Jestr

Journal of Engineering Science and Technology Review 17 (3) (2024) 88 - 94

Research Article

JOURNAL OF
Engineering Science and Technology Review

www.jestr.org

Optimized Deep Learning Classification Model for Intelligent Edge devices

Soumyalatha Naveen[1] and Manjunath R Kounte[2,*]

[1]*Department of Computer Science and Engineering, Manipal Institute of Technology Bengaluru, Manipal Academy of Higher Education, Manipal, Karnataka, India*
[2]*School of Electronics and Communication Engineering, REVA University, Bangalore, Karnataka, 560064, India*

Received 24 April 2023; Accepted 30 June 2023

Abstract

Deep learning models enable state-of-the-art accuracy in computer vision applications. However, the deeper, computationally expensive, and densely connected architecture of deep neural networks (DNN) have limitations for deploying the model on resource-constraint embedded IoT devices. We propose an efficient neural network compression framework that performs filter pruning, fine-tuning and 8-bit quantization to reduce computational complexity, inference time, and memory footprint. Furthermore, reducing the bit widths of activation and weights helps design a compact deployment model on resource-limited IoT devices such as smartphones. The proposed system is evaluated extensively on the CIFAR-10 dataset for Resnet34 and VGG16 models. In addition, we examine the efficacy of a larger model. The result shows that pruning followed by quantization compresses the neural network and compared to the baseline model, achieved an accuracy of 78.01% for Resnet34 and 82.34% for Vgg16 after pruning and quantization which is <1% of marginal loss in accuracy compared to the baseline model. Further, 80x unique parameters from the weight matrix of the model are reduced using k-means clustering along with 8-bit quantization. The study demonstrates that the pruning process had a minimal impact on ResNet34's accuracy, while VGG16 maintained its accuracy even after pruning. Both models showed a reduced memory footprint after applying k-means clustering and 8-bit quantization, making them more efficient for inference tasks without sacrificing performance significantly. Applications like Smart Traffic Management and autonomous vehicles involve deploying edge devices with cameras and sensors at intersections and roadsides to monitor and analyze real-time traffic conditions. The proposed optimized model can be employed for efficient object recognition and classification of vehicles, pedestrians, and traffic signs.

Keywords: Deep Learning, Edge clusters, heterogenous, Pruning, Quantization

1. Introduction

Convolutional neural networks (CNNs) are the neural network suits for image and video recognition tasks. CNNs consist of multiple layers of convolutional and pooling operations, which are designed to extract features from the input data. Many computer vision applications have shown considerable success with convolutional neural networks.

CNNs are well-suited for edge computing due to their efficiency in feature extraction, real-time inference capabilities, reduced data transmission requirements, and potential for model optimization. Edge devices process data locally, reducing the need for data transmission to remote servers [1-3], leading to low latency and enabling real-time inference, which is critical in applications where quick responses are essential, such as autonomous vehicles, robotics, and real-time monitoring systems. Transmitting large amounts of data to centralized servers can strain network bandwidth. Bandwidth usage is optimized by processing data locally on edge devices and only sending necessary results [4-5]. In addition, edge computing helps to maintain data privacy [6] and enhances data security [7], as data is less susceptible to interception during transmission [8]. Hence deploying deep learning models on edge devices offers numerous advantages, including improved performance, reduced latency, enhanced privacy, better resource efficiency, and the ability to operate offline. This approach expands the scope of intelligent applications in various domains, enabling more responsive, secure, and efficient systems for the benefit of end-users and society.

