2023 8th International Engineering Conference on Renewable Energy & Sustainability (ieCRES) | 979-8-3503-0075-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/ieCRES57315.2023.10209460

The 8th International Engineering Conference on Renewable Energy & Sustainability (ieCRES 2023)

Accelerating the Run-Time of Convolutional Neural Networks through Weight Pruning and Quantization

Rajai Alhimdiat, Wesam Ashour
Computer Engineering dept.
Islamic University of Gaza, IUG
Gaza, Palestine
e-mail: rajaihimdiat@hotmail.com, washour@iugaza.edu.ps

Ramy Battrawy, Didier Stricker
German Research Centre for Artificial Intelligence.
DFKI.
Kaiserslautern, Germany
e-mail: ramy.battrawy@dfki.de, didier.stricker@dfki.de

Abstract*— Accelerating the processing of Convolutional Neural Networks (CNNs) is highly demand in the field of Artificial Intelligence (AI), particularly in computer vision domains. The efficiency of memory resources is crucial in measuring run-time, and weight pruning and quantization techniques have been studied extensively to optimize this efficiency. In this work, we investigate the contribution of these techniques to accelerate a pre-trained CNN model. We adapt the percentile-based weights pruning with focusing on unstructured pruning by dynamically adjusting the pruning thresholds based on the fine-tuning performance of the model. In the same context, we perform uniform quantization for presenting the weights values of the model's parameters with a fixed number of bits. We implement different levels of post-training and aware-training -fine-tuning- the model with the same learning rate and number of epochs as the original. We then refine-tune the model with a lower learning rate and a factor of 10x for both techniques. Finally, we combine the best levels of pruning and quantization and refine-tune the model to explore the best-pruned and quantized pre-trained model. We evaluate each level of the techniques and analyze their trade-offs. Our results demonstrate the effectiveness of our strategy in accelerating the CNN and improving its efficiency, and provide insights into the best combination of techniques to accelerate its inference time.

Keywords: Pruning and Quantization, CNN's Run-Time, Inferences acceleration.

I. INTRODUCTION

The demand for efficient processing of Convolutional Neural Networks (CNNs) in the field of Artificial Intelligence (AI) has led to the development of weight pruning and quantization techniques. These techniques optimize memory resources and reduce run-time, making CNNs more suitable for deployment on devices with limited hardware resources and strict latency requirements [1].

Pruning involves removing unnecessary weights from the CNN by selectively setting weights to zero based on their magnitude [2]–[4]. Quantization reduces the precision of weights and activations by representing them using a smaller number of bits [1], [3], [5]–[9]. Weight pruning and quantization are straightforward strategies to post-process pre-trained models by using fewer bits for memory allocation, which can accelerate the inference time of CNNs [10], [11]. These techniques can be used individually or in combination to reduce the complexity of the CNN and improve its efficiency, without significantly degrading its performance. Combining pruning and quantization can be particularly effective, but the optimal combination of techniques for a given CNN and application [9], [12], [13], must be carefully identified.

Our study investigates the impact of pruning and quantization on the performance of CNN-based solutions. Mainly our study focuses on a solution for scene flow estimation, which is crucial for applications such as autonomous vehicles, robot navigation, and advanced driver assistance systems. Specifically, we examine the DeepLiDARFlow [14] model, which fuses LiDAR measurements and monocular camera inputs for a dense scene flow estimation as 2D representations. This model achieves accurate results, but its large memory footprint and high computational requirements hinder its use in real-time applications.

As the pre-trained model of DeepLiDAFlow [14] is already trained by FlyingThings3D [15] dataset, we do not need to do training from scratch when applying pruning and quantization techniques. Using the KITTI [16] dataset benchmark, we evaluate the performance of the pre-trained DeepLiDARFlow [14] model after applying various levels of pruning and quantization techniques, on both post-training and aware-training. We fine-tune the model using the same learning rate and number of epochs as the original [14] and then refine-tune it with a lower learning rate and a factor of 10x for both techniques. Finally, we combine the best levels of pruning and quantization to obtain the best pruned and quantized pre-trained model.

Our results demonstrate the effectiveness of our approach in improving the efficiency of the CNN and accelerating its inference time, while maintaining high accuracy. Moreover, we provide insights into the trade-offs between pruning and quantization and the best combination of techniques for a given CNN and application. Our work makes several significant contributions to the field of CNN optimization for real-time applications:

- We explore the efficacy of post-training for pruning and quantization techniques on a certain pre-trained CNN solution, fine-tuning it with the same number of epochs and learning rate as the original model.
- We investigate aware-training for pruning and quantization of the CNN's pre-trained model at various levels and different learning rates to identify the optimal

level for achieving the best balance between model size and accuracy.
- We demonstrate the effectiveness of our approach by evaluating it on a real-world dataset, showing that our optimized model achieves a lower memory footprint and runtime than the original model.
- We identify the best quantization level and pruning percentage for the CNN's pre-trained model and provide generalizations that can be applied to other solutions.
- Our findings can inform the development of efficient CNN architectures that can be deployed in resource-constrained environments, such as self-driving cars, where latency and memory constraints are critical factors.

II. Related Works

A. Weight pruning for CNN compression

Weight pruning is a popular technique for reducing the storage and computation requirements of neural networks. It aims to eliminate unnecessary connections between neurons, such as those corresponding to redundant or unimportant features, without sacrificing accuracy [8], [13]. Different studies [13], [17]–[20], have proposed various methods for weight pruning, such as pruning entire filters, channels, or layers, or selectively removing individual weights or filters based on certain criteria. Recently, several researchers have proposed methods for coordinated weight pruning [2], such as structural sparsity learning or group LASSO regression [21], which aim to prune filters, channels, or neurons in a coordinated manner, while others have focused on individual weight pruning, using techniques such as L_1 norm or Taylor expansion to identify and remove unimportant weights or filters [22]–[24]. However, these methods often lead to a significant decrease in accuracy Additionally, some studies have incorporated batch normalization or long short-term memory (LSTM) [25] based decision-making to improve the efficiency and accuracy of weight pruning.

