FastDepth: Fast Monocular Depth Estimation on Embedded Systems

Diana Wofk*, Fangchang Ma*, Tien-Ju Yang, Sertac Karaman, Vivienne Sze

***Abstract*— Depth sensing is a critical function for robotic tasks such as localization, mapping and obstacle detection. There has been a significant and growing interest in depth estimation from a single RGB image, due to the relatively low cost and size of monocular cameras. However, state-of-the-art single-view depth estimation algorithms are based on fairly complex deep neural networks that are too slow for real-time inference on an embedded platform, for instance, mounted on a micro aerial vehicle. In this paper, we address the problem of fast depth estimation on embedded systems. We propose an efficient and lightweight encoder-decoder network architecture and apply network pruning to further reduce computational complexity and latency. In particular, we focus on the design of a low-latency decoder. Our methodology demonstrates that it is possible to achieve similar accuracy as prior work on depth estimation, but at inference speeds that are an order of magnitude faster. Our proposed network, FastDepth, runs at 178 fps on an NVIDIA Jetson TX2 GPU and at 27 fps when using only the TX2 CPU, with active power consumption under 10 W. FastDepth achieves close to state-of-the-art accuracy on the NYU Depth v2 dataset. To the best of the authors' knowledge, this paper demonstrates real-time monocular depth estimation using a deep neural network with the lowest latency and highest throughput on an embedded platform that can be carried by a micro aerial vehicle.**

I. Introduction

Depth sensing is essential to many robotic tasks, including mapping, localization, and obstacle avoidance. Existing depth sensors (e.g., LiDARs, structured-light sensors, etc.) are typically bulky, heavy, and have high power consumption. These limitations make them unsuitable for small robotic platforms (e.g., micro aerial and mini ground vehicles), which motivates depth estimation using a monocular camera, due to its low cost, compact size, and high energy efficiency.

Past research on monocular depth estimation has focused almost exclusively on improving accuracy, resulting in computation-intensive algorithms that cannot be readily adopted in robotic systems. Since most systems are not only limited in compute resources, but are also subject to latency constraints, a key challenge is balancing the computation and runtime cost with the accuracy of the algorithm.

Current state-of-the-art depth estimation algorithms rely on deep learning based methods, and while these achieve significant improvement in accuracy, they do so at the cost of increased computational complexity. Prior research on designing fast and efficient networks has primarily focused on encoder networks for tasks such as image classification and object detection [1]. In these applications, the input is an image (pixel-based), and the output is reduced to a label

Authors are with the Massachusetts Institute of Technology, Cambridge, MA 02139, USA. Emails: {dwofk, fcma, tjy, sertac, sze}@mit.edu.
Project website: http://fastdepth.mit.edu

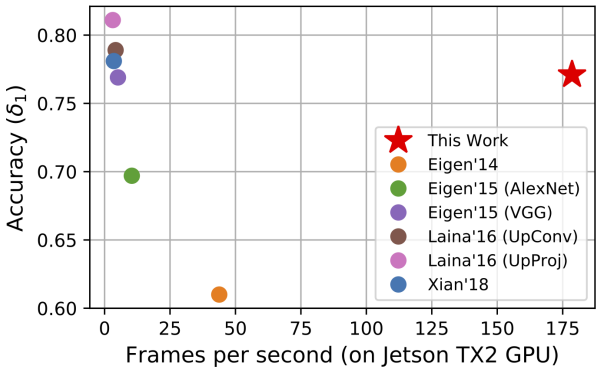


Fig. 1: Accuracy vs. runtime (in fps) on an NVIDIA Jetson TX2 GPU for various depth estimation algorithms. Top right represents the desired characteristics of a depth estimation network design: high throughput and high accuracy.

(an object class and position). To the best of our knowledge, little effort has been put into the efficient design of *both* encoder and decoder networks (i.e., auto-encoder networks) for tasks such as depth estimation, where the output is a dense image. In particular, reducing decoder complexity poses an additional challenge since there is less information reduction at each of the decoding layers and the decoder's output is high dimensional.

To address these challenges, this paper presents a low latency, high-throughput, high-accuracy depth estimation algorithm running on embedded systems. We propose an efficient encoder-decoder network architecture with a focus on low latency design. Our approach employs MobileNet [2] as an encoder and nearest neighbor interpolation with depthwise separable convolution in the decoder. We apply state-of-the-art network pruning, NetAdapt [3], and use the TVM compiler stack [4] to further reduce inference runtime on a target embedded platform. We show that our low latency network design, FastDepth, can perform real-time depth estimation on the NVIDIA Jetson TX2 [5], operating at over 120 frames per second (fps) on the TX2 GPU (see Figure 1) and at over 25 fps on the TX2 CPU only[1], with active power consumption under 10 W. The attained throughput is an order of magnitude higher than prior work on depth estimation, with only a slight loss of accuracy; FastDepth achieves a δ_1 accuracy[2] of 77.1% on NYU Depth v2.

