greatly benefit from compute-efficient monocular depth estimation. While specialized sensors like LiDAR or other ToF hardware allow for precise short-term depth estimation, these sensors are prone to noise when used outdoors. Also, apart from being expensive, they are also not (yet) readily available on commodity phones or other embedded-hardware. Cameras, however, are ubiquitous on almost every phone and embedded platform. Unfortunately, most mobile devices still only have a single camera. As such, mobile, real-time monocular depth estimation is a very interesting area of research.

In 2019, the FastDepth[33] paper published at ICRA attempted to approach the problem of monocular depth estimation by inventing a mobile optimized-architecture which was further pruned using NetAdapt[34]. The network was trained on the NYU Depth V2[28] dataset and produced impressive results when tested on mobile/embedded hardware. The goal of this work is to reproduce the results of the FastDepth[33] paper while improving upon it by using a different architecture and training methodology. This work also tests the generalizability of the overall FastDepth network design methodology to other mobile architectures.

2. Related Work

2.1. Prior Work on Depth Estimation

The past few years have seen a wide range of approaches to the problem of monocular depth estimation. While traditional methods which use multiple cameras or stereo vision like SGBM[15] continue to be popular, they've also been enhanced with bespoke hardware like LiDAR and Time-of-Flight (TOF) sensors which help provide an accurate 3D point cloud. Despite the popularity of these methods, the challenge of obtaining the depth map of a scene given just a single image (monocular depth estimation) continues to be a very active area of research.

Recent years have seen research works which have trained deep convolutional neural networks (CNN) on massive publicly available or curated datasets like KITTI[10][9], NYU Depth V2[28], Cityscapes[4] and MegaDepth[23]. The MegaDepth[23] research paper from 2018 is particularly interesting, not just for training a highly accurate monocular depth estimation CNN, but also for curating a 200GB depth dataset of outdoor scenes collected using COLMAP[27]. Based on empirical usage in production systems, MegaDepth continues to be one of the most accurate and stable depth prediction networks. The only challenge with MegaDepth is that the large size of the network and the enormous dataset (199GB when compressed) makes it very hard to reproduce the results of MegaDepth on a budget.

There have also been extremely successful approaches like Monodepth[11] and MonodepthV2[12] which have relied on self-supervised learning using image-reconstruction loss to exploit the left-right consistency constraint of stereo images. These works are especially important because they've developed a methodology for using largely unlabeled (or partially labeled) datasets for developing very accurate depth maps. These papers makes use of easily-obtainable binocular stereo images to train their network. The use of unlabeled data potentially allows for collection and use of enormous quantities of data which can be used for network training.

Other recent approaches have relied on transfer learning[3] and a high performing pre-trained encoder based on DenseNet[20]. This paper showed that with a sufficiently accurate and pre-trained encoder, even a simple decoder architecture can lead to good results for depth estimation. Leveraging transfer learning on a pre-trained dataset, this work also helps reduce the reliance on very large labeled datasets.

Other recent works have also exploited commonly available dual-pixel hardware to estimate monocular depth[8].

The challenge with all of the works cited above is that none of them were trained for inference on low-compute budget platforms like mobile or embedded devices. The FastDepth[33] paper published in 2019 stands out here by attempting to create an efficient, small and fast network architecture which can easily perform inference on embedded platforms. As mentioned previously, this works takes inspiration from the FastDepth paper and attempts to recreate and then extend the original paper's approach.

2.2. Prior Work on Mobile-optimized CNN Architectures

In parallel, the deep-learning research community has also published several works optimizing CNN architectures for regression and classification on mobile and embedded devices. One of the first such architectures was SqueezeNet[21] which achieved the accuracy of AlexNet[22] with 50x fewer parameters. This was followed shortly thereafter by ENet[24], a CNN architecture optimized for real-time semantic segmentation.

Later, seminal works like MobileNet[17], MobileNetV2[26] and MobileNetV3[16] revolutionized mobile CNNs by using depthwise-separable convolutions and inverse-residual bottleneck layers which dramatically reduced the size of the underlying networks while achieving impressive classification accuracy on ImageNet[5] dataset. These works are especially important because they helped define a general methodology for shrinking existing architectures for mobile devices.

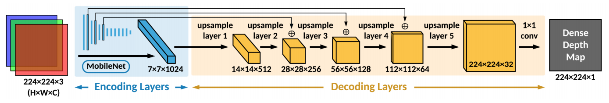

Figure 1. Architecture of the original FastDepth[33] architecture from the original paper

More recent results like EfficientNet[31] have come up with a very interesting way to develop a compound scale coefficient to scale the depth/width/resolution of a network all at one. The goal is to allow the network to maintain maximum accuracy with minimum number of operations. By tuning the EfficientNet scale factor, the network can easily trade off efficiency for accuracy.