Weight pruning can be categorized as structured or unstructured pruning, depending on whether it leads to changes in the network architecture [10], [13], [17], [18], [26]–[28]. Structured pruning methods aim to remove entire filters, channels, or layers, while unstructured pruning involves selectively removing individual weights or filters. Our work focuses on unstructured pruning, where we adapt a strategy of dynamically adjusting the pruning threshold based on the fine-tuning performance of the network on the dataset used.

Overall, weight pruning is a promising approach for accelerating the run-time of convolutional neural networks while minimizing storage and computation requirements. Different pruning methods and criteria have been proposed, and the choice of pruning strategy may depend on factors such as network architecture, dataset, and performance objectives.

B. Weight quantization for CNN compression

Weight quantization is a technique used to reduce the number of bits required to store pre-trained model parameters and feature values in memory [1], [8], [23], [29], [30]. Various quantization techniques and approaches have been explored to speed up CNNs. Some studies have implemented full-precision quantization to convert low-precision CNN models, while others have attempted to train binary networks directly [30]. Some researchers have explored the use of low rank factorization to reduce the number of calculations and improve inference speed [24], [31]. However, these methods often lead to a significant decrease in accuracy, especially when used on larger networks.

Despite ongoing research, there are still challenges associated with weight quantization in deep neural networks (DNNs) and CNNs. One of these challenges is the accuracy loss that occurs during the quantization process, which often requires an increased number of training iterations and fine-tuning epochs to achieve convergence. In this study, we investigate the use of conscious training quantization and compare the results with post-training quantization in an effort to address the issue of low weight accuracy during quantization.

Quantization involves converting a pre-trained model's full-precision weights (i.e., 32-bit floating-point) into a low-precision version where the weights are constrained to be either a power of two or zero, similar to [1]. This method is advantageous because it reduces the memory required for the original floating-point multiplication operation. During either post-quantization or aware-quantization, very small weight values are pruned based on a percentile threshold. Our experimental results show the percentage of pruning that occurs during the quantization process. This combination of quantization and pruning can accelerate the CNN and decrease the memory footprint, as demonstrated in [3], which inspired us to merge quantization and pruning as part of our approach. Other researchers have also explored merging pruning and quantization techniques to avoid over pruning [12]. Merging these techniques can compress the model storage and improve the model's inference speed. Our proposed method and strategy have been extensively tested, and the results are presented in the experiments and results section.

III. Proposed Method

Convolutional Neural Network implementations on Graphical Processing Units (GPUs) use power-inefficient Floating Operation Points (FLOPS). This paper aims to develop a compressed version of a CNN. To reduce the storage and computational requirements of a CNN by quantizing and pruning its weights and activations, we propose a strategy in which it is hypothesized that quantizing and pruning the weights and activations of a CNN will reduce its storage and computational requirements without significantly degrading its performance.

Our method is summarized according to the following:

1- Identify a CNN solution as available. Therefore, we identify a DeepLiDARFlow [14], which is a model for estimating dense scene flow from monocular camera and sparse LiDAR.
2- Profile the solution and identify the size and computational cost of its components.

3- Since its pre-trained model is available, there is no need to train it from scratch with full precision. We can directly implement the post-training quantization and post-training pruning. Then, it is to fine-tune the pre-trained model with pruning and quantization in aware-training/fine-tuning.
4- Quantize the weights and activations: The quantization is implemented in post and aware training quantization, and comparing the results and performance with the full precision version.
5- Prune the weights and activation: Pruning is to reduce the memory footprint by pruning unnecessary weight values. Similar to quantization procedures, pruning is also implemented in post and aware training, and comparing the results with full-precision version.
6- Evaluate and compare the performance for each with the original version.
7- The experimental procedures are followed, where the quantized version and pruned versions of the CNN are fine-tuned and are evaluated on the same dataset to assess the impact of both techniques on performance.

A. Profiling CNN solution – DeepLiDARFlow

In this work, we notice that state-of-the-art basic architectures such as DeepLiDARFlow [14] has becomes less efficient in extremely small networks because of the costly dense convolution. However, it is one of the state-of-the-art in the way of fusing LiDAR measurement data with monocular camera setups for scene flow estimation as 2D representations. It is worth implementing compression through pruning and quantization to accelerate its inference time to be convenient with real-time use.

We start our work by profiling the DeepLiARFlow [14] model. Profiling is necessary to study the modules that are on a high budget. In order to determine the road map of our work and which modules or components can be targeted. This profiling establishes the model components and their budget costs for each. Table I summarizes the modules and components of DeepLiDARFlow [14], parameters, FLOPS, and inference time for the entire model. Model parameters, which are in general weights that are learnt during training, are measured according to [32] as shown in hereunder equations. However, they are weight matrices that contribute to the model's predication power, changed during the back-propagation process. FLOPS are also measured based on the equations [28]. This process aims to overview which components occupy a high budget for running.

For measuring Convolutional layers' parameters, it is satisfied as in (1):

$$Conv_{parameters} = \sum_{l=1}^{l}((m^{l} * n^{l} * d^{(l-1)}) + 1) * k^{l}. \quad (1)$$

Where m is the shape of width, n is the height, d is the layer's filters, l is layer number and k is the account for all such filters in the current layer, 1 is added as a bias term for each filter. When l = 0, as it is, the input layer has nothing to learn, as its core, which provides the input images' shape.

Thus, the number of parameters = 0. Hence, Pool Layer: generally, the pooling layer has no backpropagation learning involved, thus the number of its parameters = 0.

TABLE I. DEEPLIDARFLOW MODEL SUMMARY PROFILE.