The low latency and high throughput attainable with our network is motivated by practical robotic systems, where

[1]This throughput is achieved with a batch size of one and 32-bit floating point precision. Throughput can be increased by using a larger batch size (at the cost of higher latency), and/or reducing bitwidths through quantization.

[2]Accuracy metrics are defined in Section IV-A.

multiple programs (such as localization, mapping, motion planning, control, and potentially other perception tasks) all run in parallel. Each program demands a certain amount of computational resources, and consequently, the CPU/GPU is not dedicated to the depth estimation task alone. A lower latency network design enables real-time performance even with a limited computing budget.

In summary, this paper demonstrates real-time monocular depth estimation using a deep neural network that achieves the lowest latency and highest throughput on an embedded platform that can be carried by a micro aerial vehicle.

II. Related Work

In this section, we summarize past research done on depth estimation, efficient neural networks, and network pruning.

A. Monocular Depth Estimation

Depth estimation from a single color image has been an active research topic in both the robotics and computer vision communities for over a decade. Early works on depth estimation using RGB images captured by a monocular camera usually relied on hand-crafted features and probabilistic graphical models. For instance, Saxena et al. [6] estimated the absolute scales of different image patches and inferred depth using a Markov Random Field model. Non-parametric approaches [7–10] were also exploited to estimate the depth of a query image by combining the depths of images with similar photometric content retrieved from a database. Since then, depth estimation has evolved from using simple hand-crafted feature representations [6] to modern deep learning based approaches [11–14].

State-of-the-art RGB-based depth estimation methods use deep learning based methods to train a convolution neural network using large-scale datasets [12, 15, 16]. Eigen et al. [11] suggested a two-stack convolutional neural network (CNN), with one stack predicting the global coarse scale and the other stack refining local details. Eigen and Fergus [16] further incorporated other auxiliary prediction tasks into the same architecture. Liu et al. [15] combined a deep CNN and a continuous conditional random field, and attained visually sharper transitions and local details. Laina et al. [12] developed a deep residual network based on ResNet [17] and achieved higher accuracy than [15, 16]. Qi et al. [18] trained networks to estimate both the depth and the normals, as a way to address the problem of blurriness in predictions. Semi-supervised [19] and unsupervised learning [20–22] setups have also been explored for disparity image prediction. For instance, Godard et al. [22] formulated disparity estimation as an image reconstruction problem, where neural networks were trained to warp left images to match the right. Mancini et al. [23] proposed a CNN that took both RGB images and optical flow images as input to predict distance. Fusion of RGB images and sparse depth measurements [24, 25] at early stages also improved the accuracy of depth estimation.

All of these methods focus heavily on attaining higher accuracy at increased complexity and runtime cost, with diminishing accuracy improvement. For instance, the δ_1 accuracy of depth estimation on the NYU Depth V2 dataset [26] saturates at around 82% in recent years [12, 18].

B. Efficient Neural Networks

There has been significant effort in prior work to design efficient neural networks. As an example, for image classification, MobileNet [2] achieves similar accuracy as VGG-16 [27] but has 2.7 times fewer multiply-and-accumulate operations (MACs) and 32.9 times fewer weights. For object detection, SSD [28] is 6.6 times faster than Faster-RCNN [29] with a higher mean average precision (mAP).

However, most previous work in this space has focused on encoder networks that reduce an input image into a label. Designing efficient neural networks for applications requiring pixel-based results, where an encoder network is followed by a decoder network, has been less explored. As will be shown in Figure 3, in existing designs that achieve close to state-of-the-art accuracy on the depth estimation task, the decoder largely dominates inference runtime. In our work, we emphasize efficient encoder-decoder network design. In particular, our usage of depthwise separable convolution in the decoder differentiates us from existing approaches and enables us to develop an architecture in which the decoder no longer dominates inference runtime.

C. Network Pruning

Hand-crafted networks are usually over-parameterized due to the ease of training, which reduces network efficiency. To address this problem, network pruning, such as [3, 30–32], is widely used to identify and remove redundant parameters and computation. However, these pruning methods are mainly applied on encoder networks. In this work, we adopt a state-of-the-art algorithm, NetAdapt [3], to demonstrate how pruning can improve the efficiency of *both* encoder and decoder networks used in a depth estimation network design.

III. Methodology

In this section, we describe our proposed network architecture, the motivation behind our design choices, and the steps we take to reduce inference runtime.

A. Network Architecture

Our proposed fully convolutional encoder-decoder architecture is shown in Figure 2. The encoder extracts high-level low-resolution features from the input image. These features are then fed into the decoder, where they are gradually upsampled, refined, and merged to form the final high-resolution output depth map. In developing a depth estimation network that can run in real-time, we seek low-latency designs for both the encoder and the decoder.

