Fall 2021

Faster Depth Estimation for Situational Awareness on Urban Streets

Sanjana Srinivas
San Jose State University

FASTER DEPTH ESTIMATION FOR SITUATIONAL AWARENESS ON URBAN STREETS

A Thesis

Presented to

The Faculty of the Department of Computer Engineering

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Sanjana Srinivas

December 2021

The Designated Thesis Committee Approves the Thesis Titled

FASTER DEPTH ESTIMATION FOR SITUATIONAL AWARENESS ON URBAN STREETS

by

Sanjana Srinivas

APPROVED FOR THE DEPARTMENT OF COMPUTER ENGINEERING

SAN JOSÉ STATE UNIVERSITY

December 2021

Mahima Agumbe Suresh, Ph.D.	Department of Computer Engineering
Magdalini Eirinaki, Ph.D.	Department of Computer Engineering
Gheorghi Guzun, Ph.D.	Department of Computer Engineering

ABSTRACT

FASTER DEPTH ESTIMATION FOR SITUATIONAL AWARENESS ON URBAN STREETS

by Sanjana Srinivas

Depth estimation algorithms are useful components of computer vision systems to assess video streams on urban streets. They can provide important information about the street space and improve situational awareness for humans. Deep learning algorithms for depth estimation are slow in providing on-time street evaluation for the users. To provide situational awareness to humans, the inference time needs to be small, so that results are fresh and meaningful. This paper explores approaches like switching to efficient convolutional neural network, quantization and pruning to improve inference time with a little compromise on the performance. We explore the impact of each of these methods independently and introduce a hybrid method. We evaluate the execution time, resource utilization, and performance of various state-of-the-art depth estimation algorithms. We compare these with our approach of using the three optimization techniques both independently and in hybrid. We observe that using these optimization techniques can improve the inference time dramatically, with a 57.3% speedup in inference time and a 94.5% reduction in memory utilization while improving the object level performance (RMSE) by 3.8%.

ACKNOWLEDGMENTS

I would like to thank my advisor - Dr. Mahima Agumbe Suresh, SJSU Department of Computer Engineering, for giving me the opportunity to work on this research. Her continuous support and guidance carried me through all stages of the study, experimentation, and writing of this work.

I would also extend warm regards to my committee members - Dr. Magdalini Erinaki and Dr. Gheorghi Guzun, whose knowledge and perspective were instrumental in the completion of the thesis.

I am extremely grateful to Cheryl R. Cowan, SJSU Graduate Studies Associate, for assisting me with the guidelines and submissions.

Last, but not least, I would like to thank my family. Without their support, I would have never been able to complete this thesis or pursue my dreams.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

In recent years, urban streets have become increasingly more multi-modal with the introduction of micro-mobility alternatives such as electric scooters and skateboards, in addition to the bikes, cars, and pedestrians that have occupied the street space. An increasing number of accidents are attributed to this disruptive change [1]. With advancements in artificial intelligence and a multitude of smart devices on everyday objects and the advent of edge computing, there is a unique opportunity to improve situational awareness for humans using the street spaces.

One of the key steps to providing situational awareness is to be able to assess the scene in which a human is present, often captured by a camera. Computer vision algorithms, especially depth estimation, is a critical component for this purpose. Depth estimation provides important information about the distance between a human and the surrounding objects. Most state-of-the-art results for depth estimation are based on fully supervised learning [2], [3], which makes use of ground truth depth labels. However, it is tedious to acquire a massive amount of accurate ground truth depth labels. Recently, competing results are available, where self-supervised techniques are used for stereo training from multiple cameras [4], [5].

In this research, we focus on monocular depth training, since it is most suitable for our application. Smart devices that can be used by humans for real-time inference need to have a small form factor, and it is preferable to have as few components as possible on-board. Monocular depth training requires only a single camera as opposed to stereo training. With this constraint, depth estimation using a single image sequence is considered to be an ill-posed problem as many points in 3D converge to the same point in 2D. Current depth estimation techniques handle this problem but have a very high computational overhead, which makes them unsuitable out-of-the-box for real-time applications [6]–[10]. To execute these algorithms at the edge in real-time, there is a need

to introduce performance optimizers that can provide slightly less accurate inference within a deadline on these resource-constrained devices.

Recently, efficient convolutional neural networks have proven to provide competing performance, that is comparable with complex networks like ResNet while ensuring low inference time and memory utilization. Additionally, *quantization* as a technique to reduce memory and computation footprint are being studied [11]–[14]. Another technique, called *pruning*, leverages an understanding of the inner workings of neural networks to eliminate redundant nodes [15]–[21]. For our application, these techniques can be used to reduce the inference time and make the complex monocular depth estimation techniques suitable for real-time computation at the edge.

1.1 Motivating Application

A report by Consumer Product Safety Commission states that between 2017-2019, it is estimated that e-Scooter related accidents have resulted in 50,000 emergency department visits and 27 fatalities [1]. These numbers are on the rise year over year. A vast majority of the accidents are attributed to riders lacking awareness of their surroundings and failing to anticipate collisions. This raises the need for better e-Scooter safety measures and regulations. Smart helmets can be used as a safety precaution by e-Scooter riders to avoid casualties. They can help in detecting and warning the rider about the approaching obstacles in real-time. Depth information is critical to understand the environment and detect any obstacle. A smart helmet, embedded with a simple safety assessment algorithm that makes use of the depth map, can potentially avoid e-Scooter accidents. The pixels within a specific threshold of the depth map are identified and categorized as dangerous obstacles.

The big-picture vision of this research is to develop a smart helmet that accurately identifies dangerous situations such as obstacles and intersections, and warns the users in

real-time, i.e., without them experiencing any time lag. We envision that the computational needs of such an application can be fulfilled completely at the edge.

The nature of the applications changes the performance demands. Since the result of depth estimation is consumed by a human rather than a machine/algorithm, we can make a few compromises on the performance of the depth estimation. Specifically, we do not require high accuracy at the pixel level. On the other hand, we require high accuracy at the object level. The application also demands that results of depth estimation are "fresh" for the consumption of the rider. Older results can be more harmful than helpful. But, it is intractable to quantify by a deadline as the environment - rider, scene, and camera - are dynamic. The specific deadline for such a real-time system is an orthogonal area of research and is out of the scope of this research. In this work, we aim to make use of different optimization techniques to bring down the inference time so that the predictions are generated in real-time. Our goal is to study if depth estimation algorithms can achieve minimal inference time and computational costs on an edge device without losing accuracy at the object level.

The key contributions of this research are:

- We present a deep learning technique that uses a combination of quantization and pruning on existing monocular depth estimation algorithms.
- We modify the existing baseline Monodepth2 [6] architecture with an edge-friendly, efficient convolutional neural network.
- We evaluate the inference time, memory footprint, and performance of the depth architecture with an efficient network, quantization, pruning, and a combination of these approaches
- We evaluate the hypothesis that the combination of pruning and quantization improves inference time without losing the performance of the deep learning network.

- We propose a metric to evaluate depth estimation that is coarse-grained in contrast to the pixel level. This is useful in applications that have a human in the loop.

The remainder of this thesis is structured as follows. First, Chapter 2 provides an insight into the previous work on depth estimation, efficient convolutional networks, quantization, and neural network introspection. Chapter 3 elaborates on the methodology, including the loss functions for monocular depth estimation, the network architecture, and techniques to improve inference time at the edge. Chapter 4 consolidates the experimental results and their comparison with the previous state-of-the-art methods. It also includes the step-by-step experiment setup, their results, and an overview of the system requirements and hyperparameters used. Chapter 5 provides additional experimental results that support our research. The proposed object-level evaluation metric is discussed in Chapter 6. Finally, Chapter 7 contains the future work and Chapter 8 concludes with an overview of the outcome.

2 RELATED WORKS

Depth estimation is crucial to understand the 3D world for navigation assistance in computer vision and robotics systems. Several methods to estimate depth have been adopted over the years. Initially, active sensors like LiDAR and RADAR were used on the navigating agent. But due to its limited range, increased noise, power consumption, and cost, a deep convolutional neural network with passive sensors like a camera became popular. Deep convolutional neural network methods can be mainly categorized into 3 major types - 1) fully supervised learning, 2) self-supervised stereo learning and 3) self-supervised monocular learning. Our application demands the use of self-supervised monocular learning because of resource constraints and the ease with which data can be collected.

2.1 Self-Supervised Monocular Depth Estimation

Depth estimation using monocular images is considered an ill-posed problem as more than one point in 3D converges to the same point in 2D. Therefore, the state-of-the-art results on depth estimation are mostly from fully supervised [2], [3] or self-supervised stereo methods [4], [5]. However, monocular images are easier to acquire when compared to stereo images or depth ground truth data. One of the initial monocular depth estimation approaches [8], proposes a dual network to estimate the depth and pose independently using consecutive temporal frames. This enables training with only monocular video and produces competing performance. Using the dual network architecture, recent works in [6] and [7] have closed the performance gap between monocular depth estimation and fully supervised or self-supervised stereo methods. Godard et al. [6] use superior loss functions to handle occlusions and mask training pixels that violate the camera motion assumption. This is extended by [7] to obtain a sharper depth map. This research uses self-attention and discrete disparity volume decoder to analyze a broader context and achieve improved performance. The state-of-the-art architectures proposed for monocular

depth estimation are computationally complex. Smart helmets used by e-Scooter riders are edge devices. These edge devices have computational restrictions and the application demands lesser inference time. Therefore, deploying the depth estimation architecture on such devices without deteriorating the performance is the challenge.

2.2 Small and Efficient Convolutional Neural Networks

With edge computing becoming increasingly essential, we are observing a gradual shift from accuracy-driven neural networks like ResNet [22], Inception [23] and DenseNet [24] to solutions that offer a balance between performance and computational requirements. Techniques like Neural Architecture Search [25] and RegNet Design Spaces [26] are automated machine learning techniques that explore the design spaces of the artificial neural network to achieve exceptional results and inference latency. However, these techniques require vast computing resources. Therefore, in this research, we incline towards small and efficient manual-designed convolution neural networks. SqueezeNet [27], MobileNet [28], [29], ShuffleNet [30] and EfficientNet [31] are some of the examples of such networks.

