IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier ACCESS.2020.DOI

RRNet: Repetition-Reduction Network for Energy Efficient Depth Estimation

SANGYUN OH[1,2]*, (Student Member, IEEE), HYE-JIN S. KIM[3,4]*, (Student Member, IEEE), JONGEUN LEE[1,2], (Member, IEEE), and JUNMO KIM[3], (Member, IEEE)

[1]School of Electrical and Computer Engineering, UNIST, Ulsan, Korea
[2]Neural Processing Research Center, Seoul National University, Seoul, Korea
[3]Robotics Program, School of Electrical Engineering, KAIST, Daejeon, Korea
[4]Artificial Intelligence Laboratory, ETRI, Daejeon, Korea

Corresponding author: Junmo Kim (e-mail: junmo.kim@kaist.ac.kr).

The two authors* contributed equally to this paper.
This material is based upon work supported by the Ministry of Trade, Industry Energy (MOTIE), Korea, under the i-Ceramic platform construction project- i-Ceramic manufacturing innovation platform No. 20004367, "Development of Cloud Big Data Platform for the Innovative Manufacturing in Ceramic Industry" and is supported by NRF grants funded by the MSIT of Korea (No. 2017R1D1A1B03033591, No. 2020R1A2C2015066), the IITP grant funded by the MSIT of Korea (No. 1711080972), and the Free Innovative Research Fund of UNIST (1.170067.01).

ABSTRACT Lightweight neural networks that employ depthwise convolution have a significant computational advantage over those that use standard convolution because they involve fewer parameters; however, they also require more time, even with graphics processing units (GPUs). We propose a Repetition-Reduction Network (RRNet) in which the number of depthwise channels is large enough to reduce computation time while simultaneously being small enough to reduce GPU latency. RRNet also reduces power consumption and memory usage, not only in the encoder but also in the residual connections to the decoder. We apply RRNet to the problem of resource-constrained depth estimation, where it proves to be significantly more efficient than other methods in terms of energy consumption, memory usage, and computation. It has two key modules: the Repetition-Reduction (RR) block, which is a set of repeated lightweight convolutions that can be used for feature extraction in the encoder, and the Condensed Decoding Connection (CDC), which can replace the skip connection, delivering features to the decoder while significantly reducing the channel depth of the decoder layers. Experimental results on the KITTI dataset show that RRNet consumes 3.84× less energy and 3.06× less memory than conventional schemes, and that it is 2.21× faster on a commercial mobile GPU without increasing the demand on hardware resources relative to the baseline network. Furthermore, RRNet outperforms state-of-the-art lightweight models such as MobileNets, PyDNet, DiCENet, DABNet, and EfficientNet.

INDEX TERMS Computer vision, Deep neural network, Depth estimation, Encoder–decoder network, Lightweight neural network, Machine learning, Mobile Graphical Processing Unit (GPU), Unsupervised learning.

I. INTRODUCTION

Depth estimation is crucial for several computer vision applications. Many technological goals, including localization in augmented reality (AR) or virtual reality (VR), advanced robotics, the reliable operation of autonomous vehicles or drones, and smart factories, cannot be realized without accurate depth estimation. Furthermore, deep learning approaches [1]–[9] convincingly outperform attempts to manually solve this problem [10], [11]. Nevertheless, these approaches involve resource-intensive computation. Consequently, their use in mobile applications that involve a lightweight neural network model and relatively low-end graphics processing units (GPUs) remains limited. Moreover, in most cases, they make use of the Compute Unified Device Architecture (CUDA) parallel computing platform [12] and the related neural-network library cuDNN [13]. As we will show in the subsequent sections of this paper, this can strongly affect performance.

The most intuitive method of designing a lightweight model is to employ light layers with small sized kernels and to suitably scale the number of channel parameters using appropriate sub-sampling. However, performance can suffer

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3000773, IEEE Access

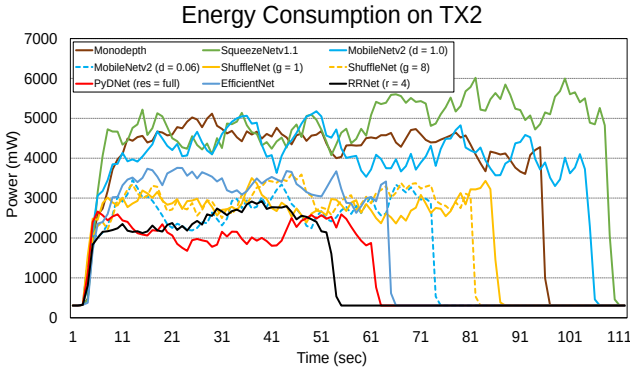


FIGURE 1. Energy consumption against time for six models on NVIDIA TX2.

because trainable data are expensive and limited in volume. Various compensation techniques for trainable data have been implemented [14]–[17], but lightweight models such as MobileNet [18], [19], SqueezeNet [14], ShuffleNet [20], and EfficientNet [21] generally involve encoder-only networks with classification-type architectures and a narrow output resolution. Current state-of-the-art lightweight models cannot be applied to the decoder portion of an encoder–decoder network: the differences in model design and data flow offset the advantages of the reduced model with the encoder structure.

Most recent lightweight networks make extensive use of depthwise convolution layers to reduce computational complexity. While the benefit of such an approach may be considerable, there remain problems with latency which have not been investigated heretofore.

In general, depth estimation methods require an encoder–decoder network, which entails more computation and memory usage than a network of the kind used for classification or detection. In addition, many feature channels in the encoder network can lead to extensive computation in the decoder because of direct layer-wise connections between the two networks; although the deep network structures and overlapping information in both the encoder and decoder tend to improve the performance, such tightly coupled networks require significant hardware resources, increasing the model complexity and hindering efficient deployment in a mobile environment.

This study addresses the problem of designing an energy efficient, lightweight encoder–decoder model for depth estimation.

- We design a model that has the advantages of a light weight and low latency by controlling the ratio of depthwise convolution to other convolutions.
- We propose *Repetition-Reduction (RR) block*, which is an energy efficient encoder design module.
- We propose a *Condensed Decoding Connection (CDC)* tied to the encoder's special repetition structure that delivers feature information to the decoder efficiently with a high feature density while suppressing the rapid growth of model complexity; the CDC's efficiency is maximized by the RR block.
- We propose a *Repetition-Reduction Network (RRNet)*, which is an energy efficient encoder–decoder model based on RR blocks and CDCs that outperforms current state-of-the-art complex and lightweight models in terms of performance, run time, and energy consumption using practical mobile GPU hardware.

The rest of the paper is organized as follows: Section II introduces related works ranging from classical depth estimation approaches to state-of-the-art lightweight neural network model designs. Our design modules, namely, the RR block, CDC, and RRNet, are comprehensively discussed in Section III. Then, Section IV discusses the implementation of the proposed model and compares the experimental results of RRNet with those of various models. Finally, in Section V, we present our conclusions.

II. RELATED WORK

In this section, we briefly review some of the major approaches to supervised and unsupervised depth estimation and lightweight neural network design.

Supervised Depth Estimation: Most depth estimation methods [4]–[6], [9] use supervised approaches, which typically achieve better performance than unsupervised ones. In particular, [6], [9] and [22] employ one or two spatial-pyramid networks with Spatial Pyramid Pooling (SPP) [23], which has been intensively used in encoder–decoder networks [22], [24], [25]. In [25], [26], the SPP module uses adaptive average pooling to compress features into four scales, followed by pointwise convolution to reduce feature dimensions and the concatenation of different feature-map levels to form the final SPP feature maps.

The good performance of supervised methods is due to the presence of ground truth (GT) data. However, it is difficult to prepare reliable depth GT datasets because humans have limited depth-labeling capability and human-reported depth datasets usually contain considerable noise. More accurate GT depths can be obtained with the help of expensive depth sensors such as LiDARs, radars, and laser scanners, but these sensors also have limitations (apart from their cost): LiDAR has shallow channels that hardly cover the full range of image resolutions; active sensors such as Kinect and Time of Flight (ToF) sensors have holes around the object boundaries and are sensitive to strong visible light. Moreover, depth labeling must cover multiple camera viewpoints and thus involves elaborate camera calibrations. Therefore, more attention is now being focused on unsupervised learning, which does not require the manual labeling of datasets.