The FastDepth[33] paper mentioned earlier (which forms the foundation of this work) makes heavy use of MobileNet inspired depthwise-separable convolutions. However, while the other architectures discussed so far focus on the encoder side of the architecture for prediction and classification embeddings, FastDepth goes a step further and experiments with optimal decoder architectures for mobile and embedded platforms. FastDepth's decoder architecture is called **NNConv5** which efficiently up-samples feature maps into full size depth maps. This decoder architecture is explored more in the Approach section of this report.

3. Approach

3.1. Background on FastDepth

As previously mentioned, the FastDepth[33] paper explored mobile-optimized architectures for monocular depth inference. This work aims to recreate the results of the FastDepth paper and improve upon the results using a different training methodology.

Figure 1 shows the FastDepth network architecture. At its core, the architecture is an encoder-decoder network which is optimized for mobile/embedded use. The overall architecture makes heavy use of depthwise-separable convolutions which are a hallmark of MobileNet family of architectures. If C_o is the number of output channels of a convolution and h and w are the height and width of the convolutional filter respectively, then use of a depthwise-separable convolution instead of a traditional convolution reduces the number of MAC operations by a factor of $(1/(C_o) + 1/(h * w))$[17].

The original paper did a very thorough job of benchmarking different combinations of encoder and decoder architectures. In the original ablation studies, a mobilenet encoder was found to provide the best balance of speed and accuracy for embedded use cases.

The paper also developed a novel decoder architecture called **NNConv5**[33] which consists of 5 sets of upsampling layers, each of which consists of a 5x5 convolution followed by a nearest-neighbor upsampling. In order to optimize the network for mobile use cases, the convolution operations consist only of depthwise-separable convolutions.

Despite this encoder-decoder architecture, the paper found that the network had difficulty preserving the more course details of the depthmap. To counter this, skip connections were added from the encoder to the upsampling layers. While both concatenation and addition operations for skip connections were benchmarked, addition was faster and resulted in fewer MAC (multiply-accumulate) operations. As such, the final FastDepth architecture made use of addition.

The final network illustrated in Figure 1 was trained on 33GB of NYU Depth V2[28] dataset using SGD with momentum optimizer and a learning rate of 0.01.

It is worth noting that the FastDepth authors also pruned their network using NetAdapt[34]. This further reduced the size of their network. However, given the time and monetary budget constraints of this project, this work does not attempt to try to prune and retrain the network. The FastDepth architecture shown in 1 forms the basis of exploration for this work.

3.2. Network Architecture

While starting with the base architecture described above, **this work made several changes to the network in order to improve network accuracy and generate more perceptually accurate depth maps**. Perhaps the biggest change to the network architecture was use of a MobileNetV2 backbone pretrained on ImageNet as the encoder of choice. The original paper made use of a pretrained MobileNet. There were several reasons for this choice. The original intuition was that MobileNetV2 improved upon MobileNet in terms of accuracy (while having comparable speed). As such, use of MobileNetV2 was worth exploring. It is also a well studied and deployed architecture which has generalized well to a large number of tasks such as segmentation and object detection (This is demonstrated in the original paper[26] by creating optimized networks for object-detection and segmentation). Studies described later in the Experiments section further justify this choice.

Table 1 describes the architecture of the encoder. The final average-pooling and 1×1 convolutional layers were removed from the MobileNetV2 architecture. These layers were initialized using weights from a MobileNetV2 model pre-trained on the ImageNet[5] dataset.

Note that the encoder decoder architecture used by FastDepth made use of skip connections from the encoder to the decoder. Thus, in order to make use of **additive** skip connections, the decoder architecture must depend on the output channels of the encoder. This would not be a problem if concatenative skip connections were used, but as the FastDepth paper demonstrated, concatenative skip-connections can be more expensive overall. As such, the architecture of the NNConv5 decoder was adjusted significantly in order to be compatible with a MobileNetV2 based encoder.

Table 2 broadly describes the architecture of the decoder. Following the NNConv5 pattern established in FastDepth, this decoder consists of 5 sets of upsampling layers. Each such layer is a depthwise-convolution+batchnorm+relu combination followed by a bilinear upsample which upsamples the feature by a factor of 2 at each pass. This structure efficiently upsamples the encodings from the preceding MobileNetV2 encoder into a full sized depth map. The base implementation of the architecture and layers were taken from

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1×1	-	1280	1	1

Table 1. The above table describes the architecture of the **encoder**. Each line describes a layer which is repeated n times. Each layer in a row has the same number of output channels c. The stride s is used for the first layer of each sequence followed by a stride of 1. The expansion factor t is applied to the input in the inverse-residual (bottleneck) layers.

.