Item	FE Module		SF Module	CN Module	CV Module	Warping Module	Total
	FPN	FM					
Number of parameters	**3,849,028 (% 46.40)**	1,006,720 (% 12.14)	2,916,852 (% 35.16)	522,436 (% 6.30)	0	0	8,295,036
Number of CNN layers	**240 (% 61.70)**	65 (% 16.7)	76 (% 19.5)	8 (% 2.1)	0	0	389
FLOPs	33.45 B. (% 24.92)	24.50 B. (% 18.25)	**45.38 B. (% 33.81)**	30.48 B. (% 22.71)	0.406 B. (% 0.3)	0.011 B. (% 0.01)	134.227 B.
Inference time	137.4 ms = 0.1374 seconds						

FE= Feature Extraction Module, **FPN**= Feature Pyramids Network, **FM**= Fusion Model, **SF**= Scene Flow Module, **CN**= Context Network Module, **CV**= Cost Volume Module, ms. = 10^{-3} seconds, and **B**= Billion. Yellow marker means the highest cost. Inference time is computed using GPU GTX 1660. 6. Giga Byte (GB).

Additionally, for measuring the Fully Connected (FC) layer, since every neuron has been connected to every other neuron for the next layer, it has the highest number of parameters. Hence, the number of parameters for FC is satisfied as in (2):

$$FC_{parameters} = \sum_{l=1}^{l}(((c^{l} * p^{(l-1)}) + 1) * c^{l}). \quad (2)$$

Where, c is the current layer neurons, p is previous layer neuron. Equation (2) is also used to measure learnable parameters for soft max layers.

The convolution is implemented by a sliding window on the layer, for computing the FLOPS from the following equation (3):

$$Conv_{FLOPS} = \sum_{l=1}^{l}(2HW(c_{in}K^{2} + 1) * c_{out}). \quad (3)$$

Where:

- H,W and c_{in} are height, width and number of channels of the feature map respectively.
- K is the kernel width.
- l is the layer.
- c_{out} is the number of output channels.

For measuring FLOPS of fully connected layers is satisfied as in (4):

$$FC_{FLOPS} = \sum_{l=1}^{l} 2 * size_{input} * size_{out}). \quad (4)$$

In computing, FLOPS is a measure of computer performance, useful in fields of scientific computations that require floating-point calculations. For such cases, it is a more accurate measure than measuring instructions per second. The size means the size of feature maps.

Table I. shows that the current CNN has hundreds of layers, thousands of channels and millions of parameters, thus requiring computation at billions of FLOPS. This report examines the opposite extreme: pursing the best accuracy in very limited computational budgets, focusing on common computer vision high-level tasks and platforms such as robots.

The following section previews the quantization and pruning strategy which are implemented on the DeepLiDARFlow [14] model and its pre-trained model.

B. Pruning

Weight pruning is a technique for reducing the number of parameters in a neural network by removing weights that are not important for the model's performance [3], [9]. We adapt the Percentile-based weight pruning, which involves

identifying the weights in the network that are within a certain percentile range of the distribution of weights, and then setting these weights to zero [28], [33].

One advantage of percentile-based weight pruning is that it is flexible, as the percentile threshold can be adjusted to control the amount of pruning as is done.

To perform percentile-based weight pruning, we generally follow these steps: we compute the distribution of weights in the network. Then identify the percentile threshold that we want to use for pruning in both post and aware training. For example, we may want to prune the lowest 10% of weights. We identify the range of weights that corresponds to the percentile threshold. If we are using a 10% threshold, we identify the range of weights that corresponds to the lowest 10% of the distribution. Finally, we set all of the weights within the identified range to zero.

Notably, weight pruning can also negatively affect the performance of the model if the pruning is not done carefully. For example, if the pruning threshold is set too high, the model may not be able to learn effectively, as it will have fewer weights available to use. Therefore, the model may not be able to learn effectively. On the other hand, if the pruning threshold is set too low, the model may not be able to achieve a significant reduction in the number of weights. As such, the percentile threshold to find the right balance between model complexity and performance, so careful tuning is necessary.

In general, pruning too many weights or biases can degrade the accuracy of the CNN, so careful evaluation is necessary. It may also be necessary to fine-tune the CNN after pruning it further to improve its performance. This is detailed in Experiments and Results section IV. Therefore, we implement post-pruning experiments and compare the experiments with the original version of DeepLiDARFlow [14] pre-trained model. Then we apply aware-training pruning by fine-tuning the model using the original pre-trained model version, using the same datasets and learning as [14]. Then, following the same procedures which are applied in quantization, the same thing in aware-training for pruning, using the same the *lr* and epochs to compare each of the pruning aware training and quantization aware-training and its effects on accuracy. The run-time is also measured and compared with the original, to get a better picture of the impact of each of what is applied on the model's performance. Hence, these results can be generalized to other similar models targeting compression and reducing memory allocation.

C. Quantization

There are several ways to decrease the number of bits in a Convolutional Neural Network. One approach is to use low-precision data types, such as 8-bit integer (int8) or 16-bit floating point (float16), to represent the weights and activations. These data types use fewer bits than the standard 32-bit floating point (float32) data type, which is typically used to represent weights and activations in deep learning models.

Another approach is to use quantization techniques, which involve approximating the values of the weights and activations with values from a smaller set of discrete values. For example, weight quantization involves rounding the weights to the nearest value in a set of discrete values, while activation quantization involves rounding the activations to the nearest value in a set of discrete values.

We follow uniform quantization, which is a technique for representing the weights and activations of a CNN with a fixed number of bits. It works by dividing the range of possible values into a fixed number of equally sized intervals, and assigning each interval a unique quantization level. This is the following by rounding the weights to the nearest value. We generally follow these steps: we determine the range of values that we want to quantize. This will typically be the minimum and maximum values of the data that are quantized. We then choose the number of discrete values that we want to use for quantization. This will typically be a power of 2, as this allows the quantized values to be represented using a fixed number of bits. Further, we divide the range of values into intervals of equal size. The number of discrete values, which are used for quantization, determines the size of the intervals. Then, we assign a quantized value to each interval based on the midpoint of the interval. For example, if the first interval has a midpoint of 9.5, we could assign the value 10 to this interval, and so on. Finally, for each data point, determine the interval that it falls within, and replace the original value of the data point with the quantized value for that interval. However, this is also followed by aware training/fine-tuning to improve the accuracy of results.