Unsupervised Depth Estimation: Unsupervised depth learning [1]–[3], [7], [8] offers the benefit of not requiring annotated GT depths. In [3], [7], unsupervised learning removes the need for separate supervisory signals (depth or ego-motion ground truth, or multi-view video) and achieves good performance by introducing camera motion in the learning process. Monodepth, presented at CVPR 2017 by

Godard et al. [27], achieves good performance using a pair of calibrated cameras (right and left) by generating the right image from the left image via left–right consistency alone, without the ground truth. Monodepth uses the VGG [28] and ResNet [29] architectures and generates a decoder with architectures similar but inverse to those of the encoder.

Lightweight Network Designs: Several methods have been proposed for making deep neural networks lighter, among them are deep compression [30], low-bit quantization [31], low-rank approximation [32], matrix decomposition [33], and sparse Winograd-based convolutions [34]. There are also various architectural engineering approaches based on the bottleneck structure and layer factorization; these are mainly introduced in state-of-the-art lightweight networks such as SqueezeNet [14], MobileNet [18], [19], ShuffleNet [20], and EfficientNet [21].

Depthwise Separable Convolution (DWconv) [35] is a layer-factorization approach that is widely used in lightweight models. DWconv factorizes one convolution layer into a depthwise convolution and a pointwise convolution. The depthwise convolution has very light computation requirements and few parameters, because each channel of the input performs a convolution using a single unique depth filter; this is an extreme case of group convolution. The pointwise convolution forms a final output layer that interchanges information to compensate for any performance degradation caused by the depthwise convolution. Our design modules also utilize DWconv, which is covered in detail in Section III.

SqueezeNet [14] is a backbone network showing similar performance to AlexNet [36]. It is composed of Fire modules, each of which first factorizes one convolution layer into two different kernels of sizes, 3×3 and 1×1, and then concatenates them into one output convolution layer. SqueezeNet has the advantage of reduced weight parameters because of the small kernel sizes and the division of one layer into two.

MobileNet [18], [19] uses a design block that is primarily based on inverted residual connections, linear bottlenecks, and DWconv [35]. The scaling parameters of a lightweight backbone network of such modules could be used to change the network channels or input feature size for various purposes, such as accommodating specific target hardware platforms.

ShuffleNet [20] utilizes a group convolution and adopts channel shuffling, which allows active interchange of the feature information. The effectiveness of channel shuffling, which compensates information loss due to group convolution through a group scaling parameter along with a backbone network, has been experimentally demonstrated.

Other recent efficient architectures such as ESPNetv2 [37], DiCENet [38], and DABNet [39] also utilize DWconv. ESPNetv2 [37] applies DWconv with atrous convolution within the dilation rate range of the number of branches, so that each DWconv has a differently sized receptive field. DiCENet [38] transposes one input to three so that each height, width, and channel are at the input depth, processes them using DWconv, and then transposes them back to the original shape in order to concatenate those into one layer. DABNet [39] halves the number of input channels by a 3×3 standard convolution and then applies DWconv. The 3×3 receptive field is then divided into two 3×1 paths. DWconv is performed again with the opposite 1×3 receptive field, and passes through a 1×1 pointwise convolution that restores the number of channels to the original.

On the basis of the above studies, it seems that the bottleneck block is very efficient for model compression and that it is well-utilized in lightweight networks. Enriched information in the upper layer is transmitted to the bottom layer and is contracted by the pointwise convolution layer. MobileNetv2 [19] considers inverted residual blocks to account for residual connections to the bottom layer. However, this structure hardly handles the residual connections to multiple connections in the bottom layer or skip connections to the layers in the decoder. This drawback has inspired the method proposed in this study.

III. METHODS

In our model, RRNet, we consider both GPU latency and computation. We propose the RR block and CDC as building components for an efficient encoder–decoder model. To design RRNet, we have studied lightweight convolutions such as *depthwise* convolution and *pointwise* convolution from the mobile GPU perspective. We have observed that although depthwise convolution involves a small amount of computation, its GPU latency is higher than that of other convolution operations such as the 3×3 standard convolution and the pointwise convolution, which we have described in detail in the Bottleneck Part in Section III A. Therefore, our approach to designing the model architecture of the RR block is to form an efficient structure of lightweight layers by keeping the number of channels in the depthwise layer sufficiently large to reduce computation, but also small enough to enhance the latency. Despite the latency issue, depthwise convolution offers substantial benefits in terms of computation and the number of parameters. Therefore, we strive to maintain the correct ratio between the depthwise convolution layers and other kinds of convolutions. In addition, CDC, the other building block, is designed to maintain the directionality with which the computation is reduced, while passing rich information from the encoder to the decoder.

Next, we briefly introduce the roles of the RR block and the CDC. The RR block is designed for the encoder and it extracts the feature information. To do this efficiently, lightweight layers are used, which can be repeated according to the repetition parameter value r; thus, the performance of the encoder can be improved as the model scales up and the number of parameters increases. The CDC is designed for the decoder. The RR block increases the number of layers while performing repetition through the lightweight layers; the CDC collects some of these layers through a concatenation operation. Thus, the CDC and RR block work closely together through the cycle of repetitions. The output layer

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3000773, IEEE Access

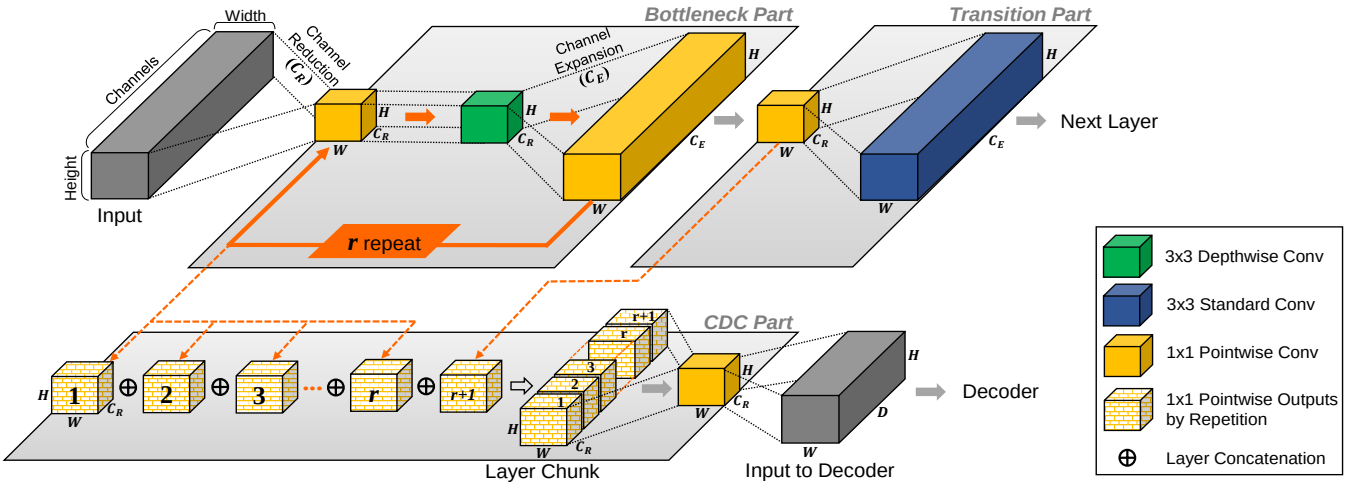


FIGURE 2. Proposed RR block (Bottleneck Part + Transition Part) and CDC Part. In the ***Bottleneck Part***, the input channel is reduced to the value cr through channel reduction by 1×1 pointwise convolution, and 3×3 depthwise convolution is processed with the same channel value. Then, the channel value is increased to ce through channel expansion by 1×1 pointwise convolution. This process is repeated r times, as indicated by the orange arrows. During the repetition, small pointwise convolution outputs that have the channel value cr are concatenated sequentially in the ***CDC Part***. When the repetition ends, the ***Transition Part*** reduces the channel of the Bottleneck Part output to cr through 1×1 pointwise convolution, increases the channel back to ce through 3×3 standard convolution, and forwards it to the input of the next encoder layer. At this time, the ***CDC Part*** concatenates one last small pointwise convolution output (cr channel) and finally performs a reduction process by 1×1 pointwise convolution to reduce the channel size of the concatenated output from $cr \times (r+1)$ to cr through pointwise convolution. The result is provided as a decoder input.