Input	Operator	filter size	padding	outchannels
$7^2 \times 1024$	depthwise conv2d	5×5	2	512
$7^2 \times 512$	upsample bilinear	-	-	512
$14^2 \times 512$	depthwise conv2d	5×5	2	32
$14^2 \times 32$	upsample bilinear	-	-	32
$28^2 \times 32$	depthwise conv2d	5×5	2	24
$28^2 \times 24$	upsample bilinear	-	-	24
$56^2 \times 24$	depthwise conv2d	5×5	2	16
$56^2 \times 16$	upsample bilinear	-	-	16
$112^2 \times 16$	depthwise conv2d	5×5	2	32
$112^2 \times 32$	upsample bilinear	-	-	32
$224^2 \times 32$	pointwise conv2d	1×1	1	1

Table 2. The above table describes the architecture of the **NNConv5 decoder**. Each depthwise conv2d layer consists of a depthwise convolution followed by a batchnorm layer and a relu non-linearity. They have been grouped together in the table above for brevity.

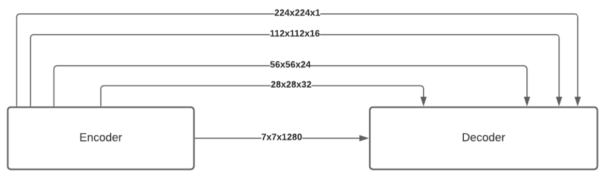

Figure 2. Figure shows the additive skip connection resolutions between the encoder and the decoder. From the encoder, the last layer with the appropriate resolution $h \times w \times c$ is used as the skip connection to the decoder layer with the same resolution.

the open-source FastDepth[32] code with several modifications to support MobileNetV2 and custom skip connections to the modified decoder layer.

As shown in figure 4, there are 4 sets of additive skip connections between the encoder and the decoder. In each case, the last repeated block with the appropriate resolution from the Mobilenetv2 encoder is used as a skip connection into the decoder.

The network encoder is initialized with weights pretrained on ImageNet[5] data. The decoder, on the other hand, is initialized using Kaiming Initialization[14].

3.3. Optimizer

The original FastDepth paper used SGD with momentum as its optimizer. For this work, both SGD with momentum and Adam were compared and SGD with momentum was eventually chosen because it provided qualitatively and quantitatively superior results during experimentation.

3.4. Loss Function

The primary loss function used for comparing ground truth depth data with predicted depth maps was Smoothed L1 Loss[1]. This loss function combines the best aspects of L1 and L2 losses and is more robust to outliers. The loss for pixels a and x in the prediction and ground truth can be defined as follows:

$$l(a,x) = \begin{cases} 0.5(a-x)^2/\beta, & \text{if } |a-x| < \beta \\ |a-x| - 0.5\beta, & \text{otherwise} \end{cases}$$

The original FastDepth paper and open-sourced code made[32][35] use of L1 (MAE) and L2 (MSE) losses respectively. During experiments though (see Experiments section), Smoothed L1 loss produced qualitatively (and perceptually) better results.

Merely using this simple loss though is not sufficient to generate depth maps with good perceptual quality. Simply using this loss causes the network to generate overly blurry/smoothed results. It is extremely important to maintain the perceptual quality of the depth map. In order to preserve object boundaries, an additional **edge preserving**

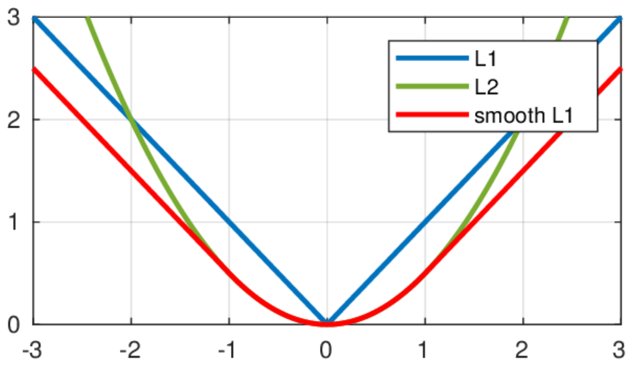

Figure 3. Comparison of L1, L2 and Smooth L1 loss. Figure from paper[7] published in 2018.

loss was added. This loss applies a differentiable sobel operator[6] to the ground truth depth and the prediction and then takes a smoothed L1 loss between the two resultant gradient images.

$$EdgeLoss(G,P) = SmoothedL1(Sobel(G), Sobel(P))$$

Where G and P are the ground truth and predicted depth maps respectively. The final loss function for the network is defined as:

$$L(G,P) = SmoothedL1(G,P) + \lambda EdgeLoss(G,P)$$

The lambda parameter was determined experimentally based on the magnitudes of the two losses. For the purposes of this work, $\lambda = 12$ was chosen.

3.5. Note on Learning Rate

The original FastDepth paper and open-source[32][35] implementations used a default learning rate of 0.01. However, for this work, I wanted to explore a more dynamic way to determine the learning rate. Following the general method outlined in Leslie N Smith's paper Cyclical Learning Rates for training Neural Networks[30] and implemented in both the FastAI[18] library and PyTorch LR-Finder[29], the learning rate is increased exponentially after each batch till the loss diverges or till we reach the maximum learning rate configured. Running this LR-Finder on the network and data helped tune the learning rate for the specific problem. For the final training, a learning rate of 0.08 was determined using LR Search. This learning rate was decayed by a multiplier of 0.9 every 10 epochs.