1) Encoder Network: The encoder used in depth estimation networks is commonly a network designed for image classification. Popular choices include VGG-16 [27] and ResNet-50 [17] because of their strong expressive power and high accuracy. However, such networks also suffer from high complexity and latency, making them unsuitable for applications running in real-time on embedded systems.

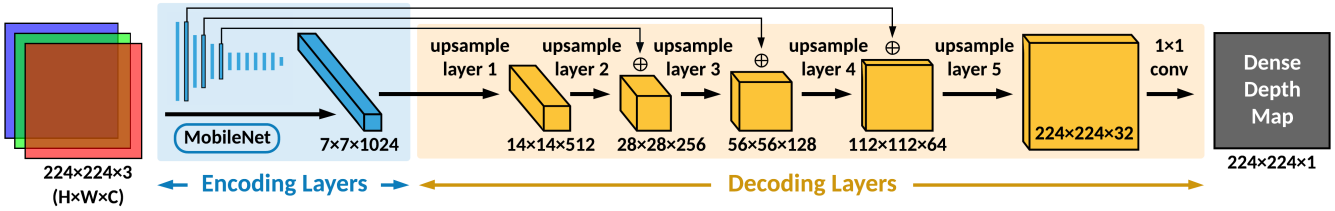


Fig. 2: Proposed network architecture. Dimensions of intermediate feature maps are given as height × width × # channels. Arrows from encoding layers to decoding layers denote additive (rather than concatenative) skip connections.

Targeting low latency, we employ a state-of-the-art efficient network, MobileNet [2], as our encoder of choice. MobileNet makes use of depthwise decomposition, which factorizes an $m \times m \times n$ standard convolutional layer into n $m \times m$ depthwise layers and a 1×1 pointwise layer. Since each filter in a depthwise layer only convolves with a *single* input channel, the complexity of a depthwise layer is much lower than that of a standard convolutional layer, where each filter convolves with *all* input channels. Moreover, each pointwise filter is just a 1×1 kernel, so the number of MACs performed by a pointwise layer is m^2 times smaller than that of the original standard convolution. Therefore, depthwise decomposition significantly reduces the complexity of a convolutional layer, making MobileNet more efficient than networks with standard convolution like ResNet and VGG. This translates to reduced latency,

2) Decoder Network: The objective of the decoder is to merge and upsample the output of the encoder to form a dense prediction. A key design aspect of the decoder is the upsample operation used (e.g., unpooling, transpose convolution, interpolation combined with convolution).

Our decoder network (termed NNConv5) consists of five cascading upsample layers and a single pointwise layer at the end. Each upsample layer performs 5×5 convolution and reduces the number of output channels by $1/2$ relative to the number of input channels. Convolution is followed by nearest-neighbor interpolation that doubles the spatial resolution of intermediate feature maps. Interpolating *after* convolution instead of before lowers the resolution of feature maps processed by the convolutional layers. We use depthwise decomposition to further lower the complexity of all convolutional layers, resulting in a slim and fast decoder.

3) Skip Connections: Encoder networks typically contain many layers to gradually reduce the spatial resolution and extract higher-level features from the input. The output of the encoder into the decoder becomes a set of low resolution features in which many image details can be lost, making it more difficult for the decoder to recover pixel-wise (dense) data. Skip connections allow image details from high resolution feature maps in the encoder to be merged into features within the decoder; this helps the decoding layers reconstruct a more detailed dense output. Skip connections have been previously been used in networks for image segmentation such as U-Net [33] and DeeperLab [34], showing that they can be beneficial in networks producing dense outputs.

We incorporate skip connections from the MobileNet encoder to the outputs of the middle three layers in the decoder. Feature maps are combined with via addition rather than concatenation, to avoid increasing the number of feature map channels processed by the decoding layers.

B. Network Compilation

Our proposed network architecture is fully convolutional and makes use of depthwise decomposition in both the encoder and the decoder. Depthwise separable convolutional layers are currently not yet fully optimized for fast runtime in commonly-used deep learning frameworks. This motivates the need for hardware-specific compilation to translate the complexity reduction achievable with depthwise layers into runtime reduction on hardware. We use the TVM compiler stack [4] to compile our proposed network design for deployment on embedded platforms such as the Jetson TX2.

C. Network Pruning

To reduce network latency even further, we perform post-training network pruning using the state-of-the-art algorithm, NetAdapt [3]. Starting from a trained network, NetAdapt automatically and iteratively identifies and removes redundant channels from the feature maps to reduce the computational complexity. In each iteration, NetAdapt generates a set of network proposals simplified from a reference network. The network proposal with the best accuracy-complexity trade-off is then chosen and used as the reference network in the next iteration. The process continues until the target accuracy or complexity is achieved. Network complexity can be gauged by indirect metrics (e.g., MACs) or direct metrics (e.g., latency on a target hardware platform).