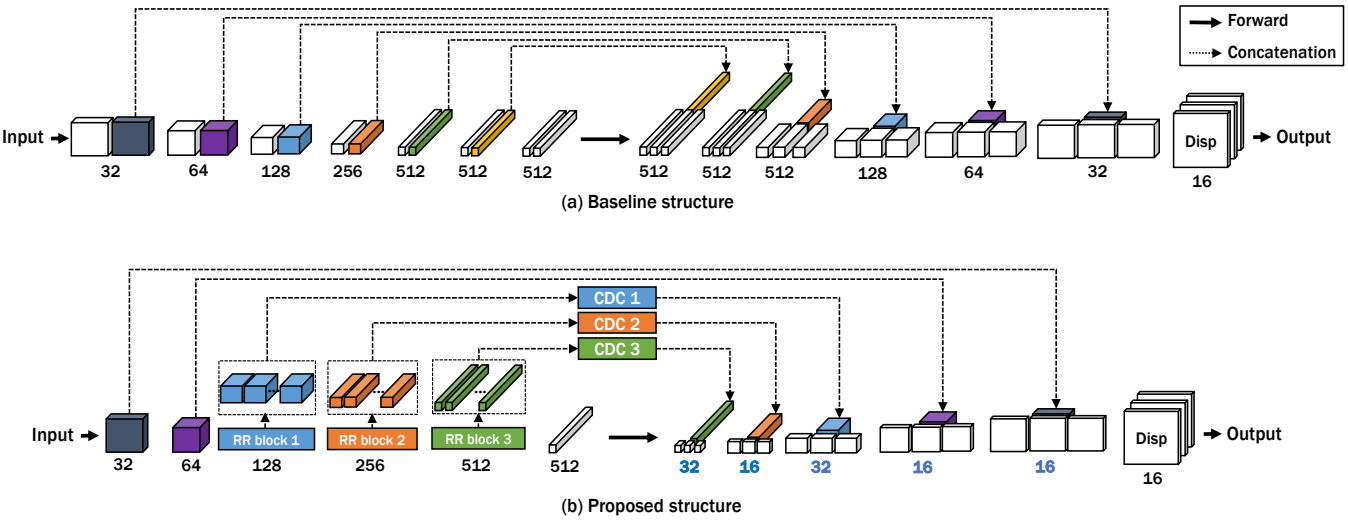


FIGURE 3. Network comparisons. All three-dimensional boxes represent convolution layers; the number below each box is the output channel depth of the layer. Multiple outputs of *k*-th RR block become the inputs of *k*-th CDC (*k* values are $1 \sim 3$ in this figure). (a) Baseline network from Monodepth [27]. (b) Proposed network: RRNet.

that is concatenated by the CDC is called the *layer chunk*. After all the concatenations, the channel of the layer chunk will be as large as the number of repetitions r. Therefore, the process of reducing this channel is performed through pointwise convolution. The channel-reduced layer chunk is then provided as an input to the decoder. Figure 2 illustrates the main building blocks of the proposed model and their operational flow in detail.

A. REPETITION AND REDUCTION (RR) BLOCK

The RR block can be used as a building block for the encoder. It is divided functionally into two parts, the *bottleneck part* and *transition part*, as shown in Figure 2. Each of these is described in detail below.

Bottleneck Part: The bottleneck part consists of lightweight convolution layers arranged in triplets, the first and third convolutions in each triplet being pointwise and the middle convolution being depthwise. The target amount of computation or number of parameters can be scaled by adjusting the number of output channels from each of these layers. Such triplets are called *bottleneck structures*; they have frequently been used in other lightweight model studies, e.g., the DeepVision workshop in CVPR [40], MobileNetv2 [19]

and ShuffleNet [20]. However, the present study differs from earlier ones in always keeping the number of output channels of the depthwise convolution relatively small, because of the critical latency issue in GPU processing.

Let a pointwise or depthwise convolution having an output channel value K be called *pointwise(K)* or *depthwise(K)*. Let CE*(Channel Expansion)* denote a high channel value such as 384 or 512, and CR*(Channel Reduction)* denote a low channel value such as 16 or 32. Then, the structure we propose is *pointwise(CR)–depthwise(CR)–pointwise(CE)*, and this sequence is always maintained. (Channel reduction and channel expansion in the bottleneck part are illustrated in Figure 2.) The purpose of this structure is to reduce the usage of depthwise convolution while using lightweight layers. Depthwise convolution has insufficient GPU-level optimization; therefore, it has a significantly high inference latency for its small amount of computation.

MobileNetv2 [19] is a typical bottleneck structure with the layer sequence *pointwise(CE)–depthwise(CE)–pointwise(CR)*, i.e., with flow opposite to that in our structure. This CE–CE–CR sequence pattern means depthwise convolution always has high output channels, thereby increasing the depthwise convolution percentage.

To gain an intuition regarding the value of the latency and the factors controlling it, we measured the latency of the bottleneck structure of MobileNetv2 [19] layer by layer. The results are indicated in green in Figure 4. As can be seen from the Figure, depthwise convolution has approximately twice the latency of the pointwise convolution, even though the computation is only about one-seventh that of the pointwise convolution before it. According to our detailed analysis of this phenomenon, depthwise convolution is not supported by the GPU-optimized library cuDNN; therefore, it is not possible to perform high-speed computation in this layer as it is in the others. Instead, a standard group convolution function is used to process depthwise convolution. However, this function has not been sufficiently optimized at the GPU level in most DNN frameworks, such as TensorFlow. In addition, we also observed the same latency issue in experiments with the recent CUDA 10 and cuDNN 7.

Figure 4 shows the reason for reducing the use of depthwise convolution as the basic strategy of the RR block. In this example, the expansion value CE and reduction value CR are interchanged in the corresponding bottleneck block of MobileNetv2 [19] (from *pointwise(CE)–depthwise(CE)–pointwise(CR)* to *pointwise(CR)–depthwise(CR)–pointwise(CE)*). In order to match the amount of computation, the input channel of the first *pointwise(CR)* convolution of the RR block was increased from CR to CE (64 to 384). The measured latency is shown in orange in Figure 4. This figure shows that the unnecessary latency of the depthwise convolution can be reduced.

Our approach avoids the undesirably high GPU latency caused by the lack of depthwise-convolution support in the optimization library. Any performance degradation caused by depthwise convolution reduction can be compensated by scaling up the pointwise convolution or by making use of the *Transition Part*, which we will discuss later.

To observe the latency effect in an actual inference, we also measured the latency of the entire encoder for three models (MobileNetv2 [19], DiCENet [38], and the proposed RRNet model) on NVIDIA TX2, scaling each model by its scaling parameters to ensure similar computation. The results, shown in Figure 5, clearly reveal the negative impact of increasing the depthwise convolution usage on GPU latency. The latency of DiCENet was more than twice that of MobileNetv2; this is because DiCENet, despite its small amount of computation, has more than twice the percentage of depthwise convolutions. In addition, the figure shows that RRNet effectively reduces the usage of depthwise convolution and has a positive effect on the actual GPU latency; it is the fastest among the models utilizing lightweight layers.

We can mathematically demonstrate the reduced computational cost of using lightweight layers in the bottleneck part in comparison with standard convolution. The output feature maps of the standard convolution can be expressed as

$$OFM_{h,w,m} = \sum_{i,j,c} IFM_{h+i-1,w+j-1,c} \cdot W_{i,j,c,m}, \quad (1)$$

where h, w, c denote height, width, and channels of input feature maps respectively, m denotes the number of ouptut feature maps, W denotes the weight parameters, OFM denotes the output feature maps, and IFM denotes the input feature maps. Therefore, we can calculate the computational cost $Cost_{conv}$ of standard convolution as

$$Cost_{conv} = H \times W \times C \times M \times K \times K, \quad (2)$$

where H and W are the resolutions of the output feature maps, K is the kernel size, C is the number of input channels, and M is the number of output channels.

We can express the output feature maps of depthwise convolution as

$$OFM'_{h,w,c} = \sum_{i,j} IFM'_{h+i-1,w+j-1,c} \cdot W'_{i,j,c}. \quad (3)$$

Then, we can calculate the computational cost $Cost_{dw}$ of depthwise convolution as

$$Cost_{dw} = H \times W \times C \times K \times K. \quad (4)$$

We can also calculate the computational cost $Cost_{pw}$ of pointwise convolution as

$$Cost_{pw} = H \times W \times C \times M \times 1 \times 1, \quad (5)$$

where the kernel size is 1. Thus, we can calculate the reduction in the computational cost (*CostReduction*) of the bottleneck part:

$$Cost_{bottleneck} = Cost_{pw} + Cost_{dw} + Cost_{pw} \quad (6)$$

$$CostReduction = \frac{Cost_{bottleneck}}{Cost_{conv}} = \frac{1}{M} + \frac{2}{K^2}. \quad (7)$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3000773, IEEE Access

Therefore, we limit the kernel size of the RR block to not exceed 3. It is possible to reduce the amount of computation or the number of parameters by a factor of approximately nine by adopting the bottleneck structure.