3.6. Data

FastDepth was trained on 33GB of the NYU Depth V2 dataset. In order to keep the scope of this work manageable and to maintain an apples-to-apples comparison, this

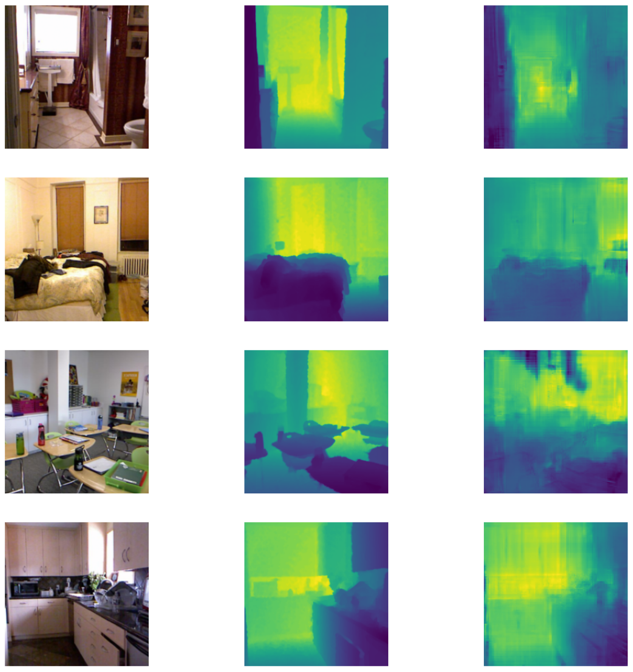

Figure 4. The above image shows predictions from the final network. From left to right a) The original image, b) the ground truth depth map and c) the predictions from the network.

work also relies on the very same dataset. The dataset, the preprocessing tools and the PyTorch[25] dataloaders were generously open-sourced by the FastDepth team in their official repository[32]. This work relied on the same data for training the final network architecture. However, as elaborated upon in the Experiments section, all comparative studies and ablations were carried out on a subset of the NYU Depth V2 Dataset to save on time and compute resources.

3.7. Training Statistics

The final network was trained on 33GB of the NYU Depth V2 dataset on a NVIDIA P6000 GPU for 100 epochs. **The training took a week to complete and achieved a $\delta 1$ score of 0.82. $\delta 1$ score is the percentage of pixels in the predicted depth map which differ by less than 25% from the ground truth. To my knowledge the FastDepth network achieved a $\delta 1$ score of 0.77 on the full dataset**.

The experimental methodology that led to this final network, training techniques and hyperparameters, is explained in the next section.

4. Experiments

4.1. Experimental Methodology

In order to achieve the final network, hyperparameters and training methodology listed in the previous section, it was important to explore the network architecture and hyperparameter space. Doing so with the entire NYU Depth V2 dataset would have been prohibitively expensive and would have taken much more time than is available for the purposes of this project. As such, it was important to develop an experimentation methodology which would rely on a subset of the data. For this purpose, the NYU Depth V2 Labeled Dataset[28] was chosen for hyperparameter tuning. This is a collection of 1400 images along with their ground-truth depth maps. Each combination or architecture/hyperparameter was iterated on for 50 epochs and evaluated on the basis of the final loss and $\delta 1$ score achieved. In the interest of keeping this report manageable, I have attempted to summarize the most important results in this section.

4.2. Comparing MobileNet to MobileNetV2

One of the first experiments performed, was the comparison of MobileNet to MobileNetV2. As mentioned previously, the original FastDepth[33] paper made use of a MobileNet encoder combined with an NNConv5 decoder with additive skip connections. In the paper's analysis this architecture led to the best balance of accuracy, model size and inference speed. The authors of the paper generously open-sourced their code making a comparison to their original model simpler.

Replacing MobileNet with MobileNetV2 for the encoder also required re-engineering the decoder layer to be able to handle additive skip connections from the encoder layer (please see previous section for architecture details). Initial comparison between MobileNet and MobileNetV2 on the NYU Depth V2 Labeled Dataset did not result in much of a difference in qualitative or quantitative results when trained for 50 epochs. The MobileNet variant achieved a $\delta 1$ score of 0.743 while the MobileNetV2 variant achieved a $\delta 1$ score of 0.740. Similarly while the MobileNet variant achieved a loss of 0.40, the MobileNetV2 variant achieved a loss of 0.41. While these results show the MobileNet variant to be slightly better, the results are close enough to be due to random noise (especially given the random initialization of weights using Kaiming Initialization[14] and the random shuffling of the training dataloader).