IV. Experiments

In this section, we present experiment results to demonstrate our approach. We first present an evaluation against existing work and then provide an ablation study of our design. We offer comparisons of various encoder and decoder options, analysing them based on accuracy and latency metrics. We also show that hardware-specific compilation reduces the runtime cost of the depthwise separable layers within our network and that network pruning helps improve the efficiency of both of the encoder and the decoder.

A. Experiment Setup

We train our networks and evaluate their accuracy on the NYU Depth v2 dataset [26] with the official train/test data split. Network training is similar to [24] and is implemented

in PyTorch [35] with 32-bit floating point precision. For training, a batch size of 8 and a learning rate of 0.01 are used. The optimizer is SGD with a momentum of 0.9 and a weight decay of 0.0001. Encoder weights (e.g., for MobileNet and ResNet) are pretrained on ImageNet [36]. Accuracy is measured by both δ_1 (the percentage of predicted pixels where the relative error is within 25%) and RMSE (root mean squared error). For evaluation, a batch size of 1 is used. Precision is kept at 32-bit floating point. Our primary target platform is a Jetson TX2 in max-N mode.[3]

B. Final Results and Comparison With Prior Work

Results achieved with our methodology are summarized in Figure 3. ResNet-50 with UpProj serves as a baseline; this network follows the architecture described in [12].[4] The runtime of this baseline network is largely dominated by the decoder. In our approach, the most immediate and significant runtime reduction comes from using a smaller and computationally simpler decoder. However, the decoder continues to dominate network runtime when combined with the MobileNet encoder. By simplifying the convolutions within the decoder with depthwise decomposition, the runtime of the decoder begins to more closely match that of MobileNet. Pruning and compiling the network for the target Jetson TX2 platform lower the runtime of both the encoder and the decoder even further, ultimately reducing total inference runtime by a factor of 65 times relative to the baseline and enabling an extremely high throughput of up to 178 fps.

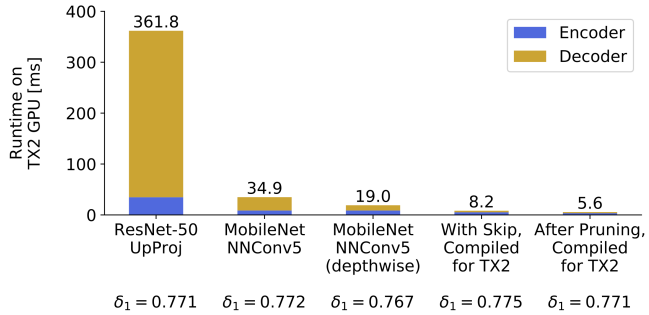


Fig. 3: Reduction in inference runtime achieved with our approach. Stacked bars represent encoder-decoder breakdown; total runtimes are listed above the bars. The row of δ_1 accuracies listed at the bottom shows the impact of individual steps in our approach on accuracy. Relative to ResNet-50 with UpProj, our final model achieves 65 times speedup while maintaining accuracy.

Accuracy and latency metrics of our model in comparison with prior work[5] are summarized in Table I. We evaluate against depth estimation methods that use deep learning but do not involve additional processing such as CRFs, since the additional processing incurs computation and runtime cost.

[3]Max-N mode: all CPU cores in use and GPU clocked at 1.3 GHz.

[4]With modifications, namely a 224×224 input to the encoder and five (instead of four) upsample layers in the decoder. This is done to match our architecture and allow for a more fair comparison to our network. Consequently, the accuracy and runtime reported in Figure 3 will differ from that of the unmodified model reported as part of our evaluation in Table I.

[5]MACs and runtimes were generated from our re-implemented models.

TABLE I: Comparison with prior work. For δ_1, higher is better. For all others, lower is better. Runtimes are measured on a Jetson TX2. Our network design achieves close to state-of-the-art accuracy and is an order of magnitude faster.

on NYU Depth v2	Input Size	MACs [G]	RMSE	δ_1	CPU [ms]	GPU [ms]
Eigen *et al.* [11]	228×304	2.06	0.907	0.611	307	23
Eigen *et al.* [16] (AlexNet)	228×304	8.39	0.753	0.697	1391	96
Eigen *et al.* [16] (VGG)	228×304	23.4	0.641	0.769	2797	195
Laina *et al.* [12] (UpConv)	228×304	22.9	0.604	0.789	2384	237
Laina *et al.* [12] (UpProj)	228×304	42.7	**0.573**	**0.811**	3928	319
Xian *et al.* [37]	384×384	61.8	0.660	0.781	4429	283
This Work	224×224	**0.37**	0.604	0.771	**37**	**5.6**

We measure the active power consumption when running our model on the TX2 in max-N mode to be under 10 W. We additionally report runtime and power consumption data for the TX2 in the more energy-efficient max-Q mode.[6] Table II summarizes this data. We note that with the TX2 in max-Q mode, our model can achieve close to real-time speeds on the CPU and can still easily surpass real-time speeds on the GPU, with active power consumption under 5 W.