Transition Part: The transition part consists of two layers: pointwise convolution and 3×3 standard convolution. In the bottleneck part, we aim to improve the latency efficiency by reducing the depthwise convolution usage. However, this may eventually be disadvantageous for performance, as it decreases the total number of parameters. Therefore, we have designed the transition part to carry out a performance compensation for the encoder. In this part, we use only GPU-optimized functions to reduce the unnecessary consumption of latency caused by insufficient utilization of GPU. Therefore, we select pointwise convolution and 3×3 standard convolution, which are capable of GPU acceleration through the cuDNN library.

The 3×3 standard convolution should be carefully placed, because it introduces a large amount of computational load. In the case of RRNet, we observed empirically that it is possible to supplement the performance sufficiently by using a single 3×3 standard convolution with each RR block. Therefore, our proposed RRNet has one 3×3 standard convolution for each RR block, and the number of output channels of this layer is set to $CE(Channel\ Expansion)$, which is the maximum output channel value used in the bottleneck part.

Repetition in the Bottleneck Part: The CDC, which we discuss later, receives feature information to be sent to the decoder from the bottleneck part. Therefore, all three layers of the bottleneck part (*pointwise–depthwise–pointwise*) are defined as repetition targets, and these layers are repeated according to the value r of the scaling parameter. Repeating these layers not only improves the performance of the encoder, but also generates feature information to be sent to the CDC in a lightweight manner. Moreover, because the amount of feature information to be sent to the decoder via the CDC is determined according to the r value, scaling r value plays an important role in improving the performance.

In more detail, the bottleneck part reduces the dimension, largely similar to principal component analysis (PCA) [41]. To leverage this critical reduction potential, the RR block intensifies the bottleneck part by iterating the lightweight layers and stacking each output per repetition. Therefore, the scaling parameter r denotes the number of repetitions in the bottleneck part of the RR block. The r-fold repetition results in multiple connections to two paths. One path leads to the next layer in the encoder, which includes the transition part; the other leads to the decoder. In the decoder path, each output layer per iteration is provided to the CDC. Therefore, repetition in the RR block not only enriches the model information, but also provides feature-intensive decoder connections.

B. CONDENSED DECODING CONNECTION (CDC)

The CDC, which is represented by patterned light yellow boxes with repetition flow in Figure 2, repeatedly stacks the pointwise convolution layers provided by the RR block (*layer chunk*) in order. We deploy a pointwise convolution layer with output channel scaling to handle the layer chunk; it can also efficiently handle feature explosion, computations, and model size before sending the features to the decoder. A CDC is a module that can replace an existing skip connection, such as the single layer concatenation structure in UNet [42]; its intended use is not replacing or adjusting the decoder structure. However, with our empirically discovered backbone *RRNet* using CDC, we have been able to reduce the channel depth of the decoder significantly with little degradation of the accuracy.

The preservation of a limited number of channels with a linear bottleneck is the key to efficiency in MobileNetv2 [19].

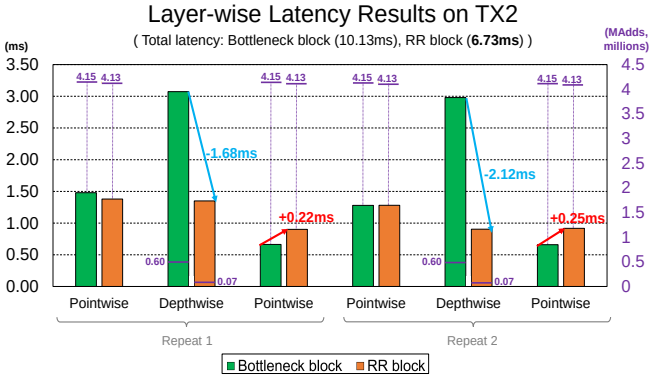
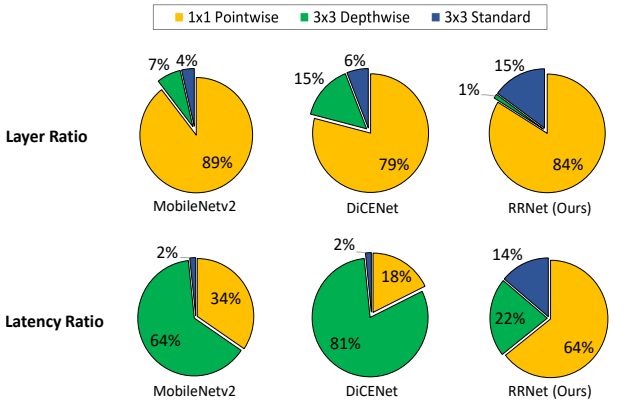


FIGURE 4. Layer-wise latency results on NVIDIA TX2. Data in purple indicate the amount of computation. For MobileNetv2 [19], we selected the fourth bottleneck partially, which shows the typical tendencies of latency. In this figure, the CE(Channel Expansion) value is 384 and the CR(Channel Reduction) value is 64. The output channels for the bottleneck block are 64(input)-CE-CE-CR-CE-CE-CR, whereas those for the RR block are 384(input)-CR-CR-CE-CR-CR-CE.



Latency	100ms	249ms	**72ms**
MAdds	0.62B	**0.53B**	0.68B

FIGURE 5. Encoder latency results of MobileNetv2 [19], DiCENet [38], and the proposed RRNet on NVIDIA TX2. The top circle charts show the distribution of different types of convolution layers in each model. The other circle charts show the corresponding percentage contributions to the latency from each type of layer. For a fair comparison, we used the scaling parameters of each model to scale the computation to a similar level.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3000773, IEEE Access

TABLE 1. Encoder ablation study on KITTI. With two exceptions, all models have the same skip connection [42] and UpConv decoder [27]). In the case of *RRNet with CDC* (our final model, used for comparison with the other models), the skip connection is replaced with the CDC, and the UpConv decoder has a significantly reduced depth channel, as shown in Figure 3 (b). PyDNet uses its own skip connection and decoder as well. Because these two models have different skip connections and decoders, they are separated by a line at the bottom of the table. All models were trained with the *do-stereo* option, which provides stereo-pair images as the ground truth. In all the tables, we chose the *B0 case* for EfficientNet, which is the lightest model presented in this paper. NOTATION: *res* = resolution of the feature map; *r* = repetition parameter; *d* = depth multiplier; *g* = number of groups in the channel shuffling layer.

Encoder	Abs Rel	Sq Rel	RMSE	RMSE log	$\sigma < 1.25$	$\sigma < 1.25^2$	$\sigma < 1.25^3$	MAdds	Params
Monodepth [27] (baseline)	0.068	0.850	4.406	0.146	0.942	0.977	0.989	42.36B	31.6M
SqueezeNetv1.1 [14]	0.074	1.108	4.638	**0.151**	**0.943**	**0.977**	**0.988**	47.95B	15.3M
MobileNetv2 [19] *(d = 1.0)*	0.074	1.194	4.764	0.157	0.942	0.975	0.987	32.46B	17.5M
MobileNetv2 [19] *(d = 0.06)*	0.076	1.011	4.702	0.152	0.939	**0.977**	**0.988**	7.21B	5.1M
ShuffleNet [20] *(g = 1)*	0.120	1.144	5.348	0.206	0.869	0.953	0.978	11.72B	20.3M
ShuffleNet [20] *(g = 8)*	0.094	1.116	4.954	0.170	0.920	0.972	0.986	10.61B	17.2M
EfficientNet [21]	0.088	**0.756**	4.643	0.162	0.912	0.971	**0.988**	10.17B	10.4M
RRNet *(r = 4, no CDC)*	0.074	1.116	4.727	0.153	0.942	0.976	0.987	11.95B	7.8M
PyDNet [45] *(res = full)*	0.094	1.222	5.118	0.175	0.917	0.968	0.976	9.84B	1.9M
RRNet *(r = 4, with CDC)*	**0.071**	0.893	**4.539**	**0.151**	0.940	0.976	**0.988**	**3.26B**	**1.1M**

In this study, the linear bottleneck connects not only the residual but also the decoder layers using CDC. The stacked features are compressed by the pointwise reduction layer, which is then delivered to the decoder in a non-activated form that guarantees linearity. Because of repetition, the number of stacked CDC channels is large; for example, 128 channels repeated 6 times is equivalent to 768 channels. We reduce this number by applying a pointwise reduction layer, resulting in 128 output channels that are connected to the decoder layer as seen in Figure 2. Owing to the pointwise convolution, the total output size becomes significantly smaller than in conventional architectures. In short, by using a smaller unit RR block, enriched CDC features, and a pointwise linear reduction layer, we significantly reduce the number of parameters and computational cost and obtain a level of performance similar to the original one.