4.3. Impact of LR Search

The real difference in results emerged once LR Search[30][29] was introduced to find the optimal learning rate. Before training of the network, an LR-Search was run for both the MobileNet and MobileNetV2 variants while continuing to use the labeled subset data. This is where the MobileNetV2 variant (with optimized learning rate) saw a significant improvement. The MobileNetV2 variant achieved a $\delta 1$ score of 0.815 and a loss of 0.31 compared to the MobileNet variant's score of 0.724 and loss of 0.53 respectively. Thus, at least when evaluating this particular dataset, it was clear that MobileNetV2 based archi-

Network	$\delta 1$	MSE	s/it
MobileNetV2	0.740	0.40	1.76
MobileNet	0.743	0.41	1.76
MobileNetV2(searched lr=0.03)	**0.815**	0.31	1.76
MobileNet(searched lr=0.09)	0.724	0.53	1.76
MobileNetV2+Adam(searched lr=0.016)	0.485	1.165	1.76
MobileNet+Adam(searched lr=0.01)	0.612	0.926	1.76

Table 3. Results of training MobileNet and MobileNetV2 with fixed LR of 0.01 and the same results when training MobileNet and MobileNetV2 with LR-Search based learning rate. Training was also attempted using the Adam optimizer instead of SGD
.

tecture combined with LR-Search based learning rate performed the best (Please note though that since then, different runs of the experiment have presented slightly different results. **In general, the performance of MobileNet and MobileNetV2 variants is comparable**).

Table 3 shows the quantitative results of the training runs described earlier. It is worth noting that the MobileNet based architecture with LR-Search seems to perform worse than when trained with a fixed learning rate of 0.01. This is a curious result and appears to be an anomaly. I would not expect there to be too much of a difference between MobileNet and MobileNetV2 when combined with LR-Search. This is something I wish to explore in follow-up work. Another curious result is the poor performance of the default Adam optimizer ($\beta 1 = 0.9$ and $\beta 2 = 0.999$). Despite repeated trials, the default Adam configuration performed much worse than SGD on the subset dataset.

Given the quantitative results and since the goal of this work was to extend and test the generalizability of the FastDepth approach, I chose to go ahead with the custom MobileNetV2 based architecture (along with the decoder modified to handle skip connections).

4.4. Impact of Edge-Loss

One thing which is immediately clear in images from Figure 5 is that all the networks end up producing blurry/overly-smoothed depth maps. Ideally, the trained network will be able to preserve the object boundaries. In order to retain object edges, an edge preserving loss was introduced. The details of this loss are described in the earlier section on Approach. The overall idea was to use a differentiable Sobel operator[6] on both the ground-truth depth maps and the predictions to get the corresponding gradient images. A scaled version of this Smoothed L1 loss between these two images is additionally added to the existing loss. By looking at the magnitude of different loss terms, a multiplier of 12 was chosen to scale the edge loss. The output depth maps from networks trained with this combined loss showed an immediate improvement in the perceptual quality of the produced depth maps. Figure 8 demonstrates the impact on the depth maps after use of edge/sobel loss. Since other losses are on a different scale, a quantitative comparison wouldn't make much sense here.

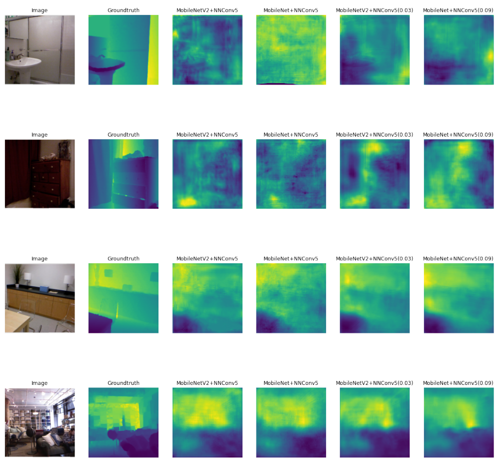

Figure 5. Comparisons of depth maps produced after 50 iterations. From left to right a) Raw image, b) ground truth depth, c) MobileNetV2+NNConv5 d) MobileNet + NNConv5, e)MobileNetV2+NNConv5(0.03) and f) MobileNet+NNConv5(0.09)

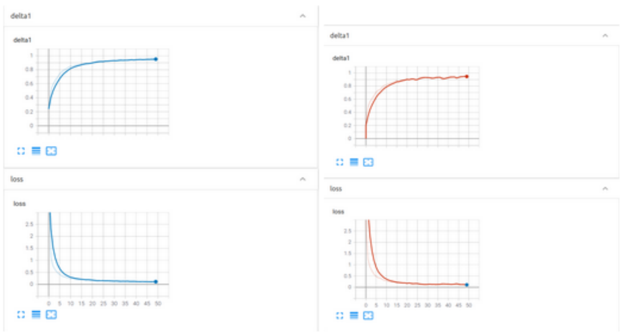

Figure 6. $\delta 1$ and loss plots of MobileNetV2+NNConv5 (left) and MobileNet+NNConv5 (right)