TABLE II: Inference runtime and peak power consumption when deploying our model on the Jetson TX2 in high performance (max-N) and high energy-efficiency (max-Q) modes. Active power consumption can be estimated by subtracting the reported idle power consumption.

Platform	Runtime	Max Frame Rate	Power Consumption
TX2 GPU (max-N)	5.6 ms	178 fps	12.2 W (3.4 W idle)
TX2 GPU (max-Q)	8.2 ms	120 fps	6.5 W (1.9 W idle)
TX2 CPU (max-N)	37 ms	27 fps	10.5 W (3.4 W idle)
TX2 CPU (max-Q)	64 ms	15 fps	3.8 W (1.9 W idle)

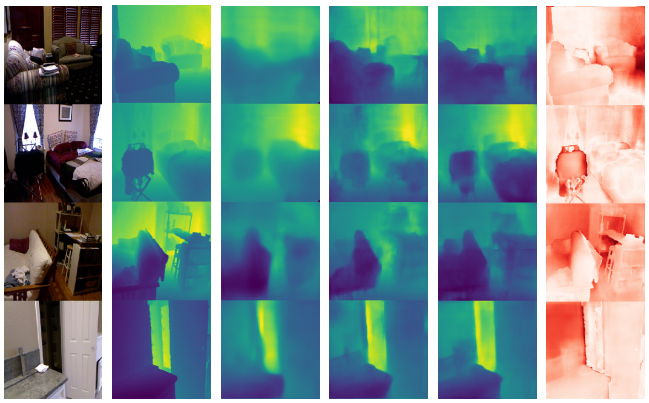
Figure 4 visualizes results of depth estimation produced by our model on images from the NYU Depth v2 dataset. Skip connections between encoding and decoding layers improve the sharpness and visual clarity of depth map outputs, while network pruning preserves and even enhances clarity. We also show an error map visualizing the difference between the output of our model and ground truth, noting that the error is highest at boundaries and at distant objects.

C. Ablation Study: Encoder Design Space

A common encoder used in existing high-accuracy approaches [12, 37] is ResNet-50 [17]. Targeting lower encoder latency, we consider the smaller ResNet-18 [17] and MobileNet [2] as alternatives to ResNet-50. The last average pooling layer and fully connected layers are removed from the MobileNet and ResNet architectures. To make the encoders compatible with a fixed decoder structure, we append a 1×1 convolutional layer to the end of ResNet encoders, such that the output from all encoder variants has a consistent shape of 7×7 with 1024 channels.

We compare all three encoder options against each other in Table III. The reported runtimes are obtained by compiling and running the encoder networks in PyTorch. Runtimes for ResNet-50 and ResNet-18 are too high, even on the

[6]Max-Q mode: only the ARM Cortex-A57 cores in use and GPU clocked at 0.85 GHz. Configured for best power-throughput tradeoff.



(a) (b) (c) (d) (e) (f)

Fig. 4: Visualized results of depth estimation on the NYU Depth v2 dataset. (a) input RGB image; (b) ground truth; (c) our model, without skip connections, unpruned; (d) our model, with skip connections, unpruned; (e) our model, with skip connections, pruned; (f) error map between the output of our final pruned model and ground truth, where redder regions indicate higher error.

TX2 GPU, to achieve real-time speeds (i.e., above 30 fps) if these encoders are paired with decoders of similar latency. In comparison, MobileNet efficiently trades off between accuracy and latency, and has a noticeably lower GPU runtime. We therefore select MobileNet as our encoder.

We note that despite its lower complexity, MobileNet is an order of magnitude slower on the TX2 CPU than ResNet-18. This can be attributed to as-of-yet unoptimized implementations for depthwise layers in deep learning frameworks, motivating the need for an alternate deep learning compiler, as will be discussed in Section IV-E.

TABLE III: Comparison of encoders. RMSE and δ_1 are for encoder-decoder networks with the decoder fixed as NNConv5. All other metrics are for the encoder in isolation. Runtimes are measured on a TX2. MobileNet is selected as best encoder option.

Encoder	Weights [M]	MACs [G]	RMSE [meters]	δ_1	CPU [ms]	GPU [ms]
ResNet-50	25.6	4.19	**0.568**	**0.800**	610	35.0
ResNet-18	11.7	1.84	**0.568**	0.782	**220**	15.2
MobileNet	**3.19**	**0.57**	0.579	0.772	3700	**8.7**

D. Ablation Study: Decoder Design Space

While encoders have been well characterized in deep learning applications, decoders have been less extensively explored, especially in the context of efficient network design. We consider several decoder design aspects: upsample operation, depthwise decomposition, and skip connections.