Notably, uniform quantization can introduce some loss of accuracy, as the original values of the data points are approximated with the quantized values. As such, tuning carefully the number of discrete values and the size of the intervals, to find the right balance between accuracy and the number of bits needed to represent the data. Therefore, in the first case, we apply the quantization directly to the pre-trained model and save a new version of the quantized pre-trained model for each percentage.

Then we investigate the experiments and compare those with a full-precision version. The second case, aware-training quantization, we apply quantization during fine-tuning once with the same number of epochs and learning rate (*lr*) stated in [14], and others using lower *lr* and more epochs for fine-tuning to improve the accuracy, which lost during the quantization versions.

IV. Experments and Results

Several experiments are conducted to verify the results of our method. As shown in section E, ablation study, we ablate the process by implementing evaluation first to the original version of DeepLiDARFlow [14] using the same dataset mentioned in [14], which is detailed below in section A.

Second, we ablate a few steps of post and aware-training for both pruning and quantization. Third, we conduct several training implementations for aware-training quantization and pruning. Finally, we implement several experimental results compared to the original. We start this section overviewing the datasets and evaluating metrics used in our experiments.

The pruning is accomplished in two phases; which is post-training pruning and aware-training pruning in different percentages. Similarly, the same approach for quantization is applied in post-training quantization, and aware-training quantization. However, we apply quantization using (16 bits, 14 bits, 12 bits, 8 bits, and 6 bits) for both post and aware-training quantization.

A. Datasets and Evaluation Metrics

Following DeepLiDARFlow [14] instructions for fine-tuning on the KITTI dataset.

KITTI [16]. This benchmark, known as the KITTI Scene Flow benchmark, is used to assess scene flow techniques. There are 200 training and 200 test scenes. We follow DeepLiDARFlow [14] preprocessing for the KITTI dataset, referred to as $\mathbf{KITTI_d}$.

The same percentage of dividing the dataset used in [14], however, 9:1 of the frames as training : testing.

We use the same evaluation metrics used in [14]. In this context, we evaluate the accuracy using End Point Error (***EPE)***, the endpoint error is > 3 pixels, and the relative error is > 5 % compared to the ground truth. Additionally, the KITTI outlier rate for scene flow (***KEO)*** is mentioned in this study, mainly in the ablation study.

Further to compute the run-time, we perform the evaluation on a computer with GPU-GeForce GTX 1660Ti 6 Giga byte concerning the same environment, same dataset and same frames of input.

B. Implementation

Furthering to the ablation study and the process to ablate our strategy; we implement post-quantization, to reduce memory usage by quantizing the model on various bit scales. We systematically increase the bit scales to evaluate their impact on model quality and run-time performance while reducing memory usage. We apply first the fine-tuning with the same dataset, learning rate and number of epochs for aware-training quantization, and pruning. We fine-tune the DeepLiDARFlow [14] pre-trained model with a batch size of 1, following the same procedures as in [14]. All subsequent fine-tuning experiments also use a batch size of 1. We gradually decrease the learning rate and increase the number of epochs during fine-tuning until the model converges on the training set. We use a validation set to monitor performance during this process and stop fine-tuning when the model starts to over fit.

C. Quantitative Results

Our research aims to accelerate the available CNN solution using pruning and quantization, and maintaining accuracy at the same time. This section evaluates the efficiency of our proposed strategy in implementing the quantization and pruning on the available CNN, DeepLiDARFlow [14], for scene flow estimation. From this part, we can explore one of the best quantization levels as well as the pruning level, so we implement a hybrid quantization and pruning with the chosen. Then, study the effects of each level of pruning and quantization and compare them with the hybrid one and the original one in terms of the accuracy and run-time. We start the quantitative results with quantization.

Fig. 1 shows the impact of quantization and pruning on the DeepLiDARFlow [14] model and its pre-trained .

On the left, the effect of post and aware-training quantization is demonstrated for the model with 4096 input points, with 6 and 8-bit quantization excluded due to their significant decrease in accuracy. On the right, EPE results are presented for post-training pruning, aware-training pruning with the original learning rate, and aware-training pruning with a lower learning rate. The red line represents the original benchmark for comparison. The results indicate that while post-pruning with high percentages can decrease accuracy, aware-training with fine-tuning in pruning can maintain accuracy, particularly at lower learning rates.

Fig. 2 demonstrate the impact of post and aware-training quantization on EPE accuracy metric for DeepLiDARFlow [14]. We explore the effect of different numbers of input samples and weight bit representations on the architecture's behavior towards weight quantization.

The left on in Fig. 2 illustrates the accuracy redundancy when using post-quantization, indicating that the architecture's behavior is not constant for different samples. For example, there are minimal effects on accuracy when testing post-quantization with 8192 input samples, and the accuracy improves with post-quantization for a number of samples such as 2048 and 8192. However, quantization with different bits, such as 16 and 12 bits, yields better accuracy for a specific number of input samples (e.g., 4096), and the accuracy deteriorates for other bits of post-quantization with the same number of samples. In contrast, the right one in Fig. 2 demonstrates the effect of aware-training quantization on EPE accuracy. Different numbers of samples are explored, and the results show a significant improvement in the CNN's behavior towards aware-quantization, which is further improved with fine-tuning. The smoothing of accuracy improvements is clearly shown, and the best results are obtained with aware-training quantization using 16 bits, leading to a significant improvement in EPE.

To verify our results, we conducted further evaluation on the pre-trained model using post-training pruning on different LiDAR data sample sizes. As seen in Fig. 3, left one, the behavior of the network remained unchanged, unlike post-quantization. The red line in both figures represents the original benchmark.