4.5. Impact of batch size

For final training of the network on the entire dataset, two different networks were trained with different batch sizes. Table 4 shows the impact of training the networks on bath sizes of 64 and 128. It is clear that the batch size of 128 produced far superior results compared to the batch size of 16. While a larger batch size will help the network be less prone to noise and converge faster, I didn't expect the dif-

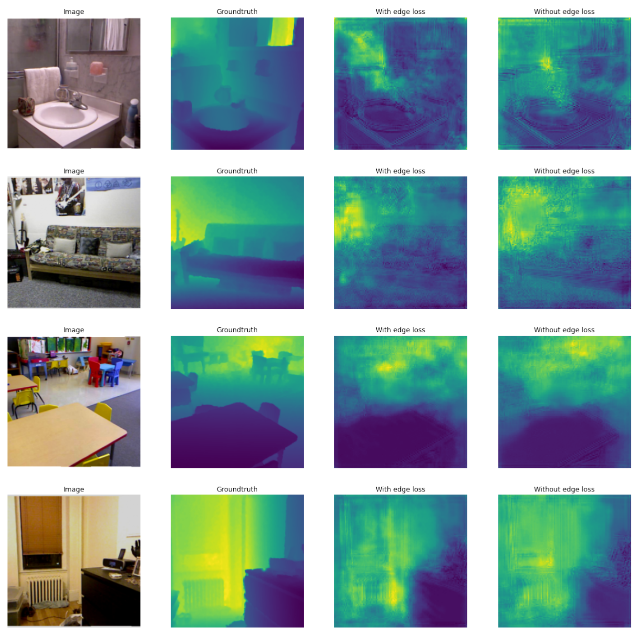

Figure 7. Images, ground truth depth maps and predictions when trained with and without Sobel Loss

Batch Size	$\delta1$ Score
128	**0.82**
64	0.61

Table 4. The table above shows impact of two different batch sizes used to train the network on the full dataset. It is clear that the larger batch-size of 128 leads to far superior results compared to a batch size of 16

.

ference to be quite so profound. A follow-up investigation could help determine whether more epochs would help the smaller batch size eventually achieve the same level of accuracy.

4.6. Deployment

The final model trained on 33GB of NYU Depth V2 data had a model size of 11.8 MB. Compressed to float16, the model has a size of 5.9 MB. In order to test the model on mobile devices, I made use of Snap Lens Studio[2], a studio editor for creating and deploying small AR applications on Snapchat app's community AR ecosystem. Lens Studio has a tool called SnapML which allows building of applications visually using trained ONNX models which can be dragged and dropped into Lens Studio. The model achieved a fairly stable 30 FPS on an iPhone XR. In theory, the model could run even faster, but a lot of compute power was spent visualizing the model as a normalized grayscale texture using a shader. It is also worth noting that this exercise made me realize that iOS CoreML NPU does not (yet) support relu6 layers which are used quite heavily in MobileNetV2.

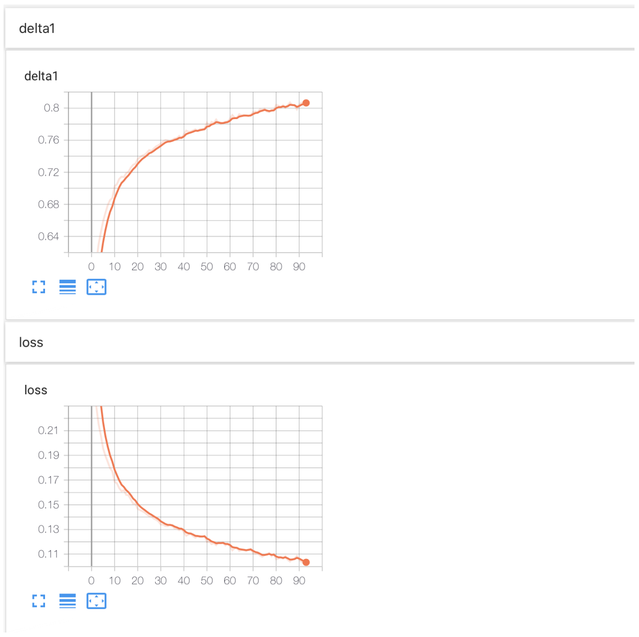

Figure 8. Final $delta1$ and loss curves for the network trained for 100 epochs on the full dataset

Replacing relu6 layers with relu, or using MobileNet will allow for the use of iOS's NPU.

While the results look promising on a high end iPhone, commodity Android phones like the LG LMK-300 can only achieve (at best) 5-10 FPS. On such devices, it is important to not try to run the depth inference on every frame (as is the case in this lens).

Figure 10 shows a Snapcode of the deployed publicly available community lens. If the reader has the Snapchat app installed, they may open it and scan the snapcode (press and hold on the Snapchat camera screen) to unlock the lens and try it. Videos and links to the project are available in the supplementary section of this report.