1) Upsample Operation: We survey four ways of upsampling in the decoder. Their characteristics are listed below, and visual representations are shown in Figure 5:

(a) UpProj [12]: 2×2 unpooling (zero-insertion) followed by a two-branched residual structure that computes a total of three convolutions (two 5×5 and one 3×3).

(b) UpConv [12]: 2×2 unpooling (zero-insertion) followed by a single 5×5 convolution.

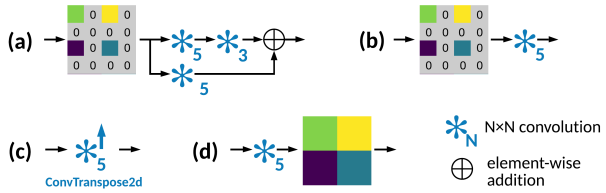


Fig. 5: Visual representations of different upsample operations. (a) UpProj, (b) UpConv, (c) DeConv5, (d) NNConv5.

(c) DeConv5: transpose convolution using a 5×5 kernel.[7]

(d) NNConv5: 5×5 convolution followed by nearest-neighbor interpolation[8] with a scale factor of 2.

We implement four decoder variants using these upsample operations, keeping the structure fixed at 5 decoding layers with 1×1 convolution at the end. Table IV compares the four decoders. UpProj is most complex, due to its larger number of convolutions per upsample layer. It achieves the highest δ_1 accuracy but is the slowest. UpConv is less complex and faster than UpProj, but its CPU and GPU runtimes are still too slow for real-time processing. DeConv5 has an identical number of weights and MACs as UpConv and is noticeably faster on both the CPU and GPU. However, it can be prone to introducing checkerboard artifacts in its outputs [38], which helps explain its lower accuracy. NNConv5 achieves higher δ_1 accuracy and lower RMSE than both UpConv and DeConv5, with a slightly lower GPU runtime. We therefore select NNConv5 as our decoder.

TABLE IV: Comparison of decoders. RMSE and δ_1 are for encoder-decoder networks with a MobileNet encoder. All other metrics are for the decoder in isolation. Runtimes are measured on a TX2. NNConv5 is selected as best decoder option.

Decoder	Weights [M]	MACs [G]	RMSE [meters]	δ_1	CPU [ms]	GPU [ms]
(a) UpProj [12]	38.1	28.0	0.599	**0.774**	3300	325
(b) UpConv [12]	**17.5**	12.9	0.591	0.771	1600	238
(c) DeConv5	**17.5**	12.9	0.596	0.766	**290**	31.0
(d) NNConv5	**17.5**	**3.21**	**0.579**	0.772	410	**26.2**

2) Depthwise Separable Convolution: After selecting MobileNet as our encoder and NNConv5 as our decoder, we observe that the runtime of our network is dominated by the decoder (see Figure 3 for the encoder-decoder breakdown). This motivates us to simplify our decoder even further. Similar to how depthwise decomposition lowers the complexity and latency in MobileNet, we now replace all convolutions within the decoder with depthwise separable convolutions.

Table V shows that depthwise decomposition in the decoder lowers inference runtime on the GPU by almost half.[9] However, as was the case with MobileNet, depthwise layers in the decoder result in a slight accuracy loss, due to the

[7] We use a kernel size of 5 to fairly compare against UpConv.

[8] An alternate option would be bilinear interpolation. We select nearest-neighbor interpolation as is it a simpler operation with more consistent implementations across deep learning frameworks and compilers.

[9] In contrast, runtime on the CPU increases, despite reduced number of MACs. This is due to the inefficient CPU operation of depthwise layers.

reduction in trainable parameters and computation. In order to restore some of the lost accuracy, we incorporate skip connections between the encoding and decoding layers.

TABLE V: Impact of depthwise decoding layers and skip connections on network complexity and TX2 runtime.

MobileNet-NNConv5	Weights [M]	MACs [G]	RMSE [meters]	δ_1	CPU [ms]	GPU [ms]
with standard decoder	20.6	3.78	**0.579**	0.772	**4100**	34.9
with depthwise decoder	**3.93**	**0.74**	0.584	0.767	5200	**18.6**
depthwise & skip-concat	3.99	0.85	0.601	**0.776**	5500	26.8
depthwise & skip-add	**3.93**	**0.74**	0.599	0.775	5100	19.1

3) Skip Connections: We consider both additive and concatenative skip connections. Concatenative skip connections increase the computational complexity of the decoder since decoding layers need to process feature maps with more channels. Table V shows that this improves the δ_1 accuracy but also noticeably increases CPU and GPU runtimes. In contrast, using additive skip connections leaves the number of channels in the decoder unchanged and has a negligible impact on inference runtime while achieving almost the same accuracy boost. We therefore use additive skip connections in our final network design. As shown in Figure 4(d), skip connections noticeably improve the sharpness and visual clarity of the depth maps output by our network design.