MobileNetv1 [28] makes use of depth-wise separable convolutions in place of standard convolutions thereby reducing the number of computations. Whereas ShuffleNet [30] makes use of point-wise convolutions in a group to achieve comparatively better results than MobileNetv1. MobileNetv2 [29] achieves state-of-the-art results by surpassing the compression rates of MobileNetv1 and ShuffleNet with the inclusion of inverted residual linear bottleneck along with depth-wise separable convolutions. It is specifically tailored for resource constraint environments and thus is suitable for our use case.

2.3 Neural Network Introspection

A neural network is widely known as a black-box model. Neural network introspection is a strategy that helps in understanding this black box. In depth estimation

networks, understanding the properties of the image recognized by each hidden node will help decode the information of the image - from pixel to depth map. With this knowledge, a significant piece of information about the network can be revealed – the redundant nodes. Fully connected networks such as the depth network are known to be over-parameterized (i.e multiple neurons encode nearly the same information) in most of the state-of-the-art algorithms and therefore consist of several redundant nodes. Pruning is an introspection technique that identified the redundant nodes and eliminates redundant weights or layers. There are two types of pruning – unstructured pruning and structured pruning.

Unstructured pruning or weight pruning is a procedure through which individual redundant parameters within a layer are set to zero or pruned. Han et al. [16] proposes to reduce the energy required to run large neural networks by learning to identify unimportant connections in the network using L1 regularization and iterative pruning. The speedup at inference is achieved by introducing sparsity with the aid of sparse matrix operation libraries and/or hardware. As opposed to [16], [17] achieves sparsity during the process of pruning with the use of an additional gate matrix. Both these methods and many weight pruning techniques have overhead costs involved to attain sparsity. Although pruning of weights reduces a significant number of parameters from the fully connected layers, it fails to adequately reduce the computation costs and run-time memory. This is because most of these resources are consumed by the activation maps which are still dense, as opposed to the weights.

The second pruning technique is structured pruning or channel pruning where the entire columns of weights of a redundant channel are pruned. [19] achieves sparsity while preserving significant performance by randomly deactivating input-output channel-wise connections in a convolution layer. [32] achieves sparse channel pruning using L1 regularization to identify redundant weights at inference. Compared to these works, [18]

proposes to achieve model compression by pruning the channels of a trained CNN using L1 regularization and fine-tuning or retraining to regain the lost accuracy.

Advantages of channel pruning include ease of implementation when compared to weight pruning. Unlike weight pruning, channel pruning does not require the support of special software or hardware libraries to speed up inference time and memory utilization. Therefore, to reduce the computational footprint of our depth network we propose a structured pruning - channel pruning technique for our use case.

As suggested in many pruning literature [18], [19], we adopt a three-step neural network channel pruning technique. The three steps include 1) Training the baseline network to learn the importance of weights in different channels, 2) Pruning to remove redundant channels, and 3) Fine-tuning to recover the accuracy lost during the process of pruning.

2.4 Quantizing Neural Networks

Quantization refers to the technique of storing floating-point values as lower bit-width values. It is one of the commonly used approaches to reduce the computational footprint of a neural network. Quantization can be classified as dynamic and static quantization or post-training quantization (PTQ) and quantization aware training (QAT).

In dynamic quantization, weights are quantized ahead of time, while the activations are dynamically quantized during inference, making this the simplest form of quantization to use. This is utilized in instances when loading weights from memory rather than computing matrix multiplications take up the majority of the model execution time (in the case of LSTM and transformer type models). Whereas in the static quantization model, weights and activation are quantified before inference. Theoretically, static quantization is said to be faster than dynamic quantization since unlike dynamic quantization the scales and zero points of activation in static quantization are determined before inference thereby reducing the computation overhead while inferencing. However, the model size and

memory consumption of the two remains to be almost the same [33]. Considering the speed, we prefer static quantization over dynamic quantization.

Post-training quantization [13], [33], [34] is a technique wherein we start with a trained model, and pre-process this model wherever possible, by merging the batch normalization layers into preceding layers. Finally, the model undergoes static or dynamic quantization. In static PTQ, to establish the best quantization parameters for activations, the process is calibrated with a representative dataset. Quantization aware training [33], [35] is the process where the model is quantized and further fine-tuned to recover loss of information - the floating-point values before quantization and after de-quantization are not completely recoverable. Although QAT achieves good performance, the process of quantization and fine-tuning itself is time-consuming.

Lower bit-width values achieved through quantization can be 16, 8, 4, and even 2-bit-widths. Layer-by-layer quantization proposed in [12] quantifies floating-point weights and output data at each layer of the neural network to 8-bit values. The resultant compression ratio after quantization is approximately 7% of the original VGG like network. [13] propose a 4-bit quantization method and its resulting compression ratio on VGG is around 12%. The performance is retained by solving for the quantization values using the minimum mean squared error method. Whereas, [35], [36] achieves binary or 2-bit quantization to drastically reduce the computational requirement. However, for very low-bit quantization, hardware support (Raspberry Pi supports 8, 16, and 32 bits) is restricted. Therefore, we consider 8-bit post-training static quantization to be a good compromise considering precision, ease of implementation, compression, and hardware support.

3 EDGE PERFORMANCE OPTIMIZATION ON DEEP LEARNING FOR DEPTH ESTIMATION

In this section, we present our depth prediction network, which generates a depth map from single color input. We first go over the essential concepts in Monodepth2 [6], a self-supervised training method for monocular depth estimation, and then we discuss the optimization strategies for efficient edge deployment.

3.1 Network Architecture for Baseline Training

Monodepth2 [6] is one of the state-of-the-art architectures for estimating depth in an unsupervised manner using monocular images. The baseline depth training is adopted from [6]. In Monodepth2, a *depth network* is used to predict the depth map D_t of image I_t. It is comprised of an encoder-decoder architecture. An RGB image I_t of height H and width W having 3 channels, is fed as input to ResNet18 [22], which learns the features of the input image to generate an output $X_t = resnet_\theta(I_t)$ having 512 channels. The last fully connected layer of ResNet18 is discarded to retain the higher resolution X_t feature maps with 512 channels. This is then traversed through the decoder which upsamples X_t by making use of the intermediate feature maps in the encoder to generate a depth map D_t.

There are two loss functions used to train the model are - photometric reprojection loss and edge-aware smoothness loss [6].

$$L_p = \min_{t'} pe(I_t, I_{t'\to t}) \tag{1}$$

$$pe(I_t, I_{t'\to t}) = \frac{\alpha}{2}(1 - SSIM(I_t, I_{t'\to t}) + (1+\alpha)||I_t, I_{t'\to t}||_1 \tag{2}$$

where $\alpha = 0.85$.

The photometric reprojection loss is calculated with the aid of a pose network. A pose network is used to predict the transformation matrix $T_{t\to t'}$ having 6 degrees of freedom

using image frames I_t and $I_{t'} \in \{I_{t-1}, I_{t+1}\}$ where $I_{t'}$ are the temporally adjacent frames to I_t. Images $I_{t'}$ are transformed to I_t using the transformation matrix, resulting in $I_{t'\to t}$. The photo-metric re-projection loss (Equation 1) is given by the minimum re-projection error *pe* (L1 distance and Self-Similarity (SSIM) (Equation 2)) in the pixel space between I_t and $I_{t'\to t}$.

$$L_s = |\partial_x d_t^*| e^{|\partial_x I_t|} + |\partial_y d_t^*| e^{-|\partial_y I_t|} \tag{3}$$

where $d_t^* = d_t / \overline{d_t}$ is the mean normalized inverse depth [37] used to prevent shrinking of estimated depth. Smoothness loss L_s takes into account the edges present in the frame and also helps to predict smooth disparity estimates in local neighborhoods. It penalizes drastic depth changes in the flat regions. This is given by Equation 3.

$$\mu = [\min_{t'} pe(I_t, I_{t'\to t}) < \min_{t'} pe(I_t, I_{t'})] \tag{4}$$

where $[.]$ is the Iverson bracket.

Following [6] auto-masking technique is used to preserve the static scene moving camera assumption in self-supervised monocular trained depth estimation methods. The binary mask $\mu \in \{0, 1\}$ (Equation 4) will help in masking the loss L_p to include only the pixels where the re-projection error of $I_{t'\to t}$ is lower than the error of $I_{t'}$.

$$L = \mu L_p + \lambda L_s \tag{5}$$

The total loss is equivalent to the sum of the masked photo-metric re-projection loss and weighted edge-aware smoothness loss (Equation 5).

This computationally heavy network is optimized to efficiently deploy on an edge device by leveraging an efficient convolutional neural network - MobileNetv2 in place of the ResNet18 encoder.

3.1.1 *Using Efficient Convolutional Neural Network for Encoder*

Although the ResNet18 encoder helps in achieving state-of-the-art performance, the memory usage and inference time due to this encoder are significantly high. Therefore, we propose to switch the underlying ResNet18 encoder with MobileNetv2 [29], which provides a favorable accuracy-speed/memory trade-off. Fig. 1 demonstrates the above mentioned switch.