We will explain the CDC mathematically. The skip connection can be represented by $F(x) + x$. The + makes the shortest path from the top layer to the bottom layer in the backward propagation process. It is an operation that adds more edges to the computation graph, smoothing the loss function and improving performance. Furthermore, a greater number of nodes in computational graph can lead to a more complex computation, but the additional edges are relatively small increasing the computational complexity. Therefore, CDCs add more edges with the same number of nodes in the network. Assuming a network with L layers with symmetric skip connections, we denote the convolution and deconvolution in each layer with ReLU as F_c and F_d and an RR block with input X_i as $R(X_i)$. Write $R^2(R^1(X_i))$ as $R^2(X_i)$ for simplicity. Then,

$$\Omega(X_i) = [R^1(X_i), R^2(X_i), ..., R^r(X_i), R^{r+1}(X_i)], \quad (8)$$

where Ω denotes the CDCs. We assume that information on the convolutional feature map X_i is to be passed to the corresponding deconvolutional layers X_{L-i-1} and X_{L-i} in the decoder; then,

$$X_{L-i} = F_d(X_{L-i-1} \oplus F_{c_{1\times1}}(\Omega(X_i))), \quad (9)$$

where $\oplus$ denotes feature map concatenation or a similar operation. For back-propagation, we consider the L^{th} layer,

$$X_L = F_d(X_{L-1} \oplus F_{c_{1\times1}}(\Omega(X_0))). \quad (10)$$

We compute the derivative of the loss ℓ with respect to a parameter θ as follows:

$$\nabla_\theta \ell(X_L) = \frac{\partial \ell}{\partial X_{L-1}} \frac{\partial X_{L-1}}{\partial \theta} \oplus \frac{\partial \ell}{\partial \Omega(X_0)}. \quad (11)$$

Therefore, CDCs carry larger gradients than skip connections, which are less likely to approach zero gradients.

C. MODEL FOR EFFICIENT ENCODER–DECODER

For the design of our backbone model *RRNet*, we use the structure of the high-performance complex encoder–decoder model from Monodepth [27] as a reference and set the boundary values of the computation (up to 3.5 billion) and model size (up to 1.5 million). We properly combine the RR blocks, CDCs, standard convolution layers, pooling layers, and other elements. Figure 3 (b) shows our RRNet structure in detail.

1) RRNet Encoder

The RRNet encoder consists of three RR blocks. We use a repetition parameter $r = 4$ for all RR blocks, having discovered from Table 2 that this is the best tradeoff case between performance, computational cost, and model size. We place two successive 3×3 standard convolution layers with maximum pooling to receive the input data; the three RR blocks are placed thereafter. To reduce the feature map resolution, maximum pooling is assigned after each of the RR blocks.

We set the channel reduction in the RR block to 32 (RR block 1) - 64 (RR block 2) - 128 (RR block 3), listed in order from the input, and 128 - 256 - 512 in the same order as that followed for channel expansion. We place an average pooling layer at the end of the encoder. We have compared the RRNet encoder with a variety of state-of-the-art lightweight designs without CDC by replacing only

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3000773, IEEE Access

TABLE 2. RR block ablation study of the repetition *r* on KITTI. (Each model has the same structure as the baseline RRNet architecture, including CDC and its own decoder, as described in Figure 3 (b). All models are trained with the *do-stereo* option). Our RRNet architecture design shows the highest performance at $r = 4$. To improve the performance beyond this point, it may be necessary to increase the number of RR blocks in the encoder along with the scaling *r* value. Alternatively, increasing the complexity of the decoder design can be considered.

Model	Repeats	Abs Rel	Sq Rel	RMSE	RMSE log	$\sigma < 1.25$	$\sigma < 1.25^2$	$\sigma < 1.25^3$	MAdds	Params
RRNet	$r = 1$	0.082	1.117	4.941	0.160	0.931	0.974	0.987	**2.80B**	**0.7M**
RRNet	$r = 2$	0.079	1.120	4.904	0.158	0.933	0.974	0.987	2.95B	0.8M
RRNet	$r = 3$	0.077	1.086	4.797	0.157	0.935	0.975	0.987	3.11B	1.0M
RRNet	$r = 4$	**0.071**	**0.893**	4.539	**0.151**	**0.940**	**0.976**	**0.988**	3.26B	1.1M
RRNet	$r = 32$	0.074	0.922	**4.523**	0.154	0.938	**0.976**	**0.988**	7.64B	4.8M

TABLE 3. CDC ablation study on KITTI (all models are trained with *do-stereo* option). Skip is a skip connection originally proposed in the UNet structure [42]. Skip and Upconv represent the baseline structure from Monodepth [27]. The UpConv decoder corresponds to Figure 3 (a) and consists of 6 decoding layers; the number of channels in each layer is 512-512-256-128-64-32. The RRNet decoder corresponds to Figure 3 (b) and consists of 5 decoding layers; the number of channels in each layer is 32-16-32-16-16. The notation $(\times k)$ means that all output channels are multiplied by *k*. For CDC, the output channel is from the final pointwise layer which directly connects to the decoder layer. In this experiment, we observed the extent to which performance is maintained when the complexity of the decoder is significantly reduced in the environment where the CDC is applied as a connection, under the same encoder and decoder condition. The encoder used is RRNet that can utilize CDC. The basic performances of a baseline [27] encoder and the RRNet encoder were first compared. When CDC is used as a connection, the performance is maintained even if the number of channels of all decoder layers is reduced from 16× to 32×.

Encoder	Decoder	Connection	Abs Rel	Sq Rel	RMSE	RMSE log	$\sigma < 1.25$	$\sigma < 1.25^2$	$\sigma < 1.25^3$	MAdds	Params
Monodepth [27]	UpConv	Skip	0.068	0.850	4.406	0.146	0.942	0.977	0.989	42.36B	31.6M
RRNet ($r = 4$)	UpConv	Skip	0.074	1.116	4.727	0.153	0.942	0.976	0.987	11.95B	7.8M
RRNet ($r = 4$)	RRNet Decoder	CDC	**0.071**	**0.893**	**4.539**	**0.151**	**0.940**	**0.976**	**0.988**	**3.26B**	**1.1M**
RRNet ($r = 4$)	UpConv ($\times 1/8$)	Skip	0.083	1.213	5.070	0.162	0.927	0.972	0.986	2.88B	1.2M
RRNet ($r = 4$)	UpConv ($\times 1/8$)	CDC	**0.079**	1.098	4.924	**0.159**	**0.931**	**0.974**	**0.987**	3.37B	1.7M
RRNet ($r = 4$)	UpConv ($\times 1/16$)	CDC	0.080	1.125	**4.908**	**0.159**	0.930	0.974	0.986	3.19B	1.3M
RRNet ($r = 4$)	UpConv ($\times 1/8$)	CDC ($\times 1/8$)	0.080	**1.095**	4.956	**0.159**	0.929	0.972	**0.987**	2.66B	1.1M
RRNet ($r = 4$)	UpConv ($\times 1/32$)	CDC ($\times 1/32$)	0.084	1.100	4.988	0.164	0.922	0.971	0.986	**1.82B**	**0.7M**

the encoder. The results, shown in Table 1, show that the RRNet encoder is comparable to other lightweight designs in terms of computation and model size, while having a special stacking structure for the CDC.

2) CDC in RRNet

CDCs are assigned to the RR blocks one by one and the channel reduction depth of the pointwise reduction layer is set to the maximum depth of the channel expansion of its corresponding RR block. Therefore, there are three CDCs in total; the reduction depth values of the CDCs corresponding to the RR blocks from one to three are 128 - 256 - 512, respectively. Table 3 compares the skip connection with our CDCs. Using the same RRNet encoder, we vary the connection type and the corresponding decoder size. In Table 3, the notation $\times 1/n$ means that the number of channels of concatenated features is reduced by a factor of n. For example, if the number of channels of a concatenated feature is 512 and the notation is $\times 1/8$, the final number of channels of this feature is 64. In the case of the VGG $\times 1/8$ decoder, the skip connection has 1.2M parameters and the CDCs have 1.16M parameters; however, the performance of the CDCs is better than that of the skip connection. Moreover, CDC $\times 1/16$ has only 0.91M parameters; yet, it achieves higher accuracy than the skip connection with 1.2M parameters. Owing to the enriched information in the CDCs, the network can preserve its performance with fewer parameters.