E. Hardware-Specific Optimization

Current deep learning frameworks rely on framework-specific operator libraries, where the level of hardware-specific optimization of operator implementations may vary. Our proposed network architecture incorporates depthwise layers throughout the encoder and decoder. These layers are currently not yet fully optimized in commonly-used deep learning frameworks. As a result, although depthwise decomposition significantly reduces the number of MACs in a network, a similar reduction is not reflected in latency. The left portion of Table VI highlights exactly this: the TX2 CPU runtime of MobileNet-NNConv5 is high to begin with, due to the prevalence of depthwise layers in MobileNet, and it increases even more when we use depthwise layers in the decoder. To address the observed runtime inefficiencies of depthwise layers, we use the TVM compiler stack [4]. TVM performs hardware-specific scheduling and operator tuning that allows the impact of reduced operations to be translated into reduced processing time. The right portion of Table VI reports TX2 runtimes for networks compiled with TVM. Depthwise decomposition in the decoder now reduces CPU runtime by 3.5 times and GPU runtime by 2.5 times.

F. Network Pruning

Prior to network pruning, our architecute (MobileNet-NNConv5 with depthwise decomposition in the decoder and additive skip connections) already surpasses real-time throughput on the TX2 GPU but does not yet achieve real-time speeds on the TX2 CPU. Network pruning lowers the model's runtime so that it can achieve a CPU framerate above 25 fps that is more suitable for real-time inference. As

TABLE VI: Hardware-specific compilation enables fast depthwise layers in our network. Runtimes are measured on the TX2.

MobileNet-NNConv5	in PyTorch		using TVM	
	CPU [ms]	GPU [ms]	CPU [ms]	GPU [ms]
with standard decoder	**4100**	34.9	176	20.9
with depthwise decoder	5200	**18.6**	**50**	8.3
with depthwise & skip-add	5100	19.1	66	**8.2**

shown in Table VII, pruning achieves a 2 times reduction in MACs, a 1.5 times reduction in GPU runtime, and a 1.8 times reduction in GPU runtime with almost the same accuracy. Figure 4(e) shows that pruning process preserves the sharpness and visual clarity of output depth maps.

Fig. 6 shows the pruned architecture. We can see that there are two bottlenecks: one in the encoder (the layer mobilenet.9) and one in the decoder (the layer decoder.2). This is consistent with the observations in [3, 39].

TABLE VII: Impact of pruning on our encoder-decoder network. Runtimes are measured post-compilation for the TX2.

	Before Pruning	After Pruning	Reduction
Weights	3.93M	1.34M	2.9×
MACs	0.74G	0.37G	2.0×
RMSE	0.599	0.604	-
δ_1	0.775	0.771	-
CPU [ms]	66	37	1.8×
GPU [ms]	8.2	5.6	1.5×

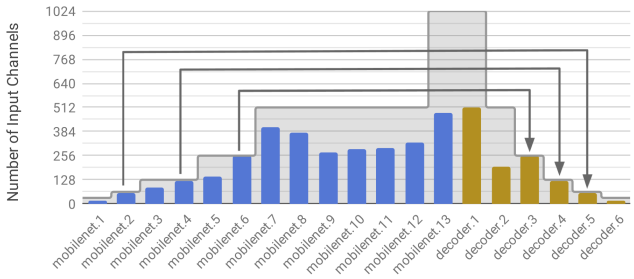


Fig. 6: Number of input channels to each layer in our network architecture after pruning. The shaded part represents the architecture before pruning. The very first layer to the network (mobilenet.0) is not shown since the channel size of the input fed into the network remains fixed at 3 channels (RGB).

V. Conclusion

In this work, we enable high-speed depth estimation on embedded systems. We achieve high frame rates by developing an efficient network architecture, with a low-complexity and low-latency decoder design that does not dominate inference runtime even when combined with a small MobileNet encoder. The size of our compact model is further reduced by applying a state-of-the-art pruning algorithm. Hardware-specific compilation is used to translate complexity reduction into lower runtime on a target platform. On the Jetson TX2, our final model achieves runtimes that are an order of magnitude faster than prior work, while maintaining comparable accuracy.

Although this work focuses on depth estimation, we believe that similar approaches can be used to achieve real-time performance with deep-learning based methods for other dense prediction tasks, such as image segmentation.

REFERENCES

[1] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[2] 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," *arXiv preprint arXiv:1704.04861*, 2017.

[3] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sadler, V. Sze, and H. Adam, "NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications," in *European Conference on Computer Vision (ECCV)*, 2018.