After fine-tuning with a lower learning rate, accuracy was improved, as shown in the right one in Fig. 3. However, some results, depicted below the red line, showed an accuracy improvement of 5% and 10% pruning. Meanwhile, other percentages did not affect accuracy significantly. Unlike post-training, aware-training achieved minimal accuracy drops in some cases while achieving accuracy improvements in others, making the reduction in accuracy negligible.

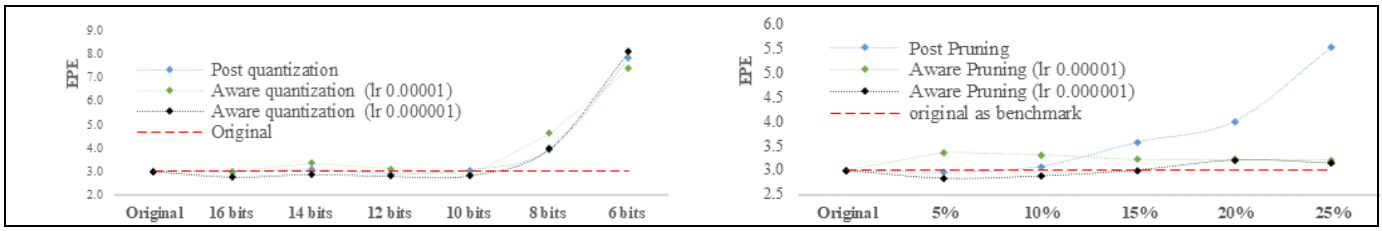


Figure 1. End Point Error metric using quantization and pruning on post and aware-training.

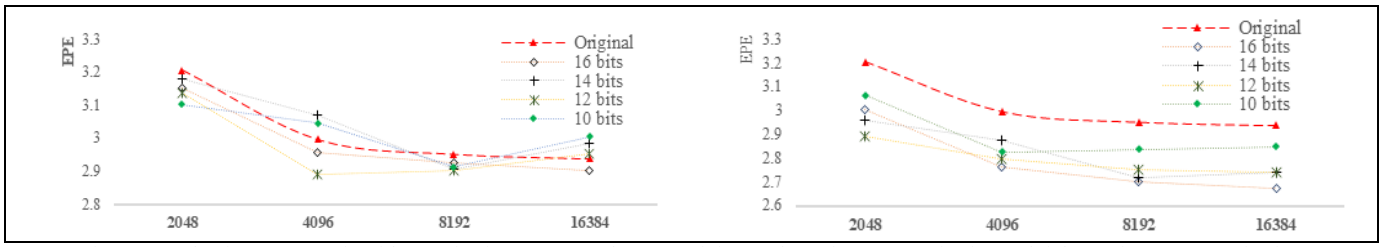


Figure 2. End Point Error metric with quantization on post and aware training. Left one is post quantization and right one is aware quantization

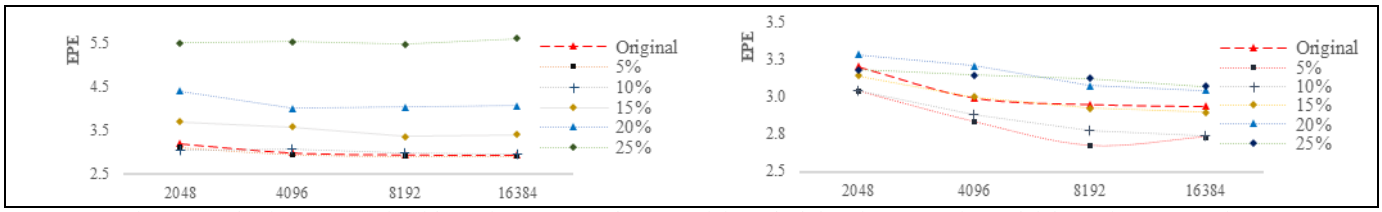


Figure 3. End Point Error metric with pruning on post and aware-training. The left on is post pruning and right one is aware pruning.

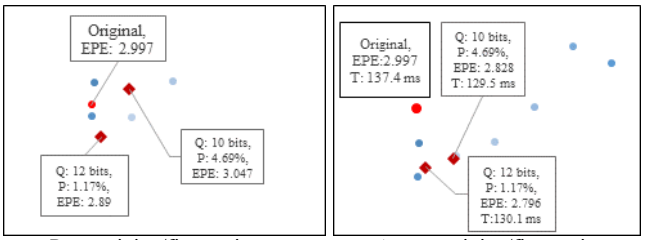
TABLE II. THE METRICS RESULTS QUANTIZATION AND PRUNING ON DEEPLIDAFLOW WITH INPUTS SAMPLES 4096.

kind	Pruning	Quantization	EPE ↓	KEO ↓	kind	Pruning	Quantization	EPE ↓	KEO ↓	Time (ms)
Original			2.997	11.565	**Original**			2.997	11.565	137.4
Post	5%	32 bits	2.955	11.282	Aware	5%	32 bits	2.835	10.568	133.8
Post	10%	32 bits	3.074	12.603	Aware	10%	32 bits	2.884	11.132	133.4
Post	15%	32 bits	3.577	18.197	Aware	15%	32 bits	3.002	11.537	132.7
Post	20%	32 bits	4.004	22.752	Aware	20%	32 bits	3.209	12.925	129.4
Post	25%	32 bits	5.536	35.992	Aware	25%	32 bits	3.147	12.771	128.7
Post	0.07%	16 bits	2.959	11.722	Aware	0.07%	16 bits	2.766	10.418	131.7
Post	0.29%	14 bits	3.071	12.268	Aware	0.29%	14 bits	2.877	10.556	130.7
Post	1.17%	12 bits	2.89	11.071	Aware	1.17%	12 bits	2.796	10.485	130.1
Post	4.69%	10 bits	3.047	11.375	Aware	4.69%	10 bits	2.828	10.035	129.5
Post	15.48%	8 bits	3.927	18.422	Aware	15.48%	8 bits	3.968	18.013	128.9
Post	33.77%	6 bits	7.836	60.558	Aware	33.78%	6 bits	8.106	56.934	128.5

Table II provides quantitative metrics and their corresponding effects on accuracy for both pruning and quantization, as well as the run-time for each method. The table also includes the percentage of each pruning level used in the hybrid strategy, which applies both pruning and quantization to the pre-trained model. All results are compared to the original model. The yellow labels indicate the best results in terms of accuracy. The findings presented in this table can be useful for future work on optimization, targeting FLOPS and weight parameters, as these factors play a significant role in run-time.