3) RRNet Decoder

The RRNet decoder consists of five upscaling layers. Each layer is based on the UNet decoder structure [42]. The upscaling layer consists of three steps: (1) upscale convolution through linear interpolation of feature maps, (2) concatenation of encoder feature information, and (3) 3×3 decoding convolution. The reduction layer from the RR block - CDC is used here for the concatenation of the second step. As mentioned above, we have been able to reduce the channel depth of all decoder layers significantly, e.g., from 512 to 16. Table 3 shows that our model can withstand 32 times lighter decoders while maintaining performance. In the RRNet decoder, the channel depths of the five layers are 32-16-32-16-16, from the input to the output.

IV. EXPERIMENTS

To evaluate the effectiveness of RRNet, we used unsupervised depth estimation [27] as our baseline. Depth estimation provides low-level information used by other higher-level applications and is frequently executed as a background process. Therefore, our objectives for this evaluation were high performance and minimal runtime and power consumption on mobile devices.

A. EXPERIMENTAL SETUP

Dataset: The KITTI dataset [43] was adopted for the evaluation; this dataset consists of 200 training image pairs and 200 test image pairs. The baseline method [27] was an unsupervised approach that did not use the ground truth depth. KITTI 2015 contains 42,382 rectified stereo pairs

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3000773, IEEE Access

TABLE 4. Comprehensive training results. Dataset K refers to the KITTI dataset and CS refers to the Cityscapes dataset.

Model	Dataset	Mode	Abs Rel	Sq Rel	RMSE	RMSE log	$\sigma < 1.25$	$\sigma < 1.25^2$	$\sigma < 1.25^3$	Params
Monodepth [27] (baseline)	K	mono	0.148	1.344	5.927	0.247	0.803	0.922	0.964	31.6M
Eigen [49]	K	mono	0.204	1.548	6.307	0.283	0.702	0.891	0.959	54.2M
Liu [50]	K	mono	0.201	1.585	6.472	0.273	0.681	0.898	0.967	40.0M
Zhou [51]	K	mono	0.209	1.769	6.857	0.284	0.679	0.886	0.958	34.2M
SqueezeNetv1.1 [14]	K	mono	0.132	1.406	6.249	0.223	0.826	0.933	0.972	15.3M
MobileNetv2 [19] *(d = 1.0)*	K	mono	0.132	1.596	6.361	0.226	0.830	0.933	0.971	17.5M
MobileNetv2 [19] *(d = 0.06)*	K	mono	0.147	1.675	6.803	0.253	0.795	0.909	0.960	5.1M
PyDNet [45]	K	mono	0.153	1.363	6.030	0.252	0.789	0.918	0.963	1.9M
EfficientNet [21]	K	mono	0.132	1.488	6.343	0.228	0.823	0.930	0.972	10.4M
DiCENet [38]	K	mono	0.165	**1.328**	5.782	0.239	0.781	0.923	0.970	3.6M
DABNet [39]	K	mono	0.160	1.338	**5.542**	0.228	0.804	0.934	**0.973**	1.9M
RRNet	K	mono	**0.130**	1.547	6.341	**0.222**	**0.834**	**0.937**	**0.973**	**1.1M**
Monodepth [27] (baseline, cap 50m)	K	mono	0.140	0.976	4.471	0.232	0.818	0.931	0.969	31.6M
Garg [8] cap 50m	K	mono	0.169	1.080	5.104	0.273	0.740	0.904	0.962	16.8M
PyDNet [45] cap 50m	K	mono	0.145	**1.014**	**4.608**	0.227	0.813	0.934	0.972	1.9M
EfficientNet [21] cap 50m	K	mono	0.128	1.205	5.940	0.223	0.823	0.932	0.973	10.4M
RRNet cap 50m	K	mono	**0.125**	1.197	5.791	**0.216**	**0.836**	**0.940**	**0.975**	**1.1M**
Monodepth [27] (baseline)	CS	mono	0.101	1.319	4.732	0.169	0.928	0.972	0.986	31.6M
SqueezeNetv1.1 [14]	CS	mono	0.102	1.716	5.184	0.177	0.932	0.972	0.984	15.3M
MobileNetv2 [19] *(d = 1.0)*	CS	mono	0.105	**1.514**	**5.139**	0.171	0.926	**0.973**	**0.987**	17.5M
MobileNetv2 [19] *(d = 0.06)*	CS	mono	0.132	1.790	5.348	0.192	0.909	0.969	0.984	5.1M
PyDNet [45]	CS	mono	0.113	1.624	5.354	0.187	0.910	0.967	0.984	1.9M
EfficientNet [21]	CS	mono	0.159	2.862	6.104	0.224	0.891	0.956	0.976	10.4M
RRNet	CS	mono	**0.095**	1.532	5.201	**0.170**	**0.933**	**0.973**	0.986	**1.1M**
Monodepth [27] (baseline)	CS + K	mono	0.124	1.076	5.311	0.219	0.847	0.942	0.973	31.6M
Zhou [51]	CS + K	mono	0.198	1.836	6.565	0.275	0.718	0.901	0.960	34.2M
PyDNet [45]	CS + K	mono	0.146	**1.291**	**5.907**	0.245	0.801	0.926	0.967	1.9M
EfficientNet [21]	CS + K	mono	**0.131**	1.554	6.317	**0.223**	**0.834**	**0.936**	0.972	10.4M
RRNet	CS + K	mono	**0.131**	1.478	6.238	**0.223**	0.828	**0.936**	**0.973**	**1.1M**
Monodepth [27] (baseline)	K	do-stereo	0.068	0.850	4.406	0.146	0.942	0.977	0.989	31.6M
SqueezeNetv1.1 [14]	K	do-stereo	0.074	1.108	4.638	**0.151**	**0.943**	**0.977**	**0.988**	15.3M
MobileNetv2 [19] *(d = 1.0)*	K	do-stereo	0.074	1.194	4.764	0.157	0.942	0.975	0.987	17.5M
MobileNetv2 [19] *(d = 0.06)*	K	do-stereo	0.076	1.011	4.702	0.152	0.939	**0.977**	**0.988**	5.1M
EfficientNet [21]	K	do-stereo	0.088	**0.756**	4.643	0.162	0.912	0.971	**0.988**	10.4M
PyDNet [45]	K	do-stereo	0.094	1.222	5.118	0.175	0.917	0.968	0.976	1.9M
RRNet	K	do-stereo	**0.071**	0.893	**4.539**	**0.151**	0.940	0.976	**0.988**	**1.1M**
Monodepth [27] (baseline)	CS + K	do-stereo	0.071	0.980	4.542	0.150	0.943	0.976	0.987	31.6M
PyDNet [45]	CS + K	do-stereo	0.089	1.179	5.056	0.171	0.921	0.970	0.985	1.9M
EfficientNet [21]	CS + K	do-stereo	0.091	**0.829**	**4.547**	0.162	0.912	0.972	**0.988**	10.4M
RRNet	CS + K	do-stereo	**0.078**	1.070	4.859	**0.157**	**0.934**	**0.974**	0.987	**1.1M**

TABLE 5. Overall inference results with NVIDIA TX2. For fairness of measurement, we measured only the inference time and set the TX2's GPU to the maximum performance mode. Each model was averaged by repeating 10 tests. Runtime refers to the execution time for 400 images from the KITTI evaluation set. Energy consumption during runtime is measured in joules (J).

Model	Memory Usage	Runtime	Energy
Monodepth [27] *(baseline)*	3926MB	96s	415.36J
SqueezeNetv1.1 [14]	4238MB	110s	531.32J
MobileNetv2 [19] *(d = 1.0)*	2658MB	106s	435.30J
MobileNetv2 [19] *(d = 0.06)*	4211MB	74s	201.24J
ShuffleNet [20] *(g = 1)*	4072MB	87s	245.45J
ShuffleNet [20] *(g = 8)*	3890MB	82s	242.52J
PyDNet [45] *(res = full)*	3431MB	62s	132.96J
EfficientNet [21]	4524MB	65s	210.97J
RRNet	**1281MB**	**54s**	**128.84J**

TABLE 6. Overall inference results obtained using Xeon E3-2620, NVIDIA Titan X, and NVIDIA TX2. TDP stands for Thermal Design Power. The results obtained for each model were averaged after performing 4000 (400 images $\times$ 10) tests. All results refer to the execution time for one image from the KITTI evaluation set.