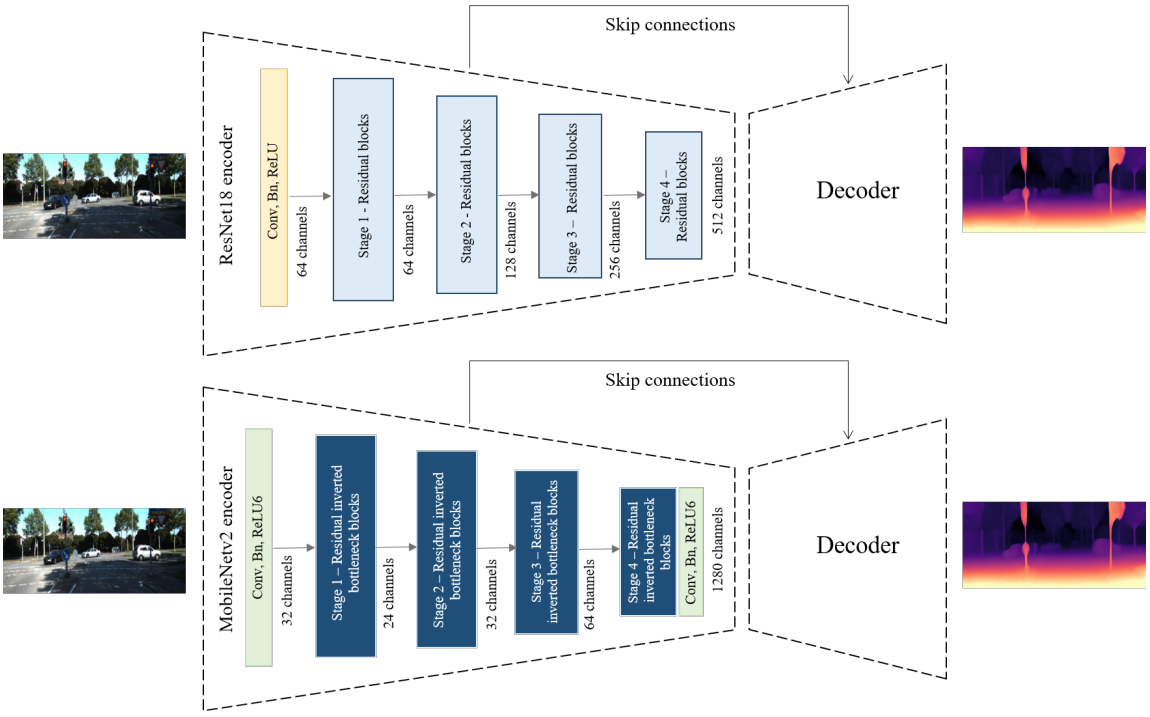

Fig. 1. **Network Architecture**. [top figure] Monodepth2 model with ResNet18 encoder. [bottom figure] Depth network with MobileNetv2 as the encoder.

Most efficient convolutional neural networks make use of depth-wise separable convolution which is a combination of depth-wise convolution and point-wise convolution. Unlike standard convolution, depth-wise convolution performs lightweight filtering by applying the convolution kernel to per input channel. The features are then combined linearly with the help of point-wise convolution which is a 1x1 standard convolutional

layer. Additionally, MobileNetv2 [29] includes inverted residual with linear bottlenecks. Residual blocks are similar to that of ResNet18, it adds a shortcut connection between current and previous activations. Inverted linear bottlenecks are used to expand the lower dimensional input to a higher dimension layer. This is then consumed by the depth-wise separable convolution that maintains low parameter count and computational costs. The higher dimensional layer is again converted back to the lower dimension using the pointwise 1x1 convolution, to allow residual connection from the previous activation. Here, since we squeeze the layer, linear activation is used to tackle the loss of information. This design is represented in Fig. 2 and Fig. 3.

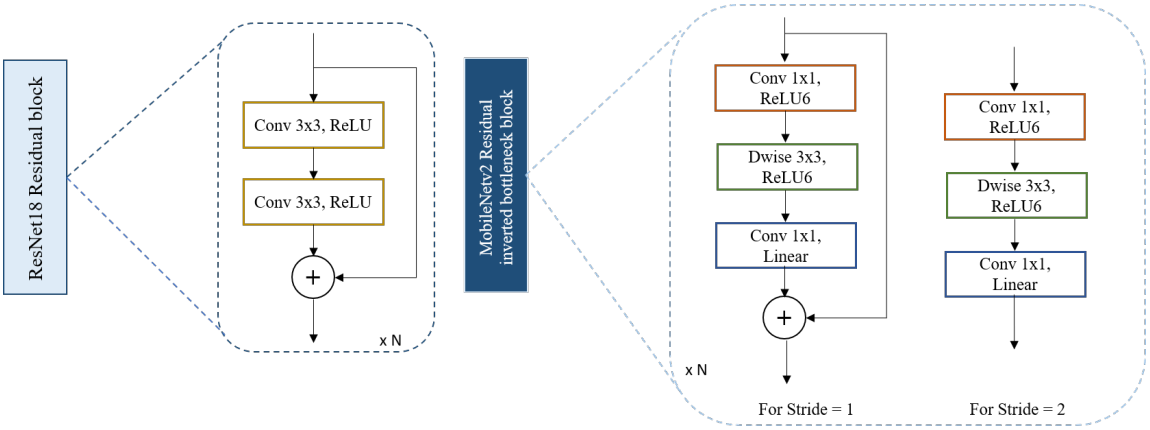

Fig. 2. **Expansion of Convolutional Blocks**. [left figure] Expansion of residual block in ResNet18. [right figure] Expansion of residual inverted bottleneck block in MobileNetv2.

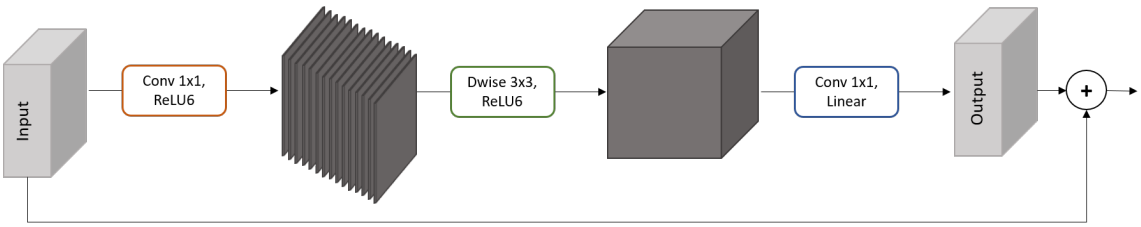

Fig. 3. Evolution of input in the residual inverted bottleneck block of MobileNetv2.

Additionally, optimization techniques like channel pruning and quantization are applied to further bring down the computational requirement. Encoder being the heavy work in the architecture as it plays a crucial role in learning the feature maps of an input image, it requires more memory and its inference time is higher when compared to the decoder. Therefore, the optimization techniques are predominantly applied to the encoder and the decoder is adjusted accordingly. Fig. 1 and 4 shows the network before and after altering the architecture.

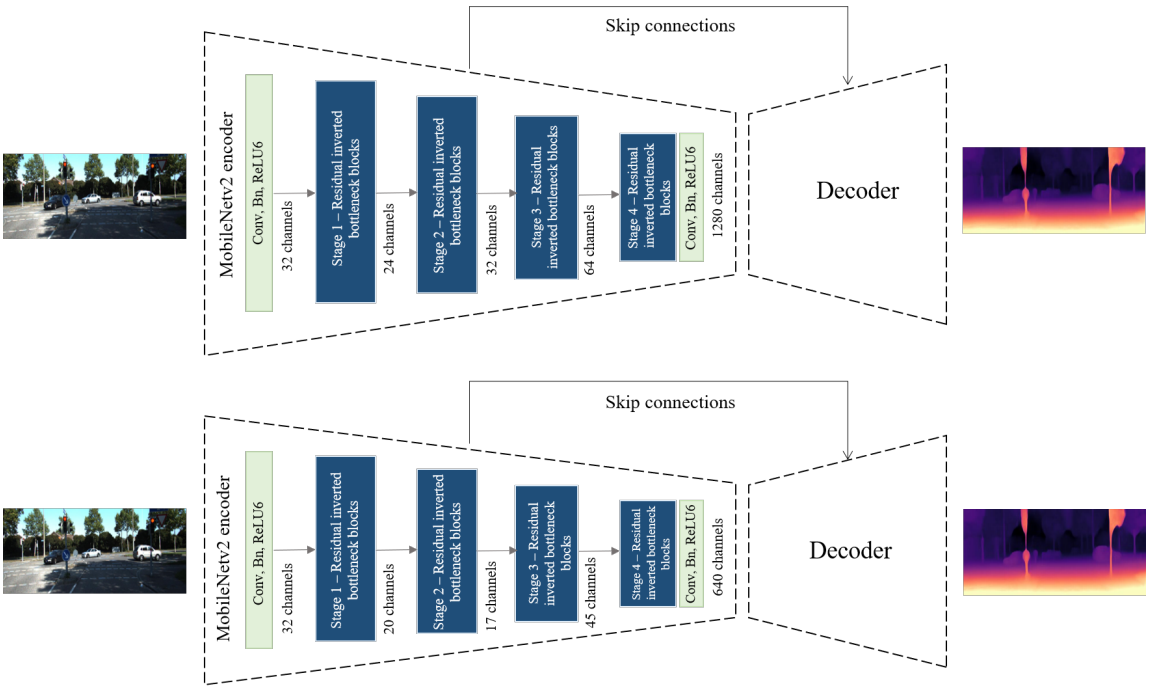

Fig. 4. **Network Architecture - Before and After Pruning**. [top figure] Baseline network architecture before pruning. [bottom figure] The pruned model achieved by removing redundant channels in convolution layers. The pruned model weights are then quantized to 8-bit integers to further reduce memory footprint and inference time.

3.2 Channel Pruning with Sparsity

Channel pruning is the process of removing the redundant channels in the layers of the network. Following [18], we adopt the channel pruning technique. Removing the

channels in one layer will change the input of the following layer. Therefore, the network needs to be accordingly modified. Channel pruning constitutes two steps - selection and reconstruction.

3.2.1 Channel Selection

First, the model is trained with the baseline architecture that has a small and efficient convolutional neural network (MobileNetv2) in the encoder.

$$||Lw_i||_1 = \sum_{x=1}^{h*w} |W_{h,w}| \tag{6}$$

where W are the corresponding weights.