5. Conclusion

Overall, FastDepth[33] has proven to be a simple, easily extendable and generalizable architecture for mobile monocular depth estimation. By applying hyperparameter tuning methods like LR-Search and applying custom loss functions, the network performance can be improved further. The overall network design philosophy can also be extended to other mobile architectures like MobileNetV2 while adapting the NNConv5 decoder architecture to support additive skip connections from the customized encoder layers. The trained model is easy to deploy on mobile devices and can run in real-time.

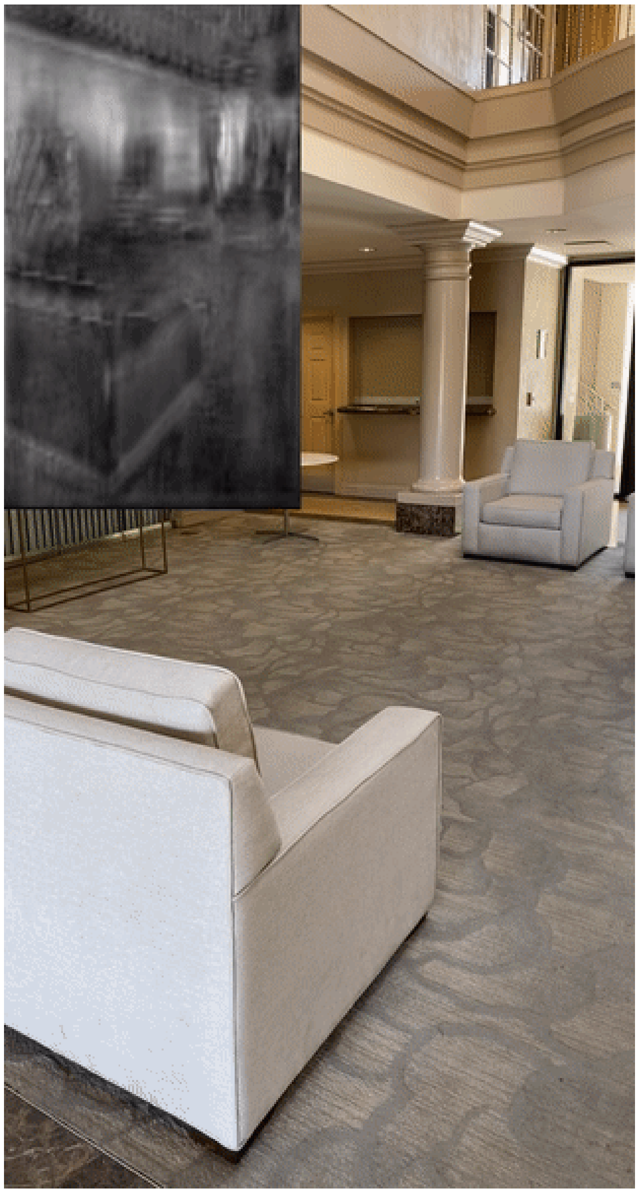

Figure 9. Screengrab of running lens deployed to an iPhone

5.1. Future Work

While this work succeeded in recreating the FastDepth paper results and extending them using different network architectures, hyperparameters and training methodologies, the time constraints for this project did not allow for all the thorough exploration and experimentation required to deploy a production-grade mobile depth estimation model. Some ideas for future work are listed below:

- **Use of more diverse datasets** - While the NYU Depth V2[28] is a great dataset for benchmarking monocular depth algorithms, it is limited in that it contains only images of indoor household scenes. A truly generalizable production quality model should incorporate outdoor images which are critical for AR, robotics and autonomous system applications. Combining NYU Depth V2 dataset with outdoor datasets like MegaDepth[23], KITTI[10][9] and Cityscapes[4] can theoretically lead to very interesting results. In particular, I did attempt to explore the MegaDepth dataset for this project but timing, budgetary and compute resource constraints did not allow me to make progress on this front.

- **Exploring different mobile encoders** - As mentioned earlier in this report, there are a wide range of mobile optimized CNN architectures. It is possible to extend the FastDepth approach to each of them to benchmark results. In particular, EfficientNet with its compound scaling coefficient could lead to interesting trade-offs between accuracy, model-size and speed.

- **Use of smaller encoders** - Depth estimation does not require semantic reasoning about the scene. As such, it is possible to consider even smaller encoder architectures like ENet[24] and Squeezenet[21] for inference.

- **Use of larger network for perceptual loss** - It may be possible to improve the results of the network by making use of a much larger network like Resnet[13] or DenseNet[20] for computing perceptual loss between groundtruth and predicted depthmaps. Unfortunately, given my limited compute budget, attempting such a training structure severely limited the batch-size I could train with to 2-4. It may still be possible to

Figure 10. Snapcode of publicly available community lens

make progress given more computational resources or by using a small batch-size coupled with gradient accumulation.

- **Exploring self-supervised approaches** - As shown by Monodepth[11] and MonodepthV2[12] papers, it is possible to use readily available stereo data to train monocular depth networks in a self-supervised manner. It should be possible to explore this approach using the same mobile optimized network used for this project.