Despite these advantages, deploying deep learning models on edge devices presents several challenges [9-10]. Edge devices often have limited processing power, memory, and energy resources. Deep learning models are computationally intensive and optimizing them for efficient execution on resource-constrained devices is a significant challenge [11]. Deep learning models are large, making them difficult to fit within the limited memory of edge devices. Reducing the memory footprint without sacrificing accuracy is crucial for successful deployment. Deep learning computations is a power-hungry, and edge devices, especially battery-operated ones, must manage power consumption efficiently to extend battery life. Deep learning models require optimization techniques like model pruning, quantization, and compression to run efficiently on edge devices. Addressing these challenges requires a combination of algorithmic improvements, hardware advancements, and careful system design. As the technology evolves, deploying deep learning models on edge devices will play a vital role in enabling intelligent and efficient applications in various domains. However, enormous computational cost of CNNs makes it very slow to run the model on resource-constrained devices such as mobile phones. It's essential to reduce the computational cost and accelerate the inference of CNN.

Neural network compression and acceleration [12-13] in CNN refers to techniques that aim to reduce the

*E-mail address: soumyanaveen.u@gmail.com

doi:10.25103/jestr.173.11

computational complexity and memory footprint of CNN models. This is typically done by reducing the number of parameters in the model, or by using more efficient algorithms for training and inference.

One common method for model compression is pruning [14], which involves removing redundant or unnecessary weights from the model. This can be done by identifying and removing weights that have little impact on the model's performance, or by using techniques such as structured pruning, where groups of weights are pruned together. Neural network pruning is a technique used to reduce the size and complexity of a neural network by removing unnecessary weights and neurons. This can be useful for reducing the memory and computational requirements of a model and can also improve its generalization performance. Pruning can be applied to CNNs by removing unimportant weights and neurons from the network, which can improve the model's performance and reduce its computational requirements.

Weight pruning [15-16] in CNN refers to the process of removing certain weights from the network that are deemed unnecessary for the task at hand. This can be done by identifying and removing weights that have the smallest magnitude or absolute value. The goal of weight pruning is to reduce the number of parameters in the network, making it more efficient and faster to train. Filter pruning [17] in CNN, on the other hand, refers to the process of removing certain filters from the network that are not contributing much to the final output. This can be done by identifying and removing filters that have the smallest magnitude or absolute value. The goal of filter pruning is to reduce the number of filters in the network, making it more computationally efficient and easier to interpret. Both weight pruning and filter pruning can be used together to achieve a more compact and efficient CNN model. It is important to note that the specific pruning techniques and the criteria for selecting which weights or filters to remove may vary depending on the task and the architecture of the network. Another approach is to use low-precision data types for the model's weights and activations. This reduces the storage requirement of the model and can also speed up computation. Quantization is a popular method that reduces the precision of the weights and activations to 8-bit or less.

Knowledge distillation [18] is a technique to train to mimic the predictions of a larger, more complex model. This allows the smaller model to achieve similar performance with fewer parameters. Other methods include weight sharing, where multiple copies of a weight are used across the model, and model compression frameworks such as TensorFlow Lite and OpenVINO, which provide tools for optimizing and deploying CNN models on a variety of hardware platforms. The aforementioned techniques can overcome the challenges [19] of deploying deep learning on edge devices.

Our contributions to this paper are as follows. First, we developed a baseline model of Resnet34 and Vgg16 to understand the efficacy of the proposed system. We apply filter pruning for CNN compression to prune the redundant filters; the model achieves comparable performance with much fewer parameters. Further, we used 8-bit quantization to reduce the model complexity still. Finally, we evaluated the model on the CIFAR10 dataset with Resnet34 and Vgg16 models to demonstrate the high effectiveness of our proposed approach.

2. Background and Related Work

Edge computing enables data processing and analysis closer to the data source, at the network's edge. AI and ML algorithms analyze and make decisions based on data collected at the edge. This algorithm is integrated into edge devices to enable real-time decision-making and automation. Edge intelligence is the ability of IoT devices to process and analyze data locally instead of sending everything to a central location for processing. As shown in Table 1, edge intelligence can be significantly improved by utilizing technologies like RFID, Edge computing, Cloud computing, and IoT, which make it possible to gather, store, and analyze data from numerous sources in real time. This permits quick decisions and responses in various fields, including manufacturing, healthcare, and retail. Organizations can use these technologies to enhance overall operations, lower expenses, and increase efficiency.