From this architecture, the redundant channels are selected with the help of the L1 norm of the weights of the channels in a given convolution layer. Consider a convolution layer with n input channels and c output channels. The expected outcome is to reduce the c output channels in the original architecture to $c^{'}$ ($1 \leq c^{'} \leq c$). L1 norm of a channel i in a convolution layer l with n input channels, spatial height of h and spatial width of w is given by Equation 6. It is the sum of the absolute weights in the given channel.

The array $Lw = [||Lw_1||_1, ||Lw_2||_1 .... ||Lw_i||_1 .... ||Lw_n||_1]$ is the list of the L1 norms of all the n channels in convolution layer l of a network. From this list Lw, k (pruning rate) minimum values are identified. The indices of these minimum values are the indices of the channels that need to be pruned. Fig. 5 demonstrates the process of channel pruning. The original trained model will have $k = 0$, as the value of k is increased the value of $c^{'}$ will decrease. In [15], [20], it is observed that the sparsity increases as we go deeper into the network. Therefore in our research, we have used incremental pruning rate as we go deeper into the neural network. The intuition behind incremental pruning rate comes from the fact that while fine-tuning the pruned model, the layers close to the input are more likely to learn the general features of the input images than the layers deeper in the network. Therefore we do not want to prune the initial layers much.

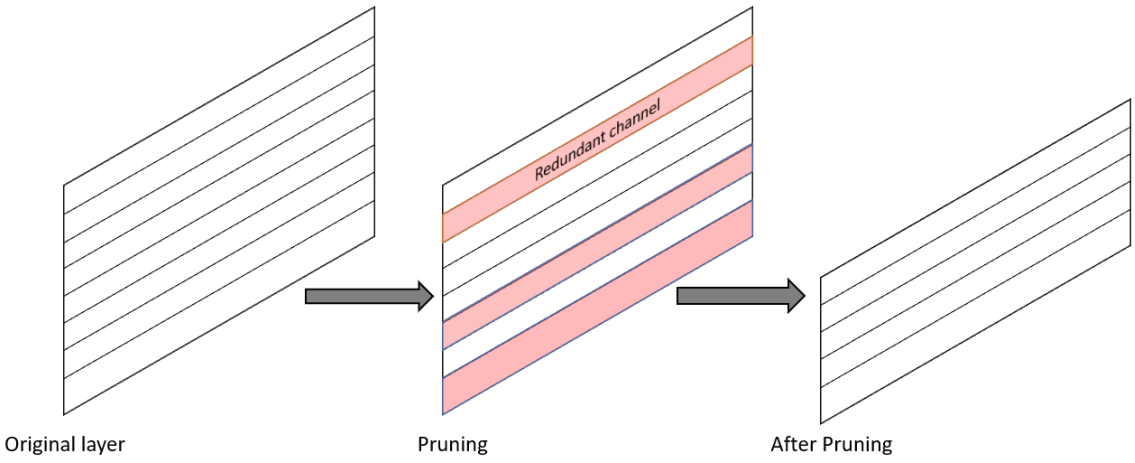

Fig. 5. **Channel Pruning**. A convolution layer consists of prescribed channels. Some of these channels can be redundant (as shaded on red). Pruning is the process of identifying these redundant channels and removing them.

3.2.2 Network Reconstruction

The encoder which is a directed acyclic graph is traversed to perform channel selection (Section 3.2.1) at each convolution layer given the desired pruning rate. The output of the channel selection process gives the indices of the channels to be pruned. These channels are pruned by removing all their incoming, outgoing connections and their corresponding weights. Note that while performing channel selection in the subsequent convolution layers, the weights corresponding to pruned channels is not considered as it is already pruned [18]. After pruning the original output at layer l, $W_l = [W_1, W_2, W_3, ...W_c]$ will become $W_l = [W_1, W_2, W_3, ...W_{c'}]$.

Pruning often leads to accuracy loss, this lost accuracy can be compensated considerably by fine-tuning or retraining the pruned network with the weights of the baseline network as the initial weights.

3.3 8-Bit Static Quantization

Another simple edge computing performance optimization technique, quantization, is performed on the pruned network. A quantized model uses integers instead of floating-point values to perform some or all of the operations on the weight and activation matrix. This enables the use of high-performance vectorized operations on a variety of hardware platforms, as well as a more compact model representation. This strategy is particularly effective in saving a significant amount of inference calculation time without sacrificing too much inference accuracy.

$$x_q = round(\frac{1}{s}x + z) \tag{7}$$

$$s = \frac{\beta - \alpha}{\beta_q - \alpha_q} \tag{8}$$

$$z = round(\frac{\beta\alpha_q - \alpha\beta_q}{\beta - \alpha}) \tag{9}$$

In static post training quantization, a floating point value $x \in [\alpha, \beta]$ is converted into a 8-bit integer $x_q \in [\alpha_q, \beta_q]$ using Equation 7 [11], [14], where s (Equation 8) is a positive floating point scale parameter and z (Equation 9) is an integer zero-point parameter. α and β are the minimum and maximum bounds of the weight/activation matrix.

To perform quantization, we use the fine-tuned pruned model, the convolution and batch normalization layers are fused so that these multiple layers are abstracted as a single layer. It is then calibrated on a representative dataset to understand the distribution and to calculate scale and zero-point parameters. Channel wise 8-bit quantization is performed on the trained floating-point weights and activations with the calculated s and z

values. At inference, these quantized weights and activations are used to achieve faster inference and reduced memory usage.

4 EXPERIMENTS

In this section, we detail our experimental methodology.

4.1 Datasets and Training

We use the KITTI Raw data set [38] to train and evaluate our technique. This is a vast dataset that is commonly used as a computer vision benchmark. Following Monodepth2 [6] we use Eigen et al.'s [39] data split and Zhou et al.'s [8] pre-processing to eliminate static frames from monocular sequences. This generates 39,810 monocular training sequences made up of a three-frame sequence, and 4,424 validation sequences.

We use Monodepth2 [6] architecture to analyze the depth results. The ResNet18 encoder of [6] is replaced with MobileNetv2 architecture. The MobileNetv2 network with decoder and pose architecture is trained on the KITTI data set. Consequently, the network is traversed to perform channel pruning (channel selection and network reconstruction) with the given pruning ratios. This process can result in a loss of significant accuracy. Therefore, the model is fine-tuned to regain accuracy. The pruned network further undergoes quantization where the 32-bit floating-point model is converted to an 8-bit quantized model.

We also use our final fine-tuned model to inference our strategy on the Cityscapes data set [40]. Cityscapes is another dataset containing video data of street scenes. We perform this inference test on the Cityscapes dataset to ensure that our model did not memorize any details pertaining to specific street characteristics within the Kitti dataset.

The quantitative analysis results are presented in Section 4.3. In Section 4.5, we also describe the ablation research comparing the impacts of our various contributions.

4.2 Implementation Details

The proposed algorithm adopts Pytorch framework [41] and the experiments to evaluate the performance of inference are performed using CPU - Intel(R) Core (TM)

i7-6700K CPU @ 4.00GHz, 32 GB Memory. To train and fine-tune our models we make use of GeForce GTX 1080 (Driver Version: 440.100, CUDA Version: 10.2) GPU. The pose and depth network will be jointly optimized using Adam Optimizer [42] with $\beta 1 = 0.9$, $\beta 2 = 0.999$. The training process is executed for 20 epochs with a batch size of 8. The learning rate used for the first 15 epochs is $1e^{-}4$, it is then decayed to $1e^{-}5$ in the remaining epochs. As with the work [6], for training the baseline network, the MobileNetv2 encoder uses pre-trained ImageNet [43] weights as it has shown to reduce training time and increase the overall accuracy of the predicted depths. λ in Equation 5 is taken as $1e^{-}3$. Input images of 640 x 192 pixels will be augmented with 50% probability using horizontal flip, random contrast (± 0.2), saturation (± 0.2), hue jitter (± 0.1), and brightness (± 0.2). Importantly, augmentations are only applied to images that are fed into the depth and pose network, whereas the loss in Equation 5 is calculated using the original ground truth images.

4.3 KITTI Results

The experiments to measure inference characteristics on the KITTI Raw dataset [38] is shown in Table 1 and Table 2. When comparing our method, we observe that we outperform in terms of inference speed and reduced memory utilization compared to the existing self-supervised monocular trained depth estimation by a significant margin with tolerable degradation of performance concerning our use case. It is also important to note that even though the performance is slightly degraded, the numbers fair well than the majority of the previous works.

From the depth maps in Fig. 6, it can be observed that our model performs almost equivalently as the baseline architecture Monodepth2 [6]. We can also observe that our model suffers with thinner structures like poles and signboards and with reflective surfaces. However, since our use case concentrates on warning the user of the impending obstacle, the details like the exact structure of thinner obstacles can be overlooked, since

the riding and avoiding obstacle decisions are made by the human riders. Prediction at reflective surfaces has been a common issue with self-supervised depth estimation as the re-projection error is ill-defined for transparent surfaces. The same is observed with both Monodepth2 and our method.

Table 1
Quantitative Results - Performance. Loss metric (Abs Rel, Sq Rel, RMSE, RMSE log) - *lower is better* and accuracy metrics (δ) - *higher is better*.

Method	Abs Rel	Sq Rel	RMSE	RMSE log	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
SfMLearner [8] $^{\perp}$	0.183	1.595	6.709	0.270	0.734	0.902	0.959
DF-Net [9]	0.150	1.124	5.507	0.223	0.806	0.933	0.973
Godard [5]	0.148	1.344	5.927	0.247	0.803	0.922	0.964
EPC++ [44]	0.141	1.029	5.350	0.216	0.816	0.941	0.976
Struct2Depth [10]	0.141	1.026	5.291	0.215	0.816	0.945	0.979
Monodepth2 [6]	0.115	0.903	4.863	0.193	0.877	0.959	0.981
Adrian [7]	0.106	0.861	4.699	0.185	0.889	0.962	0.982
Ours	0.132	1.016	5.172	0.208	0.840	0.949	0.979

$\perp$ indicates that the new results from github is considered.