Power (TDP)	250W	95W	15W
Model	Titan X	Xeon E3-2620v3	TX2
Monodepth [27] *(baseline)*	0.044 s	0.448 s	0.240 s
SqueezeNetv1.1 [14]	0.045 s	0.511 s	0.275 s
MobileNetv2 [19] *(d = 1.0)*	0.048 s	0.350 s	0.265 s
MobileNetv2 [19] *(d = 0.06)*	0.028 s	0.221 s	0.185 s
ShuffleNet [20] *(g = 1)*	0.058 s	0.268 s	0.218 s
ShuffleNet [20] *(g = 8)*	0.063 s	0.288 s	0.205 s
PyDNet [45] *(res = full)*	0.030 s	**0.124 s**	0.155 s
EfficientNet [21]	0.053 s	0.242 s	0.163 s
RRNet	**0.026 s**	0.125 s	**0.135 s**

from 61 scenes, with 1242 $\times$ 375 pixels. We evaluated 200 high-quality disparity images in the training set, covering 28 scenes. The remaining 33 scenes contained 29,000 images for training and 1,159 images for validation. For convenience, we used left and right images together as a single input.

Training details: RRNet was trained using TensorFlow 1.4.0 [44] with CUDA 8.0 and cuDNN 7.0 as the back-ends. We assessed the performance of RRNet with respect to the results reported for Monodepth [27] and PyDNet [45]. Our baseline for all evaluations was a Monodepth model based on the VGG structure. For a fair comparison, we trained our network with the same protocol as that used in [27], [45]: batches of eight images resized to 256 $\times$ 512 $\times$ 3, and 300 epochs executed on 29,000 images. Our loss function and hyperparameters also matched those in [27], [45].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3000773, IEEE Access

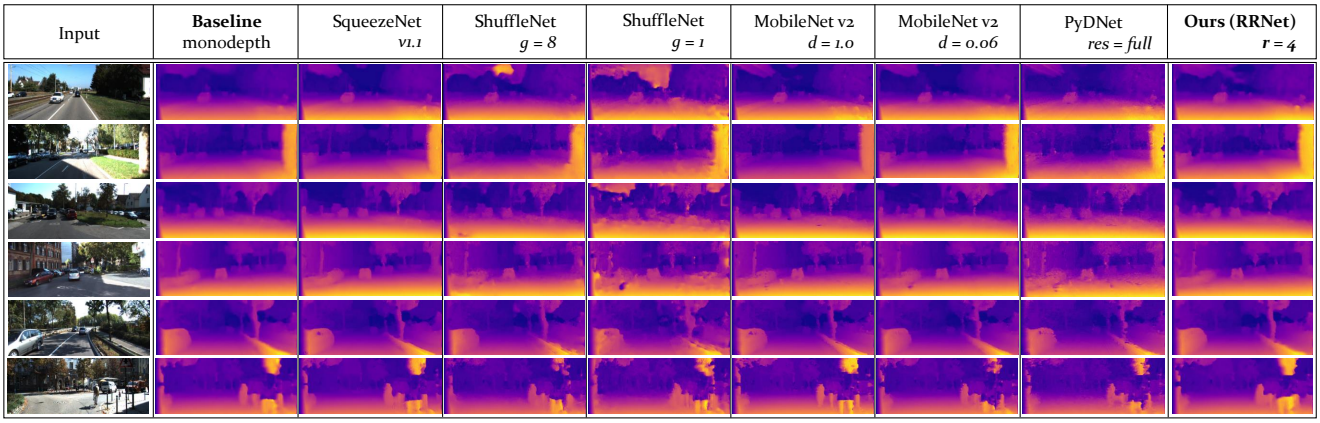


FIGURE 6. TX2 inference results on KITTI. Here *res* = resolution of the feature map; *r* = repetition parameter; *d* = depth multiplier; *g* = number of groups in the channel shuffling layer.

There were two training modes, namely *mono* and *do-stereo*. These modes were selected according to the number of inputs: *mono* mode for a single input and *do-stereo* mode for concatenated image inputs. In general, stereo approaches outperform monocular approaches via information enrichment. However, a stereo approach can significantly increase memory usage, runtime, and energy consumption. To show the effectiveness of RRNet, we adopted both *mono* and *do-stereo* modes for evaluation. Table 1 compares RRNet with state-of-the-art lightweight models. For the best case, our proposed method decreased the amount of computation by factors of 4.16 $\sim$ 13.9 and the number of parameters by factors of 4.53 $\sim$ 16.6 relative to other methods.

For evaluation on another dataset, we adopted the Cityscapes [46] dataset, which contains 22,973 training stereo pairs that comprise street scenes recorded across Germany. This dataset has a higher image resolution and quality than KITTI. The results are summarized in Table 4. Training on the Cityscapes dataset was performed for the initial 10 epochs using the same schedule as the baseline method [27]. We expected that the performance would be improved in this case in comparison with KITTI. However, the performance was in fact similar or even slightly reduced for all models, and training itself was not stable, as mentioned for Monodepth [27].

B. EVALUATION OF RRNET ON NVIDIA TX2

Model size, runtime, and energy consumption are the key considerations for resource-constrained mobile applications. We evaluated all models on the NVIDIA TX2 Development Kit [47] by using depth estimation based on an ARM-A57 CPU with 8 GB of main memory running on a Linux operating system. TX2 is a mobile GPU processor based on the Pascal architecture; it has 1.5 TFLOPS and 15 W of thermal design power (TDP). We rebuilt a common workstation environment using TensorFlow 1.4.0 [44], CUDA 8.0, and cuDNN 7.0, as in our training setup, the only difference being that we customized TensorFlow to run on the ARM architecture.

We emphasize energy consumption because it is affected by the runtime factor. In real environments, the main bottlenecks of depth estimation in mobile applications are energy consumption, runtime, and memory usage. The results of executing the models on TX2 are summarized in Figure 1 and Table 5. Here, runtime refers to the actual model execution time without considering the preparation time for iterative execution, such as model initialization or data preparation, which needs to be performed only one time during the entire process. To ensure fair measurements, we enabled the maximum performance mode via the *jetson_clocks.sh* script provided for TX2. This mode prevents the OS from adjusting the GPU's power consumption by itself, so that there is no unnecessary impact on runtime measurement. The result is the average of the results of a total of 10 tests with 400 images each, which is a total of 4000.

RRNet had a total energy consumption of 128.84 J, the lowest of all the models; moreover, it required approximately 3.84$\times$ less power and was 2.21$\times$ faster than the others. Thus, RRNet outperformed the other models in terms of energy consumption, memory usage, and computation, as shown in Figure 1 and Table 5.

In addition to the previous tests, we conducted inference tests in a high-performance workstation environment and explored whether real-time processing is possible (i.e, takes less than 0.03 s per image). The hardware used for this test consisted of a Xeon E3-2620v3 (which had a 6 cores and operated at 2.4 Ghz with 95 W of TDP) and a Titan X (which was based on the Maxwell architecture and had 11 TFLOPS with 250 W of TDP). The results are summarized in Table 6, which presents the runtime to process one image. The cases where real-time processing was possible were MobileNetv2 (d = 0.06) [19], PyDNet [45], and RRNet, all tested with Titan X. RRNet showed the best speed, taking about 0.026 s per image (38.46 FPS). For the test with Xeon E3-2620v3, PyDNet [45] and RRNet showed the best speeds, which were similar to their results obtained using the TX2 GPU.

We also analyzed the depth estimation results, as shown in Table 1. RRNet with $r = 4$ showed the best performance in terms of most evaluation metrics, except for $\delta < 1.25$, with a 0.003 difference, and for $\delta < 1.25^2$, which was degraded by 0.001, in comparison with PyDNet. Moreover, the last row in Figure 6 suggests that a close-range traffic sign was well-recognized using our method. This result is similar to that of baseline Monodepth [27], which is a highly complex model.

V. CONCLUSIONS

We proposed RR block and CDC as the fundamental components of a lightweight encoder–decoder network. The RR block and CDC proposed herein are efficient in terms of computation and parameter size, facilitating an improved design for encoder–decoder architectures. In addition, we show that such a scheme results in lightweight and high-performance network designs. The RR block forms repetitive feature connections to the CDC, which can deliver feature maps to the decoder efficiently without a rapid increase in the information or model complexity by using suppressing convolutions, as shown in Figure 2.

We also introduced our backbone network, RRNet, which is an encoder–decoder model that is very lightweight owing to the aforementioned RR block and CDC. Because RRNet's architecture is designed by considering GPU latency, it is advantageous with regard to energy efficiency as well as with regard to computation time. The proposed model is sufficiently lightweight to be applied to mobile devices and differs from previous encoder–decoder enhancement approaches [19], [48].

On a commercial mobile GPU, RRNet outperforms previous state-of-the-art models, reducing the runtime by approximately 2.21× while providing energy savings of up to 3.84× and memory savings of up to 3.06× with optimal performance. In our future work, we plan to evaluate the generalization capability of the RR block and CDC by applying them to semantic segmentation, object detection, and other significant problems.