Figures in Fig 4 shown depict the best strategies for compressing the pre-trained model of DeepLiDARFlow [14] using post-training pruning and quantization and aware-training pruning and quantization.

The green circle in both figures represents the original pre-trained model with float-32 bit precision. The right figure demonstrates that 12 bits of weight quantization with 1.17% weight pruning provides the best results in terms of accuracy and run-time for post-training compression, while 10 bits with approximately 5% weight pruning needs optimization. The left figure illustrates the best quantization bits and pruning percentages for aware training with a learning rate of 0.00001. 12 bits with 1.17% weight pruning provides the best accuracy results, which are better than the original accuracy, and 10 bits with approximately 5% weight pruning have been significantly optimized compared to post-pruning and post-quantization. They provide better results than the original, and the run-time for both is reduced by approximately 7 ms.



Post-training/fine-tuning Quantization and Pruning | Aware-training/fine-tuning Quantization and Pruning

Abbreviations: Q = Quantization, P = Pruning, EPE = End Point Error, T = Time, ms = 10^{-3} seconds

Figure 4. Combining pruning and quantization versus original

D. *Qualitative Results*

We visually compare the results of DeepLiDARFlow [14] as shown in Fig 5. In this section, the qualitative results are demonstrated after testing each step of pruning and quantization in post and aware-training/fine-tuning. Then, we compare the results with the original. Since there are many figures can be visualized, we have enough the visualization for the best, which are related to Table II in the quantitative results. In addition, we visualize part of pruning with high and low percentage to show the pruning effects on accuracy.

The visualization shows that our pruning and quantization does not drop the accuracy of the model. In contrast, our work improves the accuracy and the visualization more robust after some percentages of pruning and quantization levels. The visualizations proof the quantitative results. All these visualizations are with number of samples 4096 as input. The first two images are the inputs to the architecture. The first column is the optical flow visualization, the second is the disparity 1 (D0) and the second is disparity 2 (D1), which are components of the average KITTI outlier rate for scene flow [16].

E. *Ablation Study*

Before making the final decisions about the approach and kind of quantization and pruning strategy, we conducted several steps to verify the proposed strategy for pruning and quantization. For fair comparison, first, we evaluate the original DeepLiDARFlow [14] model with different numbers of samples, starting with e.g. 2048 as input, and increasing with factor *2x* to 16384. To investigate our pruning and quantization strategy, we apply the post-quantization with little bits than the original, where the original is 32 float, so we apply post-quantization with only 16 float bits. Second, we apply the quantization with 6 bits as big decreasing bits for the memory allocation to investigate their effects on accuracy and run-time at the same time. Hence, all these experiments are conducted using the same dataset test used for evaluating the original DeepLiDARFlow [14] with the same pre-processing files. Third, we apply the post-pruning on its pre-trained model. To evaluate the effectiveness of our pruning strategy, we start with a low percentage of weight pruning (5%) and gradually increase it by 5% increments. However, in our ablation results we present the pruning with the lowest percentage and the highest percentage, 5% and 25% respectively.

We prune both the weights and biases of CNN, as both can contribute to the complexity of the model and the overall performance. However, after comparing the results of our ablation study with those of the original model, we further investigate our strategy by applying aware-training quantization and aware-pruning quantization. For fair comparison, we first follow the DeepLiDARFlow [14] fine-tuning procedures as well as the learning rate and number of epochs on the same dataset. Then, we decrease the learning rate by a factor 10x for each 100 epochs to optimize the accuracy, which lost from the pruning and quantization. Table III shows the comparisons between our ablations with the original one in just 4096 samples.

Table III demonstrates that both quantization and pruning have an impact on the accuracy of the model. However, our strategy for pruning and quantization, aimed at optimizing and speeding up CNN architectures, has a significant effect on the performance of these models, both in terms of accuracy and run-time, as shown in Fig. 4. To investigate these effects, we applied our strategy of pruning and quantization at different levels and with different techniques, such as post-training and aware training with different learning rates and numbers of epochs. In aware training, we increased the number of epochs and decreased the learning rate to reduce the accuracy losses resulting from pruning and quantization.

We also evaluated the effects of each technique on the run-time of the model, as shown in Fig. 6 for the DeepLiDARFlow [14] architecture. The run-time was measured on 4096 sample points.

TABLE III. ABLATION STUDY FOR OUR STRATEGY

No	Quantization	Pruning	Post	Aware	lr1*	lr2*	bits Float 16	bits Float 6	5%	25%	EPE↓	KEO↓
Original on number of samples 4096											**2.99**	**11.57**
1	√	X	√	X	X	X	√	X	X	X	**2.96**	**11.72**
2	√	X	√	X	X	X	X	√	X	X	**7.836**	**60.56**
3	√	X	X	√	√	X	√	X	X	X	**2.99**	**12.31**
4	√	X	X	√	√	X	X	√	X	X	**7.41**	**51.62**
5	√	X	X	√	X	√	√	X	X	X	**2.77**	**10.42**
6	√	X	X	√	X	√	X	√	X	X	**8.11**	**58.95**
7	X	√	√	X	X	X	X	X	√	X	**2.96**	**11.28**
8	X	√	√	X	X	X	X	X	X	√	**5.54**	**35.99**
9	X	√	X	√	√	X	X	X	√	X	**3.37**	**12.93**
10	X	√	X	√	√	X	X	X	X	√	**3.22**	**14.34**
11	X	√	X	√	X	√	X	X	√	X	**2.84**	**10.57**
12	X	√	X	√	X	√	X	X	X	√	**3.15**	**12.78**