Table 2
Quantitative Results - Resource utilization

Method	CPU Inference time(ms)	Memory(MB)
SfMLearner [8]	119.82 ± 4.40	132.80
DF-Net [9]	-	396.9
Godard [5]	1070.44 ± 5.04	379.2
EPC++ [44]	-	146.10
Struct2Depth [10]	-	67.00
Monodepth2 [6]	103.91 ± 1.19	59.43
Adrian [7]	6108.50 ± 12.23	252.7
Ours	$\mathbf{44.39 \pm 1.01}$	**3.29**

4.4 Cityscapes Results

Table 3 and Table 4 presents the quantitative results on Cityscapes dataset [40] using the model trained on the KITTI Raw dataset. It can be observed from the tables and Fig. 7 that our algorithm achieves superior results in terms of inference time and memory

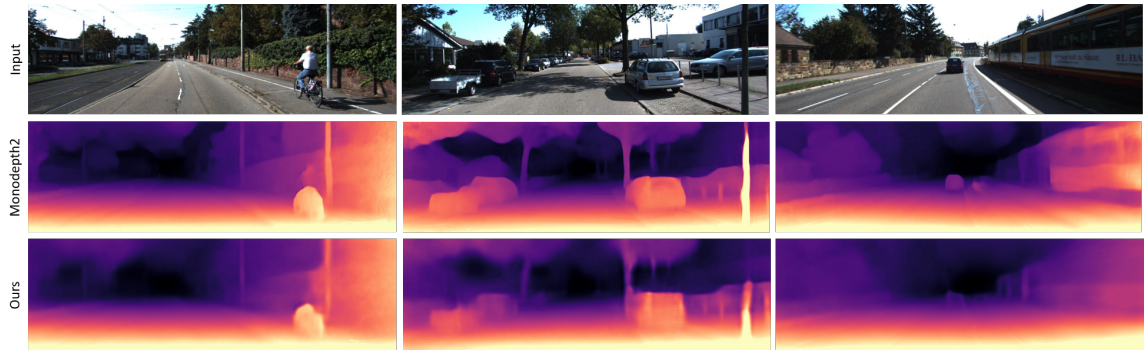

Fig. 6. **Qualitative results on KITTI Raw test set**. From the figures above we can confirm that our model has performed almost equivalently as the baseline architecture Monodepth2 [6].

utilization than the other methods that rely on monocular self-supervised depth estimation with minor loss of performance.

Table 3
Cityscapes Results - Performance. Loss metric (Abs Rel, Sq Rel, RMSE, RMSE log) - *lower is better* and accuracy metrics (δ) - *higher is better*. Q - Quantization

Method	Abs Rel	Sq Rel	RMSE	RMSE log	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Adrian [7]	0.232	2.870	11.206	0.326	0.587	0.852	0.943
Monodepth2 [6]	0.256	3.220	11.634	0.347	0.540	0.829	0.934
MobileNetv2 Baseline	0.247	3.195	11.705	0.343	0.559	0.833	0.934
Ours w/o Q	0.260	3.460	12.286	0.361	0.531	0.810	0.922
Ours	0.272	3.712	12.746	0.377	0.515	0.792	0.910

4.5 Ablation Study

Table 5 and Table 6 shows the experiments carried out in our research. We start from Monodepth2 [6] depth architecture which has ResNet18 as the encoder. Although Adrian et al.'s [7] performance is the current state-of-the-art with the best-achieved performance in the field of unsupervised monocular trained depth estimation, their model size and inference time are high when compared to Monodepth2. Since, our use case is to

Table 4
Cityscapes Results - Resource utilization. Q - Quantization, P - Pruning

Method	CPU Inference time(ms)	Memory(MB)
Adrian [7]	6109.23 ± 14.01	252.7
Monodepth2 [6]	106.55 ± 4.07	59.43
MobileNetv2 Baseline	78.26 ± 1.45	17.72
Ours w/o Q	57.61 ± 0.84	8.41
Ours	45.46 ± 1.44	3.29

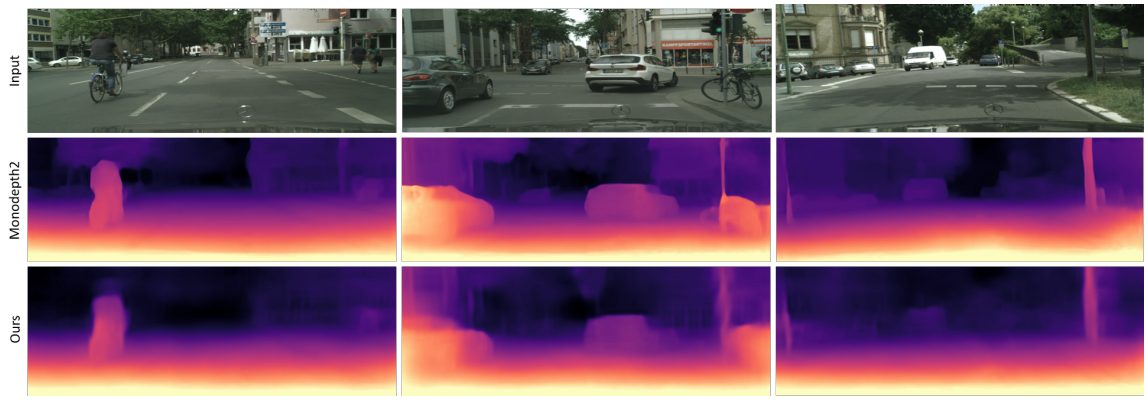

Fig. 7. **Qualitative results on CityScapes test set**. Our model has performed almost equivalently as the baseline architecture Monodepth2 [6], thus confirming that generalization was achieved.

supervise the humans to enhance safety measures and not for self-supervised use cases like autonomous driving, some compromise in the performance is tolerable.

As part of the ablation study, we first perform quantization on the Monodepth2 network to record its performance. Then we prune the original network with different pruning rates, followed by applying quantization on this pruned network. We observe steady improvement in the CPU inference time and memory utilization with tolerable degradation in the performance. These experiments are again repeated by switching ResNet18 with MobileNetv2.

Table 5

Ablation Study - Performance. Results of different experiments on Monodepth2 [6] (ResNet18) and depth network with MobileNetv2 are recorded. Pruning rate is an hyperparameter used to prune channels of convolutional layers at stage*. Loss metric (Abs Rel, Sq Rel, RMSE, RMSE log) - *lower is better* and accuracy metrics (δ) - *higher is better*. CP - Conservative Pruning, AP - Aggressive Pruning

Method	stage1	stage2	stage3	stage4	Abs Rel	Sq Rel	RMSE	RMSE log	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Encoder backbone - ResNet18 [6]											
Baseline	0	0	0	0	0.115	0.903	4.863	0.193	0.877	0.959	0.981
Baseline+Quantization	0	0	0	0	0.118	0.895	4.847	0.196	0.870	0.958	0.981
Conservative Pruning	10	20	20	30	0.118	0.908	4.879	0.197	0.869	0.958	0.980
CP+Quantization	10	20	20	30	0.118	0.912	4.886	0.197	0.869	0.958	0.980
Aggressive Pruning	20	30	30	50	0.120	0.931	4.918	0.198	0.868	0.958	0.980
AP+Quantization	20	30	30	50	0.120	0.927	4.915	0.199	0.867	0.958	0.980
Encoder backbone - MobileNetv2											
Baseline	0	0	0	0	0.120	0.916	4.937	0.196	0.865	0.957	0.981
Baseline+Quantization	0	0	0	0	0.125	0.906	5.158	0.204	0.845	0.950	0.980
Conservative Pruning	10	20	20	30	0.123	1.001	5.043	0.200	0.861	0.956	0.980
CP+Quantization	10	20	20	30	0.135	1.133	5.403	0.214	0.835	0.945	0.977
Aggressive Pruning	20	30	30	50	0.125	0.958	5.021	0.201	0.856	0.954	0.980
AP+Quantization	**20**	**30**	**30**	**50**	**0.132**	**1.016**	**5.172**	**0.208**	**0.840**	**0.949**	**0.979**

Table 6
Ablation Study - Computational Footprint. CP - Conservative Pruning, AP - Aggressive Pruning

Method	stage1	stage2	stage3	stage4	CPU Inference time(ms)	Memory(MB)	Parameters(Millions)
Encoder backbone - ResNet18 [6]							
Baseline	0	0	0	0	103.91 ± 1.19	59.43	14.84
Baseline+Quantization	0	0	0	0	78.85 ± 1.08	24.43	14.84
Conservative Pruning	10	20	20	30	93.60 ± 2.49	37.95	9.47
CP+Quantization	10	20	20	30	70.21 ± 1.04	15.45	9.47
Aggressive Pruning	20	30	30	50	82.73 ± 3.48	27.35	6.81
AP+Quantization	20	30	30	50	62.72 ± 2.73	11.18	6.81
Encoder backbone - MobileNetv2							
Baseline	0	0	0	0	77.87 ± 5.09	17.72	4.38
Baseline+Quantization	0	0	0	0	60.75 ± 1.85	7.39	4.38
Conservative Pruning	10	20	20	30	65.21 ± 4.47	10.75	2.65
CP+Quantization	10	20	20	30	50.53 ± 1.15	4.46	2.65
Aggressive Pruning	20	30	30	50	57.82 ± 0.63	8.41	2.07
AP+Quantization	**20**	**30**	**30**	**50**	$\mathbf{44.39 \pm 1.01}$	**3.29**	**2.07**

The baseline architecture with MobileNet has achieved significant speedup with reduced memory usage. Further, this network is pruned at different rates and then quantization is performed. We define conservative pruning, when 10, 20, 20, 30 percent of channels are removed from stage1, stage2, stage3, and stage4 respectively. And aggressive pruning is when 20, 30, 30, 50 percent channels are removed incrementally at different stages. A stage in an encoder is a collection of convolution layers, batch normalizations, and activation layers that has the same output resolution at each of its layers.