Table 1. Technology advancement on the path of edge intelligence

Technology advancements	Year of introduction	Applications
RFID	1983	Vehicle parking, library management
Edge	1990	Smart meter
Cloud	1996	Email
IoT	1999	Smart building
NFC	2003	Home automation
Mist	2011	Healthcare
Fog	2012	Environmental control systems
Edge Intelligence	Interest rate increased after 2016	Autonomous vehicle

The 5G networks will allow faster data transfer and low-latency communication between edge devices and the cloud, making it possible to process the data. IoT devices, such as sensors and cameras, are becoming increasingly prevalent at the edge. IoT devices collect data, which is later analyzed and used to make decisions. Containerization and virtualization technologies allow for the deployment of software and applications at the edge, making it possible to run complex algorithms and processes at the edge. Edge gateways allow for managing, monitoring, and controlling edge devices and the data they collect. They also enable data transmission to the cloud for further analysis. Low-power chips and processors enable edge devices to operate with low power consumption, making it possible to deploy edge devices in remote or hard-to-reach areas.

Several techniques for model compression in deep neural networks include pruning, quantization, and low-rank factorization. Pruning involves removing neurons or connections in a neural network deemed less critical for the task. The pruning process analyzes the network weights and removes those with a minor magnitude. Quantization reduces the precision of the weights of a neural network, typically from 32-bit floating point to 8-bit or 16-bit integers. Hence leads to a significant reduction in memory usage and computation, but at the cost of some loss in accuracy. Low-rank factorization techniques such as Singular Value Decomposition (SVD) and Tucker Decomposition can also compress neural networks by approximating a weight matrix with a low-rank matrix, reducing the number of parameters in the model. Another popular method is knowledge distillation, which uses a large, pre-trained model (the teacher) to guide the training of a smaller model (the student). The student model is trained to mimic the output of the teacher model on

a given dataset, which can lead to a smaller model with similar or even better performance.

The author [20] discusses the various network compression techniques and presents the strengths of combining pruning and quantization. The author [21] proposed a compression algorithm performing weight pruning and quantization jointly and then completed the fine-tuning. The author [22] compressed the model using the Bonsai algorithm and achieved significant improvement in results in terms of accuracy and model size. The article [23] details storage reduction, energy efficiency and high inference time algorithms using pruning, quantization, and Huffman coding. This article [24] focuses on improvement in efficiency and prediction speed and reduces the model size without overfitting. This article [25] briefed on designing an optimized model using a fused tile partitioning approach, weight pruning proved the improvement in inference latency. The author [26] proposed a network compression strategy using Bayesian optimization for pruning ratio selection, reduced the model size, and achieved better accuracy. The author [27] focuses on bit-width reduction of activation and weights for model compression. Further author explored the decline of bits to 3-bit and 2-bit and demonstrated the model's effectiveness. To reduce the deep learning model the author used quantization with knowledge distillation [28] and designed a robust model. The paper [29] compressed the model through reinforcement learning to apply pruning, mapping weights and quantize bit width and achieved significantly good compression rate. The author proposed pruning and quantization [30] on ResNet-18 and VGG-16 on CIFAR-100 focusing on energy optimization. The article proposed joint exploration [31] on pruning and quantization for acceleration of DNN models. The author proposed One-shot Pruning-Quantization for pretrained model [32] measured compression rate on ImageNet with few deep learning models.

3. Proposed System

Model compression is essential for deploying models on devices with limited resources, such as smartphones or IoT devices, and can also improve the speed and efficiency of training and inference.

The figure 1 depicts the proposed model flowchart to design a compact optimized DNN model. The key steps involved in the proposed system are Filter pruning, fine tuning, k-means clustering and 8-bit quantization.