** lr1 = 0.0001, lr2 = 0.00001*
All the metrics are conducted using number of sample = 4096

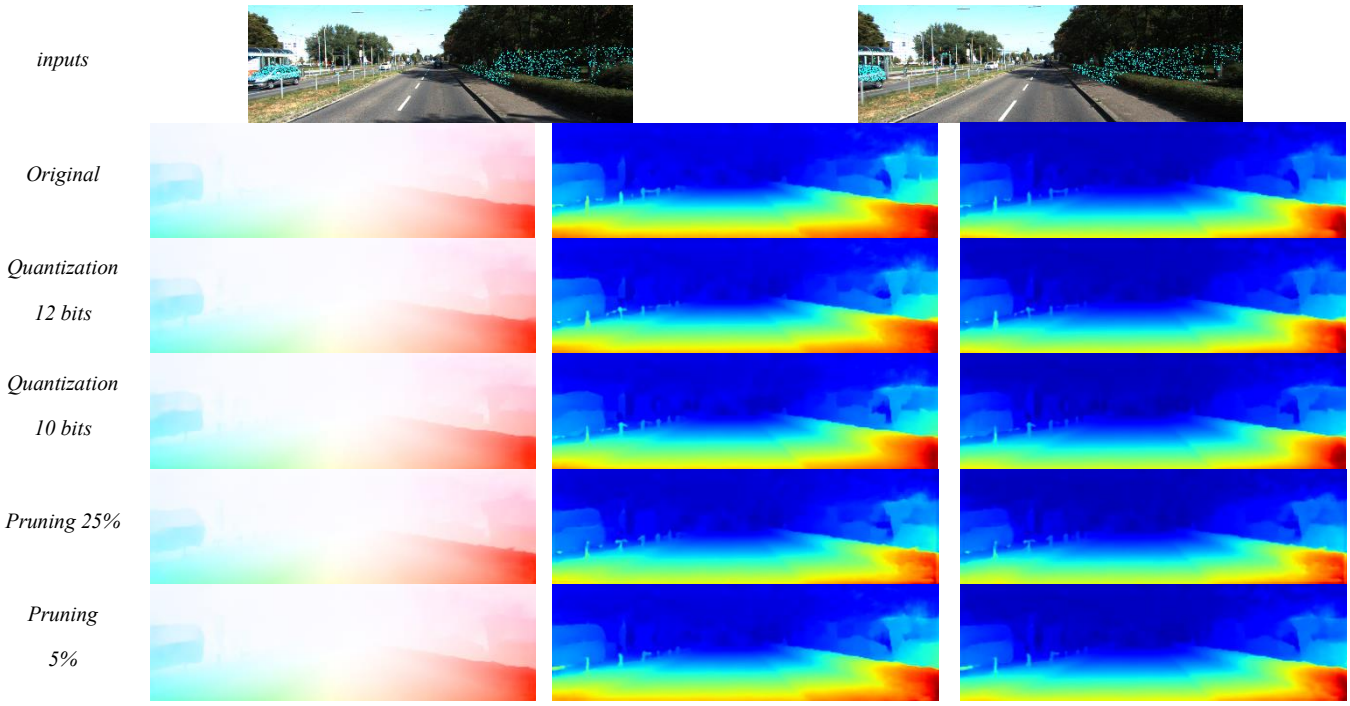


Figure 5. Visualizations of DeepLiDARFlow [14] after pruning and quantization versus the original.

We investigate the effect of each technique on the model in term of run time. Fig 6. shows the effect of both pruning and quantization on the run time. The pruning decreases the run-time/inference on DeepLiDARFlow [14] periodically by increasing the percentage of weights pruning. Furthermore, the quantization has the same effects on the run-time. The inference time for each case is measured using GPU-GeForce GTX 1660Ti 6 Giga byte.

V. Future Work

Future work in this area could involve exploring different types of pruning and quantization techniques to further optimize and speed up CNN architectures.

In future work, we could explore the application of explainable AI techniques to your computer vision models to improve its interpretability and trustworthiness.

Meta-Learning can be particularly useful in scenarios where the available data is limited or the model needs to adapt quickly to new environments. In future work, we could explore the application of meta-learning techniques to improve the generalization capabilities and adaptability to new scenarios.

Deep rewiring training is a technique allows changing the architecture of the model by decreasing the number of parameters and FLOPs.

Domain adaptation is a technique that aims to transfer knowledge learned from a source domain to a target domain, where the data distribution may be different. In future work, you could investigate the application of domain adaptation techniques to your DeepLiDARFlow model to improve its performance in different domains.

VI. Conclusion

In conclusion, we have presented an investigation into the effectiveness of pruning and quantization for scene flow estimation using a CNN on the KITTI [16] benchmarks. Our results indicate that both techniques can effectively reduce the complexity of the CNN and improve its efficiency, while maintaining acceptable levels of accuracy. Specifically, our experiments showed that pruning up to 15% of the weights and biases can reduce the memory allocation and inference time without sacrificing accuracy, and that post-quantization with 12-bit and 10-bit can also provide similar benefits. Our findings provide insights into the optimal combination of these techniques for this dataset and CNN architecture, and we recommend the use of these techniques for accelerating CNN architectures with approx. 5% for pruning and 10 bits for quantization in aware-training.

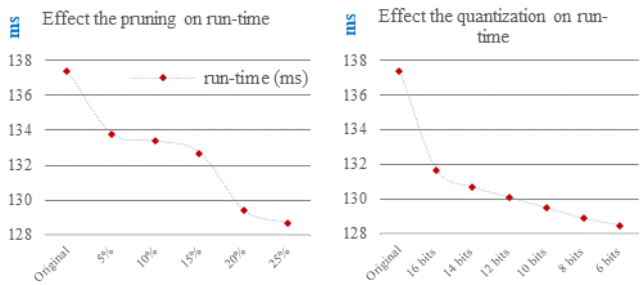


Figure 6. Effect the different levels of pruning and quantization on run-time.