The last row in the Table 5 and Table 6 gives the optimal results with 57.3% speed up in inference time and 94.5% memory savings. Therefore, this model may be adopted as the algorithm to be embedded on the smart helmets. The observed degradation in performance is elaborated in Section 6.

5 DISCUSSIONS

Previous studies [45] have shown to reduce computation footprint with help of the combination of pruning and quantization techniques. The same is validated with the below experiments. In our research, we leverage the combining technique and adapt them for a complex encoder-decoder architecture like the depth estimation network to achieve results with a considerable reduction of inference time and memory bandwidth.

To evaluate this hypothesis, we perform simple experiments using classification models - ResNet18, ResNet50 [22], VGG16 [22] and InceptionV3 [23] - on datasets like CIFAR [46], SVHN [47], and MNIST [48] to demonstrate the benefits of combining both pruning and quantization.

For these experiments, we use a batch size of 64 and the models were trained for 100 epochs on an input image of height and width of 32 pixels. The saved model weights are the ones that performed the best in the 100 epochs. The incremental pruning rates were assigned to convolution layers of different stages. For conservative pruning, the pruning rates are 10, 20, 20, 30 percent applied across stage1, stage2, stage3, and stage4 of the network and for aggressive pruning, it is 30, 40, 40, 50 percent across different stages.

From Fig. 8, 9 and 10, it can be interpreted that pruning and quantization, when used together, will drastically reduce the computation footprint. Also, when conservative pruning and quantization are performed individually on a classification model, it is observed that quantization performs better than pruning in terms of inference time. This is because the memory size is drastically reduced with quantization. After all, quantization converts floating-point values to fixed-point values. Whereas aggressive pruning performs better than quantization when performed individually on the original network, due to a drastic reduction in the number of channels of a convolution neural network. It is also important to note that in complex models like InceptionV3 and ResNet50 the accuracy of the pruned model is more than that of the original model thus validating that deep neural

networks often overparameterize and therefore can be safely pruned to improve accuracy, reduce the model size and inference time.

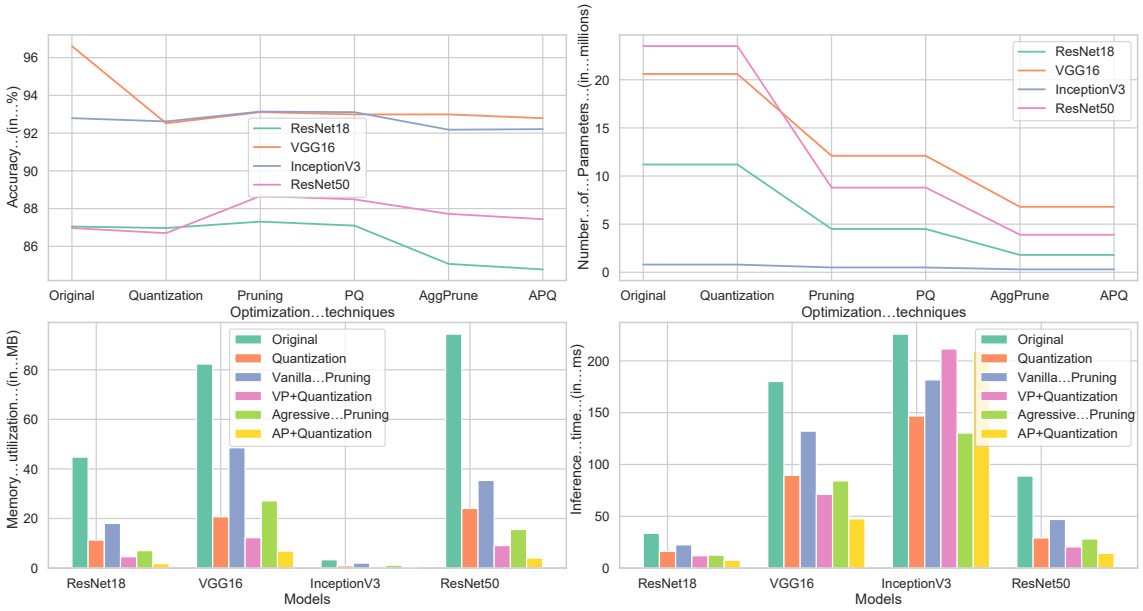

Fig. 8. **Experiments on CIFAR10 dataset**. Figure compares the accuracy, number of parameters, inference time and memory utilization of various neural network architectures on CIFAR10 dataset.CP refers to conservative pruning and AP refers to aggressive pruning. PQ is conservative pruning + quantization and APQ is aggressive pruning + quantization.

However, similar results of reduced inference time are not observed in a complex model like InceptionV3. As observed in the last plot of Fig. 8, 9, 10, an anomaly is seen when pruning and quantization are performed together. As proposed in [49], this anomaly may be due to the overhead posed by the dequantization process in complex networks. However, this doesn't explain the results obtained when quantization alone is performed on this network. This stimulates the need for additional research on the anomaly observed in InceptionV3 when both quantization and pruning are used in unison, and we carry over these experiments for our future work.

Overall, the results indicate that pruning (both conservative and aggressive) followed by 8-bit quantization improves inference time with tolerable loss in accuracy. In this

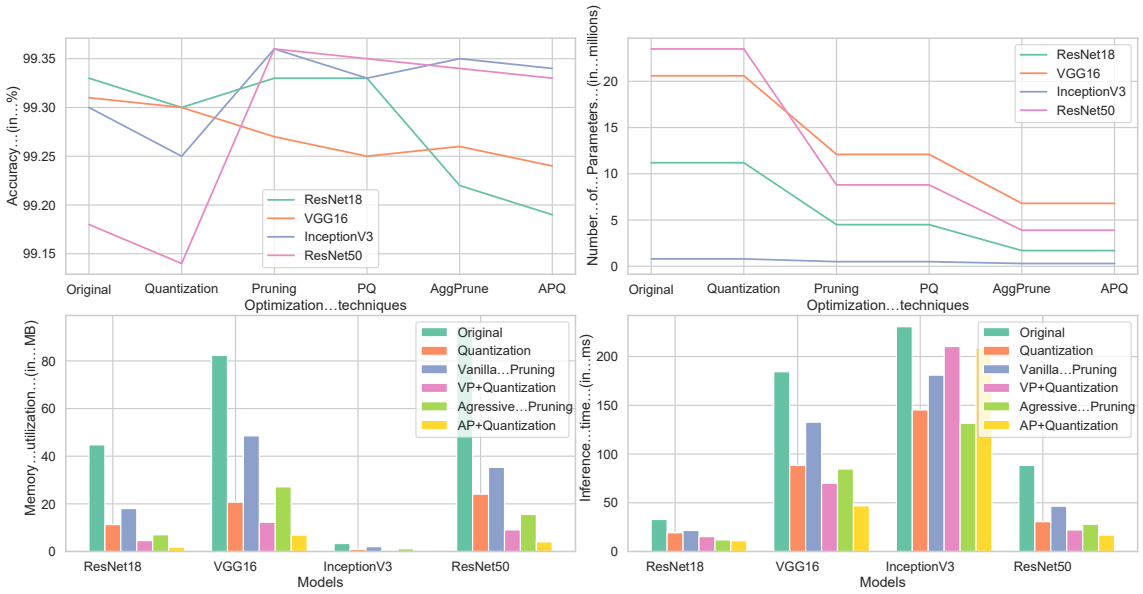

Fig. 9. **Experiments on MNIST dataset**. Figure compares the accuracy, number of parameters, inference time and memory utilization of various neural network architectures on MNIST dataset.

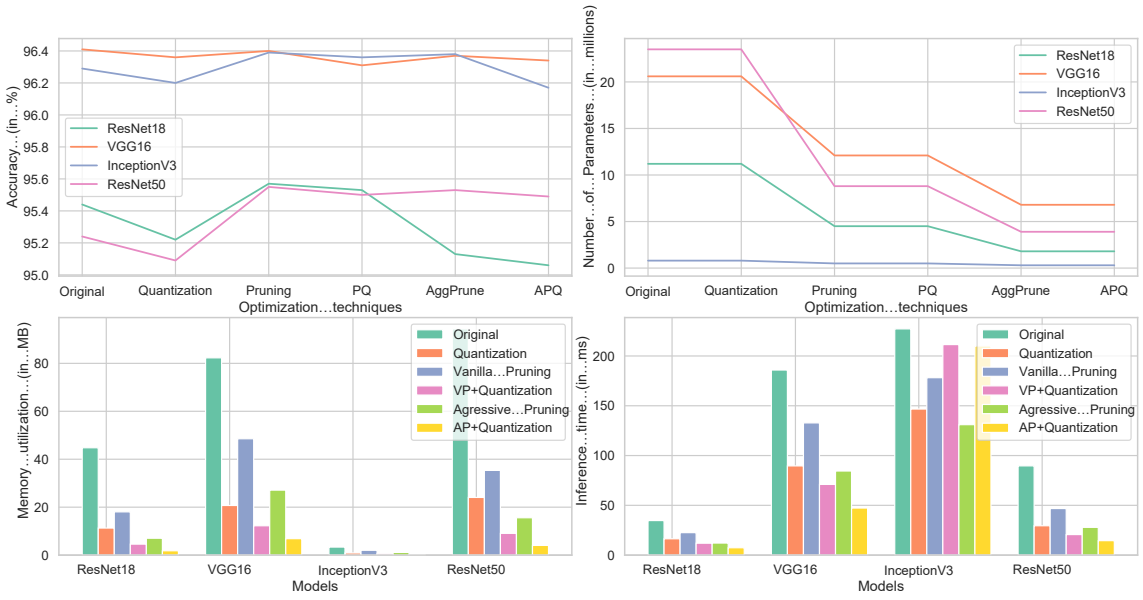

Fig. 10. **Experiments on SVHN dataset**. Figure compares the accuracy, number of parameters, inference time and memory utilization of various neural network architectures on SVHN dataset.

research, we define accuracy as the efficiency of the depth network to predict the depth at an object level or the efficiency of predicting an obstacle itself. Therefore, the loss of accuracy at the pixel level is considered tolerable as a human is involved rather than a machine to detect obstacles.