5.2. Acknowledgements

This project would not have been possible without the contributions of the original FastDepth[33] paper which thoroughly explored and benchmarked different architectures for monocular depth estimation on low-compute budgets. I would like sincerely thank the authors for their contributions and for open-sourcing[32] their code, curated datasets and PyTorch dataloaders for use by other researchers and students. The open sourced code allowed for clear comparisons and extensions which would have otherwise not been possible.

I would also like to express my appreciation to the authors of the FastAI[19][18] course and library for their approach to a callback-customizable deep-learning training infrastructure which was heavily used by this project for iterating on models and hyperparameters. Other open-source libraries like PyTorch LR-Finder[29] and Kornia[6] also helped tune hyperparameters and develop new loss functions easily. I am also grateful for Snapchat Lens-Studio team which provided an easily accessible environment for deploying mobile CNN models along with thorough documentation and example code. This report also borrowed ideas from all works cited in the biliography.

Finally, I would like to thank the course staff of CS231a Winter 2021 quarter for allowing me to explore this problem for my course project and providing advice and guidance along the way.

6. Supplementary Material

- Github Repository - Contains source code and README for training and evaluation code used to train the network.

- Videos of model running on iPhone - Contains recordings of the ONNX model running on an iPhone after being deployed as a community lens on Snapchat.

- Model checkpoints and ONNX exports - This links contains the intermediate training checkpoints stored as .pth files as well as two models exported in ONNX format.

- Tensorboard.dev link to graphs of $\delta 1$ score and loss changing during hundred epochs of training on the full dataset.

- Simple Lens Studio Depth Visualization Project - A link to a simple lens studio project which can visualize indoor depth. The github repository linked above has instructions on how to use Lens Studio.

References

[1] Smooth l1 loss. `https://pytorch.org/docs/stable/generated/torch.nn.SmoothL1Loss.html`.

[2] Snapchat Lens Studio. `https://lensstudio.snapchat.com/`.

[3] I. Alhashim and P. Wonka. High quality monocular depth estimation via transfer learning. *arXiv e-prints*, abs/1812.11941, 2018.

[4] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3213–3223, 2016.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[6] D. P. E. R. E. Riba, D. Mishkin and G. Bradski. Kornia: an open source differentiable computer vision library for pytorch. In *Winter Conference on Applications of Computer Vision*, 2020.

[7] Z. Feng, J. Kittler, M. Awais, P. Huber, and X. Wu. Wing loss for robust facial landmark localisation with convolutional neural networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2235–2245, 2018.

[8] R. Garg, N. Wadhwa, S. Ansari, and J. T. Barron. Learning single camera depth estimation using dual-pixels.

[9] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[10] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[11] C. Godard, O. M. Aodha, and G. J. Brostow. Unsupervised monocular depth estimation with left-right consistency.

[12] C. Godard, O. Mac Aodha, M. Firman, and G. J. Brostow. Digging into self-supervised monocular depth prediction. October 2019.

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition.

[14] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.

[15] H. Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):328–341, 2008.

[16] A. Howard, M. Sandler, B. Chen, W. Wang, L. Chen, M. Tan, G. Chu, V. Vasudevan, Y. Zhu, R. Pang, H. Adam, and Q. Le. Searching for mobilenetv3. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1314–1324, 2019.

[17] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications.

[18] J. Howard et al. fastai. `https://github.com/fastai/fastai`, 2018.

[19] J. P. Howard and S. Gugger. Fastai course v3: Deep learning from foundations. `https://github.com/fastai/course-v3`, 2019.

[20] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

[21] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5mb model size.

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[23] Z. Li and N. Snavely. Megadepth: Learning single-view depth prediction from internet photos. In *Computer Vision and Pattern Recognition (CVPR)*, 2018.

[24] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. 06 2016.

[25] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[26] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks.

[27] J. L. Schönberger and J.-M. Frahm. Structure-from-Motion Revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[28] N. Silberman, P. Kohli, D. Hoiem, and R. Fergus. Indoor segmentation and support inference from rgbd images. *ECCV*, 2012.

[29] D. Silva. Pytorch lr finder. `https://github.com/davidtvs/pytorch-lr-finder`, 2020.

[30] L. N. Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472, 2017.

[31] M. Tan and Q. Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 09–15 Jun 2019.

[32] D. Wofk, F. Ma, T.-J. Yang, S. Karaman, and V. Sze. Fastdepth architectures and datasets. `https://github.com/dwofk/fast-depth`, 2019.

[33] Wofk, Diana and Ma, Fangchang and Yang, Tien-Ju and Karaman, Sertac and Sze, Vivienne. FastDepth: Fast Monocular Depth Estimation on Embedded Systems. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[34] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *The European Conference on Computer Vision (ECCV)*, September 2018.

[35] S. Yehuda, G. Rafalovich, and D. Sriker. Fastdepth. `https://github.com/tau-adl/FastDepth/`, 2020.