ACKNOWLEDGMENT

This work was partially funded by the Federal Ministry of Education and Research Germany in the funding program Photonics Research Germany under the project FUMOS (13N16302) and partially under the project DECODE (01IW21001).

REFERENCES

[1] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," in *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, 2017, pp. 1–14.

[2] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning Structured Sparsity in Deep Neural Networks," *Adv. neural Inf. Process. Syst.*, pp. 2047–2082, 2016.

[3] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," *4th Int. Conf. Learn. Represent. ICLR 2016 - Conf. Track Proc.*, pp. 1–14, 2016.

[4] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," in *In European conference on computer vision*, 2016, pp. 525–542.

[5] D. Soudry, I. Hubara, and R. Meir, "Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights," *Adv. Neural Inf. Process. Syst.*, vol. 2, no. January, pp. 963–971, 2014.

[6] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized Convolutional Neural Networks for Mobile Devices," in *In Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4820–4828.

[7] 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," vol. 1, no. 1, pp. 1–13, 2016, [Online]. Available: http://arxiv.org/abs/1606.06160

[8] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning Both Weights and Connections for Efficient Neural Networks," *Adv. Neural Inf. Process. Syst.*, vol. 2015-Janua.

[9] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing Deep Convolutional Networks using Vector Quantization," *arXiv Prepr. arXiv1412.6115*, pp. 1–10, 2014.

[10] X. Zhang, J. Zou, K. He, and J. Sun, "Accelerating Very Deep Convolutional Networks for Classification and Detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 10, pp. 1943–1955, 2016.

[11] L. N. Huynh, Y. Lee, and R. K. Balan, "DeepMon: Mobile GPU-based deep learning framework for continuous vision applications," in *MobiSys 2017 - Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, 2017, pp. 82–95.

[12] Y. Guo, A. Yao, and Y. Chen, "Dynamic Network Surgery for Efficient DNNs," *Adv. Neural Inf. Process. Syst.*, no. NIPS, pp. 1387–1395, 2016.

[13] H. Li, H. Samet, A. Kadav, I. Durdanovic, and H. P. Graf, "Pruning filters for efficient convnets," in *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, 2017, no. 2016, pp. 1–13.

[14] R. Rishav, R. Battrawy, R. Schuster, O. Wasenmuller, and D. Stricker, "DeepLiDARFlow: A Deep Learning Architecture for Scene Flow Estimation Using Monocular Camera and Sparse Lidar," in *IEEE /RSJ International Conference on Intelligent Robots and Systems(IROS).*, 2020.

[15] N. Mayer *et al.*, "A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4040–4048.

[16] M. Menze and A. Geiger, "Object Scene Flow for Autonomous Vehicles," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 07-12-June, pp. 3061–3070, 2015.

[17] Y. He, X. Zhang, and J. Sun, "Channel Pruning for Accelerating Very Deep Neural Networks," *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2017-Octob, pp. 1398–1406, 2017.

[18] T. Zhang *et al.*, "A systematic DNN weight pruning framework using alternating direction method of multipliers," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 11212 LNCS, pp. 191–207, 2018.

[19] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning Efficient CNN through Network Slimming," in *In Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2736–2744.

[20] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A Filter Level Pruning Method for Deep Neural Network Compression," in *In Proceedings of the IEEE international conference on computer vision*, 2017, pp. 5058–5066.

[21] J. Ranstam and J. Cook, "LASSO regression," *J. Br. Surgery,* vol. 105, no. 10, pp. 1348–1348, 2018.

[22] J. M. M. Anderson, B. A. Mair, M. Rao, and C. H. Wu, "Weighted least-squares reconstruction methods for positron emission tomography," *IEEE Trans. Med. Imaging*, vol. 16, no. 2, pp. 159–165, 1997.

[23] G. Xie, J. Wang, T. Zhang, J. Lai, R. Hong, and G. J. Qi, "Interleaved Structured Sparse Convolutional Neural Networks," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8847–8856. doi: 10.1109/CVPR.2018.00922.

[24] K. Sun, M. Li, D. Liu, and J. Wang, "IGCv3: Interleaved Low-Rank Group Convolutions for Efficient Deep Neural Networks," *Br. Mach. Vis. Conf. 2018, BMVC 2018*, pp. 1–13, 2019.

[25] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Penksy, "Sparse Convolutional Neural Networks," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 07-12-June, pp. 806–814, 2015.

[26] Y. He, J. Lin, Z. Liu, H. Wang, L. Li, and S. Han, "AMC: AutoML for Model Compression and Acceleration on Mobile Devices. (arXiv:1802.03494v3 [cs.CV] UPDATED)," *Eccv*, 2018, [Online]. Available: http://arxiv.org/abs/1802.03494

[27] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," *7th Int. Conf. Learn. Represent. ICLR 2019*, pp. 1–13, 2019.

[28] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *7th Int. Conf. Learn. Represent. ICLR 2019*, pp. 1–42, 2019.

[29] D. A. Gudovskiy and L. Rigazio, "ShiftCNN: Generalized Low-Precision Architecture for Inference of Convolutional Neural Networks," *arXiv Prepr.*

[30] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training Deep Neural Networks with Binary Weights During Propagations," *Adv. Neural Inf. Process. Syst.*, no. Section 5, p. 10, 2015.

[31] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding Up Convolutional Neural Networks with Low Rank Expansions," 2014.

[32] V. Lebedev and V. Lempitsky, "Fast ConvNets Using Group-Wise Brain Damage," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 2554–2564, 2016.

[33] K. Azarian, Y. Bhalgat, J. Lee, and T. Blankevoort, "Learned Threshold Pruning," no. 2017, pp. 1–12, 2020.