6 PROPOSED METRICS

As mentioned previously, the performance demands are quite different for the application at hand. In this application, the result of depth estimation is used to only assist a human, and more importantly, the decision-making task is held by a human rather than a machine/algorithm. Therefore, we can make a few compromises on the performance of the depth estimation. Specifically, we do not require high accuracy at the pixel level. Whereas, we require high accuracy at the object level. Therefore, we propose a metric to measure object-level performance in such applications.

$$G_j = agg(p_1, p_2, ..., p_i, ..., p_n) \tag{10}$$

Instead of calculating loss and accuracy at a pixel level (Fig. 11), we propose to divide the images into N grids with each having n pixels as shown in Fig. 12. Let p_i be the value of the i^{th} pixel in the grid. First, we extract the aggregate (agg = mean or min or max) value G_j of the n pixels in a grid (Equation 10). Upon calculating the aggregate values of all the grids in the image we get an array $G_{img} = [G_1, G_2, ..., G_j, ..., G_N]$. This array for predicted depth map G_{pred} is compared with that of the ground truth depth map G_{truth}. The comparison provides a coarser performance measure that will more aptly evaluate the object-level performance needed in the e-Scooter application.

For our experiments, we have taken a grid size of 16 and compared the performance metrics with mean, min, and max as aggregation methods. Table 7, Table 8 and Table 9 records these experiments. However, since our objective is to detect nearby objects correctly, we believe that taking the minimum value of the pixels in the grid of a depth map is more suitable. Taking maximum will lead to accounting for the faraway objects which are unimportant for our use-case. While taking mean can suppress the importance

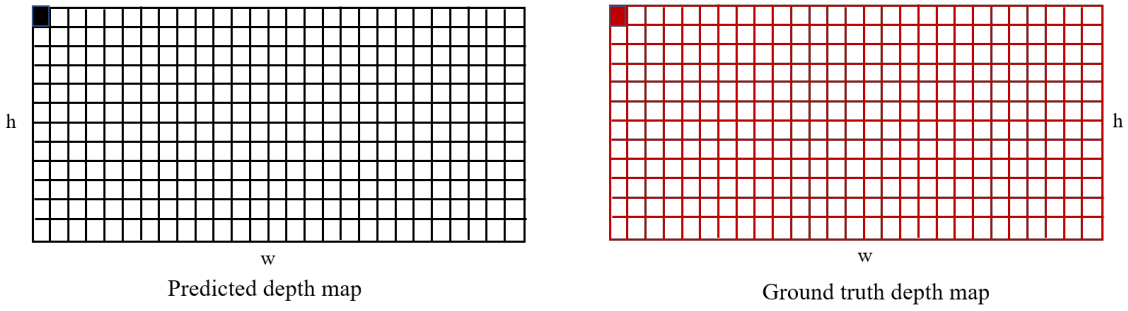

Fig. 11. **Pixel-level performance evaluation**. A pixel in predicted depth map is compared with the corresponding pixel in the ground-truth depth map.

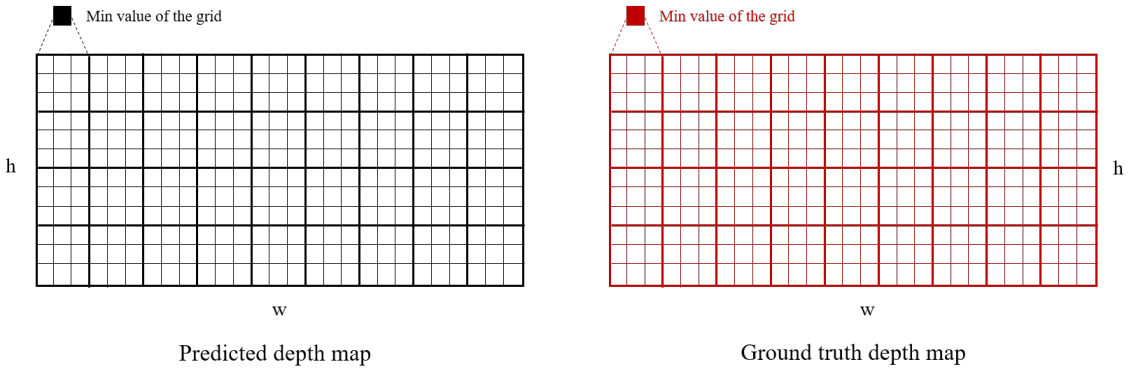

Fig. 12. **Coarser grained performance evaluation**. Here images are divided into grids (in our experiments grid size used is 16x16 pixels). The minimum value of a grid in predicted depth map is compared with the minimum value of the corresponding grid in the ground-truth depth map.

of pixels that have closer depth. Conservative pruning and aggressive pruning are defined by the same pruning rate as in Table 5.

Thus, our optimized model has achieved 57.3% speed up in inference time, 94.5% memory savings and 3.8% improvement in object level performance when compared with the state-of-the-art Monodepth2 [6] model. The comparison of performance is performed using Table 7 and rmse loss values.

Table 7

Object Level Performance Performance measured when minimum value of the grid is chosen. Loss metric (Abs Rel, Sq Rel, RMSE, RMSE log) - *lower is better* and accuracy metrics (δ) - *higher is better*.

Method	Abs Rel	Sq Rel	RMSE	RMSE log	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Encoder backbone - ResNet18 [6]							
Baseline	0.161	2.489	2.292	0.111	0.863	0.923	0.959
Baseline+Quantization	0.159	2.417	2.296	0.110	0.863	0.922	0.963
Conservative Pruning	0.157	2.361	2.226	0.107	0.867	0.932	0.962
CP+Quantization	0.157	2.350	2.246	0.108	0.865	0.931	0.963
Aggressive Pruning	0.163	2.513	2.302	0.113	0.857	0.928	0.960
AP+Quantization	0.162	2.487	2.301	0.113	0.856	0.926	0.957
Encoder backbone - MobileNetv2							
Baseline	0.154	2.240	2.136	0.106	0.872	0.934	0.963
Baseline+Quantization	0.155	2.308	2.133	0.104	0.881	0.934	0.962
Conservative Pruning	0.152	2.222	2.122	0.104	0.881	0.934	0.960
CP+Quantization	0.158	2.424	2.177	0.106	0.873	0.932	0.963
Aggressive Pruning	0.160	2.428	2.219	0.107	0.876	0.931	0.963
AP+Quantization	**0.158**	**2.388**	**2.203**	**0.106**	**0.875**	**0.929**	**0.960**

Table 8

Object Level Performance Performance measured when maximum value of the grid is chosen. Loss metric (Abs Rel, Sq Rel, RMSE, RMSE log) - *lower is better* and accuracy metrics (δ) - *higher is better*.

Method	Abs Rel	Sq Rel	RMSE	RMSE log	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Encoder backbone - ResNet18 [6]							
Baseline	0.098	1.070	2.933	0.110	0.848	0.940	0.975
Baseline+Quantization	0.098	1.075	2.958	0.110	0.844	0.940	0.973
Conservative Pruning	0.096	1.031	2.925	0.109	0.844	0.943	0.973
CP+Quantization	0.098	1.093	2.964	0.110	0.845	0.940	0.973
Aggressive Pruning	0.097	1.170	2.909	0.107	0.848	0.941	0.975
AP+Quantization	0.099	1.198	2.977	0.109	0.847	0.940	0.975
Encoder backbone - MobileNetv2							
Baseline	0.092	0.850	2.759	0.106	0.845	0.946	0.978
Baseline+Quantization	0.089	0.821	2.654	0.105	0.850	0.944	0.976
Conservative Pruning	0.094	0.904	2.857	0.109	0.847	0.941	0.973
CP+Quantization	0.091	0.833	2.732	0.106	0.845	0.943	0.975
Aggressive Pruning	0.091	0.859	2.703	0.106	0.850	0.944	0.973
AP+Quantization	**0.090**	**0.859**	**2.704**	**0.106**	**0.845**	**0.941**	**0.975**

Table 9

Object Level Performance Performance measured when mean value of the grid is chosen. Loss metric (Abs Rel, Sq Rel, RMSE, RMSE log) - *lower is better* and accuracy metrics (δ) - *higher is better*.

Method	Abs Rel	Sq Rel	RMSE	RMSE log	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Encoder backbone - ResNet18 [6]							
Baseline	0.042	0.202	0.992	0.042	0.963	0.990	1.000
Baseline+Quantization	0.043	0.217	1.011	0.042	0.963	0.988	0.999
Conservative Pruning	0.041	0.193	0.976	0.041	0.962	0.990	1.000
CP+Quantization	0.042	0.212	0.989	0.042	0.962	0.990	0.999
Aggressive Pruning	0.042	0.203	0.990	0.042	0.960	0.991	1.000
AP+Quantization	0.043	0.206	1.003	0.043	0.957	0.991	1.000
Encoder backbone - MobileNetv2							
Baseline 0.039	0.159	0.917	0.040	0.968	0.993	1.000	
Baseline+Quantization	0.038	0.156	0.876	0.038	0.968	0.993	1.000
Conservative Pruning	0.040	0.176	0.934	0.040	0.963	0.991	1.000
CP+Quantization	0.038	0.175	0.898	0.039	0.963	0.991	1.000
Aggressive Pruning	0.038	0.150	0.876	0.039	0.963	0.993	1.000
AP+Quantization	**0.038**	**0.150**	**0.878**	**0.039**	**0.965**	**0.991**	**1.000**

7 FUTURE WORK

There are still several unanswered questions in this research. First, we need a better way to quantify the real-time deadlines for depth estimation. Although various works discuss these real-time deadlines [50]–[52], there is no one standard real-time deadline defined for these use-cases. This is mostly because the deadlines are dependent on the environment, which is dynamic (example: speed of the rider). Second, there is a need to study how the system should deal with stale data and how to effectively manage resources at the edge.