3.1 Filter Pruning: Filter pruning is a technique used to reduce the number of parameters in a CNN to make it more efficient. It removes filters in a convolutional layer with a low magnitude of weights. The threshold is a value used to determine which filters to prune. Remove the filters or set them to zero for the weights below the threshold. Hence it reduces the model's complexity without compromising the model's accuracy. Stochastic gradient descent (SGD) is a widely used optimization algorithm for training neural networks. It helps to update the model's parameters by computing the gradient of the loss function for the parameters. SGD is often used with other techniques, such as filter pruning and thresholding, to improve the model's efficiency further.

3.2 Fine Tuning: After filter pruning, the pruned model may experience a drop in accuracy due to the removal of some filters. Fine-tuning is performed to recover or improve the model's performance. Finetuning is adjusting the parameters of a pre-trained neural network to adapt it to a new task or dataset. Fine-tuning can be adding new layers to the network, adjusting the existing layers, or finetuning the parameters of the pre-trained network. Finetuning is often used to transfer the knowledge learned by a pre-trained network to a new task or dataset, allowing the network to perform better on the new task without needing to be trained from scratch. Filter pruning and finetuning improve the performance and efficiency of neural networks.

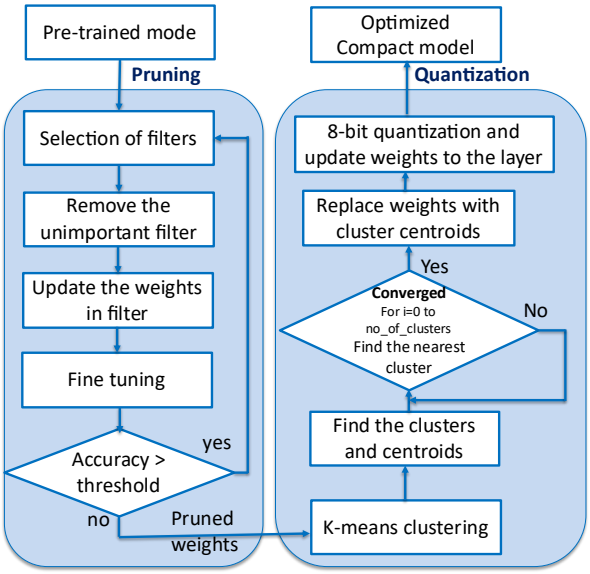

Fig. 1. Flowchart of proposed model

3.3 K-means Clustering: After fine-tuning, k-means clustering is applied to quantize the weights of the CNN. K-means clustering is a technique used in data mining and machine learning to group similar data points. An unsupervised learning method divides a dataset into k clusters, where k is a user-defined parameter. Additionally, based on reducing the distance between each data point and the cluster centroid, which is the average value of all the data points in the cluster. The algorithm updates the mean of each cluster after iteratively assigning each data point to the cluster with the nearest mean. The centroids are then revised using the updated cluster data points. Until the centroids stop changing or we have completed the maximum number of iterations, this process is repeated.

3.4 8-bit Quantization: Further, apply 8-bit quantization, reducing the precision of the weights and activations in a neural network from 32-bit floating point to 8-bit integers. Hence significantly reduces the memory and computational requirements of the network, allowing it to deploy on resource-constrained devices such as smartphones or embedded systems. 8-bit quantization can significantly reduce the model size and computational requirements, making it possible to deploy DNNs on devices with limited resources. However, it is essential to note that there may be a trade-off in accuracy, and the quantization process should be carefully optimized to minimize any loss in accuracy.

By combining filter pruning, fine-tuning, k-means clustering, and 8-bit quantization, the proposed system aims to create a more efficient and lightweight deep learning model suitable

for deployment on resource-constrained edge devices while maintaining acceptable accuracy.