[4] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze *et al.*, "TVM: An Automated End-to-End Optimizing Compiler for Deep Learning," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 578–594.

[5] "Jetson TX2 Module," https://developer.nvidia.com/embedded/buy/jetson-tx2, accessed: 2018-09-15.

[6] A. Saxena, S. H. Chung, and A. Y. Ng, "Learning depth from single monocular images," in *Advances in Neural Information Processing Systems (NIPS)*, 2006, pp. 1161–1168.

[7] K. Karsch, C. Liu, and S. B. Kang, "Depth extraction from video using non-parametric sampling," in *European Conference on Computer Vision (ECCV)*, 2012, pp. 775–788.

[8] J. Konrad, M. Wang, and P. Ishwar, "2d-to-3d image conversion by learning depth from examples," in *Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2012, pp. 16–22.

[9] K. Karsch, C. Liu, and S. Kang, "DepthTransfer: Depth Extraction from Video Using Non-parametric Sampling," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 99, pp. 1–1, 2014.

[10] M. Liu, M. Salzmann, and X. He, "Discrete-continuous depth estimation from a single image," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 716–723.

[11] D. Eigen, C. Puhrsch, and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," in *Advances in Neural Information Processing Systems (NIPS)*, 2014, pp. 2366–2374.

[12] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab, "Deeper depth prediction with fully convolutional residual networks," in *International Conference on 3D Vision (3DV)*, 2016, pp. 239–248.

[13] B. Ummenhofer, H. Zhou, J. Uhrig, N. Mayer, E. Ilg, A. Dosovitskiy, and T. Brox, "DeMoN: Depth and motion network for learning monocular stereo," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[14] H. Fu, M. Gong, C. Wang, K. Batmanghelich, and D. Tao, "Deep Ordinal Regression Network for Monocular Depth Estimation," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 2002–2011.

[15] F. Liu, C. Shen, and G. Lin, "Deep convolutional neural fields for depth estimation from a single image," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 5162–5170.

[16] D. Eigen and R. Fergus, "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture," in *International Conference on Computer Vision (ICCV)*, 2015, pp. 2650–2658.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[18] X. Qi, R. Liao, Z. Liu, R. Urtasun, and J. Jia, "GeoNet: Geometric Neural Network for Joint Depth and Surface Normal Estimation," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 283–291.

[19] Y. Kuznietsov, J. Stückler, and B. Leibe, "Semi-supervised deep learning for monocular depth map prediction," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6647–6655.

[20] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, "Unsupervised learning of depth and ego-motion from video," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[21] R. Garg, G. Carneiro, and I. Reid, "Unsupervised CNN for single view depth estimation: Geometry to the rescue," in *European Conference on Computer Vision (ECCV)*, 2016, pp. 740–756.

[22] C. Godard, O. Mac Aodha, and G. J. Brostow, "Unsupervised monocular depth estimation with left-right consistency," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[23] M. Mancini, G. Costante, P. Valigi, and T. A. Ciarfuglia, "Fast robust monocular depth estimation for obstacle detection with fully convolutional networks," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 4296–4303.

[24] F. Ma and S. Karaman, "Sparse-to-dense: depth prediction from sparse depth samples and a single image," *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[25] F. Ma, G. V. Cavalheiro, and S. Karaman, "Self-supervised sparse-to-dense: Self-supervised depth completion from lidar and monocular camera," *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[26] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, "Indoor segmentation and support inference from RGBD images," in *European Conference on Computer Vision (ECCV)*, 2012, pp. 746–760.

[27] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition,"

International Conference on Learning Representations (ICLR), 2015.

[28] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European Conference on Computer Vision (ECCV)*, 2016.

[29] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[30] Y. Le Cun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems (NIPS)*, 1990.

[31] A. Gordon, E. Eban, O. Nachum, B. Chen, T.-J. Yang, and E. Choi, "Morphnet: Fast & simple resource-constrained structure learning of deep networks," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[32] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[33] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[34] T.-J. Yang, M. D. Collins, Y. Zhu, J.-J. Hwang, T. Liu, X. Zhang, V. Sze, G. Papandreou, and L.-C. Chen, "DeeperLab: Single-Shot Image Parser," *arXiv preprint arXiv:1902.05093*, 2019.

[35] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NIPS 2017 Workshop Autodiff*, 2017.

[36] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 248–255.

[37] K. Xian, C. Shen, Z. Cao, H. Lu, Y. Xiao, R. Li, and Z. Luo, "Monocular relative depth perception with web stereo data supervision," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 311–320.

[38] A. Odena, V. Dumoulin, and C. Olah, "Deconvolution and checkerboard artifacts," *Distill*, 2016. [Online]. Available: http://distill.pub/2016/deconv-checkerboard

[39] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 4510–4520.