In our work, we proposed coarse-grained evaluation metrics that can be used to assess the performance of the depth estimation algorithm on an image. However, since our use-case predominantly concentrates on identifying closer objects, we can extend the coarse-grained metrics to be evaluated only on the pixels of the closer objects rather than on the entire image. This could be considered more suitable for our use case and it is the future direction of this research.

Another experiment includes profiling computationally diverse processing devices to study their impact on the inference time and computational footprint.

All these questions and improvements are the directions for our future work.

8 CONCLUSION

This work is motivated by the increased number of road accidents involving micro-mobility vehicles such as electric scooters. We envision that a smart helmet equipped with the capability to provide a real-time assessment of the street scene can potentially avoid accidents. Monocular depth estimation algorithms are a promising solution for this application. However, the current algorithms are too computationally heavy to run on a resource-constrained edge device in real-time. A smart helmet must be able to present results in the order of a few milliseconds to be considered an effective tool for e-Scooter riders. This is achieved by optimizing the depth network through efficient convolution neural network, quantization, and neural network introspection as proposed in the research. These techniques help in reducing the CPU inference time and compute footprint without a substantial decrease in accuracy. This allows for the efficient deployment of a depth network on an edge device.

This research presents extensive experimental results on different depth estimation algorithms by applying the aforementioned techniques to optimize performance at the edge. We observe that using MobileNetv2 provided significant improvement. This followed by channel pruning and 8-bit quantization are an effective way to reduce inference time with only a slight decrease in the depth estimation error. Qualitatively, we can tolerate the performance drop since our depth maps are consumed by humans and not machines. We also introduce object-level performance metrics that are essential for such use cases. We observed that when evaluated at an object-level, our method outperformed the baseline Monodepth2 [6].

Literature Cited

[1] B. Preston, "New study shows safety risks of riding e-scooters on the sidewalk," https://www.consumerreports.org/electric-scooters/safety-risks-of-riding-e-scooters-on-the-sidewalk-iihs-study/, (accessed Nov 1, 2020).

[2] H. Fu, M. Gong, C. Wang, K. Batmanghelich, and D. Tao, "Deep ordinal regression network for monocular depth estimation," in *2018 IEEE/CVF Conf. Computer Vision and Pattern Recognition*, pp. 2002–2011.

[3] X. Guo, H. Li, S. Yi, J. Ren, and X. Wang, "Learning monocular depth by distilling cross-domain stereo networks," 2018, arXiv:1808.06586.

[4] R. Garg, V. K. BG, G. Carneiro, and I. Reid, "Unsupervised cnn for single view depth estimation: Geometry to the rescue," 2016, arXiv:1603.04992.

[5] C. Godard, O. M. Aodha, and G. J. Brostow, "Unsupervised monocular depth estimation with left-right consistency," in *2017 IEEE Conf. Computer Vision and Pattern Recognition*, pp. 6602–6611.

[6] C. Godard, O. M. Aodha, M. Firman, and G. Brostow, "Digging into self-supervised monocular depth estimation," in *2019 IEEE/CVF Int. Conf. Computer Vision*, pp. 3827–3837.

[7] A. Johnston and G. Carneiro, "Self-supervised monocular trained depth estimation using self-attention and discrete disparity volume," in *2020 IEEE/CVF Conf. Computer Vision and Pattern Recognition*, pp. 4755–4764.

[8] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, "Unsupervised learning of depth and ego-motion from video," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, July 2017.

[9] Y. Zou, Z. Luo, and J.-B. Huang, "Df-net: Unsupervised joint learning of depth and flow using cross-task consistency," 2018, arXiv:1809.01649.

[10] V. Casser, S. Pirk, R. Mahjourian, and A. Angelova, "Depth prediction without the sensors: Leveraging structure for unsupervised learning from monocular videos," in *33rd AAAI Conf. Artificial Intelligence*, 2019.

[11] L. Mao, "Quantization for neural netoworks," https://leimao.github.io/article/Neural-Networks-Quantization/, (accessed Feb 21, 2020).

[12] J. Ma, Z. Zhu, L. Dai, and S. Guo, "Layer-by-layer quantization method for neural network parameters," in *Proc. Int. Conf. Industrial Control Network and System Engineering Research*, 2019, pp. 22—26.

[13] Y. Choukroun, E. Kravchik, F. Yang, and P. Kisilev, "Low-bit quantization of neural networks for efficient inference," in *2019 IEEE/CVF Int. Conf. Computer Vision Workshop*, pp. 3009–3018.

[14] L. Mao, "Pytorch static quantization," https://leimao.github.io/blog/PyTorch-Static-Quantization/, (accessed Feb 21, 2020).

[15] M. Lin, R. Ji, Y. Zhang, B. Zhang, Y. Wu, and Y. Tian, "Channel pruning via automatic structure search," in *Proc. 29th Int. Joint Conf. Artificial Intelligence*, 2021, arXiv:2001.08565.

[16] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Proc. 28th Int. Conf. Neural Information Processing Systems*, 2015, pp. 1135–1143.

[17] S. Srinivas, A. Subramanya, and R. V. Babu, "Training sparse neural networks," in *2017 IEEE Conf. Computer Vision and Pattern Recognition Workshops*, pp. 455–462.

[18] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," 2017, arXiv:1608.08710.

[19] S. Changpinyo, M. Sandler, and A. Zhmoginov, "The power of sparsity in convolutional neural networks," 2017, arXiv:1702.06257.

[20] T. Wu, X. Li, D. Zhou, N. Li, and J. Shi, "Differential evolution based layer-wise weight pruning for compressing deep neural networks," *Sensors*, vol. 21, no. 3, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/3/880

[21] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *2017 IEEE Int. Conf. Computer Vision*, pp. 2755–2763.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conf. Computer Vision and Pattern Recognition*, pp. 770–778.

[23] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *2016 IEEE Conf. Computer Vision and Pattern Recognition*, pp. 2818–2826.

[24] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," 2018, arXiv:1608.06993.

[25] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," 2018, arXiv:1712.00559.

[26] I. Radosavovic, R. Kosaraju, R. Girshick, K. He, and P. Dollar, "Designing network design spaces," in *2020 IEEE/CVF Conf. Computer Vision and Pattern Recognition*, Jun 2020, pp. 10 425–10 433.

[27] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size," 2016, arXiv:1602.07360.

[28] 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," 2017, arXiv:1704.04861.

[29] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," 2019, arXiv:1801.04381.

[30] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *2018 IEEE/CVF Conf. Computer Vision and Pattern Recognition*, pp. 6848–6856.

[31] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2020, arXiv:1905.11946.

[32] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *2017 IEEE Int. Conf. Computer Vision*, pp. 1398–1406.

[33] "Quantization," *pytorch.org*, accessed: Oct 1, 2020. [Online]. Available: https://pytorch.org/docs/stable/quantization.html

[34] R. Krishnamoorthi, J. Reed, M. Ni, C. Gottbrath, and S. Weidman, "Introduction to quantization on pytorch," https://pytorch.org/blog/introduction-to-quantization-on-pytorch/, (accessed Mar 27, 2020).

[35] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Proc. 28th Int. Conf. Neural Information Processing Systems*, 2015, pp. 3123—-3131.

[36] H. Pouransari, Z. Tu, and O. Tuzel, "Least squares binary quantization of neural networks," in *2020 IEEE/CVF Conf. Computer Vision and Pattern Recognition Workshops*, pp. 2986–2996.

[37] C. Wang, J. M. Buenaposada, R. Zhu, and S. Lucey, "Learning depth from monocular videos using direct methods," in *2018 IEEE/CVF Conf. Computer Vision and Pattern Recognition*, pp. 2022–2030.

[38] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE Conf. Computer Vision and Pattern Recognition*, pp. 3354–3361.

[39] D. Eigen and R. Fergus, "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture," in *2015 IEEE Int. Conf. Computer Vision*, pp. 2650–2658.

[40] 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," 2016, arXiv:1604.01685.

[41] 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," 2017, [Online]. Available: https://openreview.net/pdf?id=BJJsrmfCZ.

[42] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017, arXiv:1412.6980.

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

[44] C. Luo, Z. Yang, P. Wang, Y. Wang, W. Xu, R. Nevatia, and A. Yuille, "Every pixel counts ++: Joint learning of geometry and motion with 3d holistic understanding," 2019, arXiv:1810.06125.

[45] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," 2016, arXiv:1510.00149.

[46] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10," *Canadian Institute for Advanced Research*. [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

[47] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS Workshop Deep Learning and Unsupervised Feature Learning*, 2011, [Online]. Available: http://ufldl.stanford.edu/housenumbers.

[48] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," 1998, pp. 2278–2324.

[49] Q. Qin, J. Ren, J. Yu, H. Wang, L. Gao, J. Zheng, Y. Feng, J. Fang, and Z. Wang, "To compress, or not to compress: Characterizing deep learning model compression for embedded inference," in *2018 IEEE Int. Conf. Parallel Distributed Processing with Applications, Ubiquitous Computing Communications, Big Data Cloud Computing, Social Computing Networking, Sustainable Computing Communications*, 2018, pp. 729–736.

[50] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *25th Annual Int. Conf. Mobile Computing and Networking*, 2019.

[51] T. Braud, P. Zhou, J. Kangasharju, and P. Hui, "Multipath computation offloading for mobile augmented reality," in *2020 IEEE Int. Conf. Pervasive Computing and Communications*, 2020, pp. 1–10.

[52] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-time video analytics: The killer app for edge

computing," in *IEEE Computer*, 2017, [Online]. Available: https://www.microsoft.com/en-us/research/publication/real-time-video-analytics-killer-app-edge-computing/.