REFERENCES

[1] C. Zhou, H. Zhang, X. Shen, and J. Jia, "Unsupervised learning of stereo matching," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (ICCV)*, 2017, pp. 1567-1575.

[2] T. Zhou, M. Brown, N. Snavely, and D. Lowe, "Unsupervised learning of depth and ego-motion from video," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2017, pp. 1851-1858.

[3] R. Mahjourian, M. Wicke, and A. Angelova, "Unsupervised learning of depth and ego-motion from monocular video using 3d geometric constraints," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018, pp. 5667-5675.

[4] Z. Liang, Y. Feng, Y. Guo, H. Liu, W. Chen, L. Qiao, L. Zhou, and J. Zhang, "Learning for disparity estimation through feature constancy," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018, pp. 2811-2820.

[5] J. Zbontar and Y. LeCun, "Stereo matching by training a convolutional neural network to compare image patches," *J. Mach. Learn. Res.*, 2016, vol. 17, pp. 1-32.

[6] A. Ranjan and M. J. Black, "Optical flow estimation using a spatial pyramid network," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2017, pp. 1-2.

[7] V. Casser, S. Pirk, R. Mahjourian, and A. Angelova, "Depth prediction without the sensors: leveraging structure for unsupervised learning from monocular videos," in *Proc. of the AAAI Conf. Artif. Intell (AAAI)*, 2019, pp. 8001–8008.

[8] R. Garg, V. K. BG, G. Carneiro, and I. Reid, "Unsupervised CNN for single view depth estimation: geometry to the rescue," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2016, pp. 740–756.

[9] T. W. Hui, X. Tang, and L. C. Change, "LiteFlowNet: A lightweight convolutional neural network for optical flow estimation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018, pp. 1-9.

[10] K. Karsch, C. Liu, and S. B. Kang, "Depthtransfer: depth extraction from video using non-parametric sampling," *IEEE Trans. Pattern Anal. Mach. Intell.*, 2014, pp. 2144-2158.

[11] L. Ladický, B. Zeisl, and M. Pollefeys, "Discriminatively trained dense surface normal estimation," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2014, pp. 468-484.

[12] *CUDA: Parallel Computing Platform and Programming Model Provided by NVIDIA Corp.*, Sep. 2019, Available: https://developer.nvidia.com/cuda-zone.

[13] *cuDNN: GPU-accelerated Library of Primitives for Deep Neural Networks Provided by NVIDIA Corp.*, Sep. 2019, Available: https://developer.nvidia.com/cudnn.

[14] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1MB model size," *arXiv:1602.07360*, 2016.

[15] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv:1312.4400*, 2013.

[16] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A, Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2015, pp. 1-9.

[17] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp 2818-2826.

[18] 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:1704.04861*, 2017.

[19] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L.C. Chen, "Inverted residuals and linear bottlenecks: mobile networks for classification, detection and segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018, pp. 4510-4520.

[20] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: an extremely efficient convolutional neural network for mobile devices," *arXiv:1707.01083*, 2017.

[21] M. Tan and Q. V. Le, "Efficientnet: rethinking model scaling for convolutional neural networks," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2019, pp. 6105–6114.

[22] J. R. Chang and Y. S. Chen, "Pyramid stereo matching network," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018, pp. 5410-5418.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2014, pp 346–361.

[24] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2017, pp. 6230-6239.

[25] L. C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," *arXiv:802.02611*, 2018.

[26] L. C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *arXiv:1606.00915*, 2016.

[27] C. Godard, O. Mac Aodha, and G. J. Brostow, "Unsupervised monocular depth estimation with left-right consistency," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2017, pp. 270-279.

[28] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2015, pp. 1-14.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 770-778.

[30] S. Han, H. Mao, and W. J. Dally, "Deep compression: compressing deep neural network with pruning, trained quantization and huffman coding," *arXiv: 1510.00149*, 2015.

[31] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv: 1606.06160*, 2016.
[32] Y. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," *arXiv: 1511.06530*, 2015.
[33] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 4013-4021.
[34] X. Liu, J. Pool, S. Han, and W. J. Dally, "Efficient sparse-winograd convolutional neural networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2018, pp. 1-4.
[35] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2017, pp. 1251-1258.
[36] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2012, pp. 1097-1105.
[37] S. Mehta, M. Rastegari, L. Shapiro, and H. Hajishirzi, "ESPNetv2: A Light-weight, Power Efficient, and General Purpose Convolutional Neural Network" in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 9190–9200.
[38] S. Mehta, H. Hajishirzi, and M. Rastegari, "DiCENet: Dimension-wise Convolutions for Efficient Networks" *arXiv:1906.03516*, 2019.
[39] G. Li, I. Yun, J. Kim, and J. Kim, "DABNet: Depth-wise Asymmetric Bottleneck for Real-time Semantic Segmentation", in *Proc. Brit. Mach. Vis. Conf. (BMVC)*, 2019, pp. 1–12.
[40] S. Oh, J. Lee, and H. J. S. Kim, "Fast and light-weight unsupervised depth estimation for mobile GPU hardware," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR) DeepVision Workshop*, 2018, pp. 2-4.
[41] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2013, pp. 1139-1147.
[42] O. Ronneberger, P. Fischer, and T. Brox, "U-net: convolutional networks for biomedical image segmentation," in *Proc. Int. Conf. Med. Image Comput. Comput.-Assist. Intervent. (MICCAI)*, 2015, pp. 234-241.
[43] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2012, pp. 3354-3361.
[44] M. Abadi et al., "Tensorflow: a system for large-scale machine learning," in *Proc. USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2016, pp. 265-283.
[45] M. Poggi, F. Aleotti, F. Tosi, and S. Mattoccia, "Towards real-time unsupervised monocular depth estimation on CPU," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2018, pp. 5848–5854.
[46] 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 *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 3213-3223.
[47] *Jetson TX2 Development Kit Provided by NVIDIA Corp.*, Mar. 2017, Available: https://developer.nvidia.com/embedded/buy/jetson-tx2.
[48] M. Siam, M. Gamal, M. Abdel-Razek, S. Yogamani, and M. Jagersand, "Rtseg: Real-time semantic segmentation comparative study," *arXiv:1803.02758*, 2018.
[49] D. Eigen, C. Puhrsch, and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2014, pp. 2366-2374.
[50] F. Liu, C. Shen, G. Lin, and I. Reid. "Learning depth from single monocular images using deep convolutional neural fields," *IEEE Trans. Pattern Anal. Mach. Intell (TPAMI)*, 2016, pp. 2024–2039.
[51] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, "Unsupervised learning of depth and ego-motion from video," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2017, pp. 1851–1858.

SANGYUN OH (S'18) received the B.S. degree in computer science and engineering from Ajou University, Suwon, Korea, in 2015. He is currently pursuing the Ph.D. degree at Ulsan National Institute of Science and Technology (UNIST), Ulsan, Korea. His current research interests include machine learning, reconfigurable architectures, energy-efficient embedded accelerators for deep learning, and neuromorphic computing.

HYE-JIN S. KIM (S'20) received the B.S. and M.S. degree in POSTECH, Pohang, Korea, in 2001 and 2003, respectively. She is currently pursuing the Ph.D. degree in KAIST, Daejeon, Korea. She is currently working on ETRI, Daejeon, Korea. Her current research interests include machine learning, information theory and computer vision, specially depth estimation and light-weight model architecture for deep learning.

JONGEUN LEE (S'01-M'11) received the B.S. and M.S. degrees in electrical engineering and the Ph.D. degree in electrical engineering and computer science from Seoul National University, Seoul, South Korea, in 1997, 1999, and 2004, respectively. He is an Associate Professor at the Ulsan National Institute of Science and Technology, Ulsan, South Korea. Since 2009, he has been on the faculty of the School of Electrical and Computer Engineering at UNIST, South Korea. His research interests include neural network processors, reconfigurable architectures, and compilers.

JUNMO KIM (S'01-M'05) received the B.S. degree from Seoul National University, Seoul, Korea, in 1998, and the M.S. and Ph.D. degrees from the Massachusetts Institute of Technology (MIT), Cambridge, in 2000 and 2005, respectively. From 2005 to 2009, he was with the Samsung Advanced Institute of Technology (SAIT), Korea, as a Research Staff Member. He joined the faculty of KAIST in 2009, where he is currently an Associate Professor of electrical engineering. His research interests are in image processing, computer vision, statistical signal processing, machine learning, and information theory.