4. Experiment and Results

The proposed model is implemented using python and torch, a deep-learning framework for model training and inference. PyTorch provides functionality to perform pruning and quantization on a trained model with an optim package for optimization algorithm, dataloader API for loading data and for iterations, KMeans clustering from sklearn and user-defined code for pruning and quantization. The Table 2 depicts the hyper-parameter consideration for evaluation.

Table 2. Hyper-parameter consideration for the baseline model

Parameter	Value
Learning Rate	0.1
Weight decay rate	0.0005
Epochs	56
Repeat with different seed	3
Quantizer	Stochastic Gradient Descent (SGD)
momentum	0.9
gamma	0.1

For evaluating our proposed model, we use CIFAR-10 dataset, a widely used dataset for image classification tasks in machine learning and computer vision. It consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 test images. The classes are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The dataset is commonly used to train and evaluate the performance of deep learning models.

We measure training accuracy, validation error accuracy, validation loss. Training accuracy in a neural network refers to the percentage of accurate predictions made by the network during the training phase on the input data set. It measures how well the network can learn and adapt to the patterns in the training data and can be used as a metric to evaluate the network's performance and the training process's effectiveness. Training Accuracy is measured with following formula.

$$\text{Training Accuracy} = \frac{(\text{Training data correctly classified})}{(\text{Total number of training data})} \text{X}100$$

Validation error accuracy in a neural network measures the ability to effectively predict or classify new, unseen data (validation data) based on the training data. It is calculated by comparing the network's predictions for the validation data with the actual labels or outputs for that data and determining the correct percentage of predictions. A high validation error accuracy indicates that the network performs well on the validation data. In contrast, a low validation error accuracy indicates that the network may be overfitting or underfitting the training data.

Validation loss in a neural network is a measure of network performance on a validation dataset. This dataset is separate from the training dataset and evaluates the network's generalisation ability to new data. The validation loss is calculated by comparing the network's predictions to the actual values in the validation dataset and calculating the difference, typically using a loss function such as mean squared error. The lower the validation loss, the better the network predicts unseen data.

Training loss measures how well a neural network can learn from the training data. It measures the difference between the predicted output of the network and the actual output. Training Loss is measured with following formula.

$$Training\ Loss = \frac{1}{Number\ of\ data\ in\ Training\ set(NT)} \sum_{i=1}^{NT} Loss(True\ label, predicted\ label)\ for\ ith\ data\ in\ training\ set$$

Validation loss in a neural network is a measure of how well the model performs on a validation dataset, a set of data used to evaluate the model's performance during the training process. We calculate the validation loss by comparing the model's predictions to the actual values in the validation dataset. Validation Loss is measured with following formula.

$$Validation\ Loss = \frac{1}{Number\ of\ data\ in\ Validation\ set(NV)} \sum_{i=1}^{NV} Loss(True\ label, predicted\ label)\ for\ ith\ data\ in\ validation\ set$$

Validation accuracy in a neural network refers to the model's accuracy when evaluating a validation dataset. Validation Accuracy is measured with following formula.

$$\text{Validation Accuracy} = \frac{(\text{Count of validation data correctly classified})}{(\text{Total number of data in validation set})} \text{X}100$$

A baseline model is developed with the following hyperparameters as shown in Table 2.

Figure 2 depicts the Resnet34 baseline model training accuracy, and validation error accuracy gradually increases as the number of epochs grows. Hence the graph indicates the effective prediction or classification of input data. Figure 3 depicts the VGG16 baseline model training accuracy, and validation error accuracy gradually increases as the number of epochs grows. Hence the graph indicates the effective classification of input data.

After the baseline model construction, we prune and retrain the model. In the process we identify the important connections to acquire the information from the data and remove the redundant weights along with known amount of compression rate. Removing the filter based on magnitude of the connections having weights and gradients lesser than the threshold value. Assumption considered here is weights with less magnitude are consider as less important and having larger magnitude is considered as more important. After

pruning, we perform retraining to recover the accuracy of the model.

As shown in Figure 4 as the model is trained, the training loss decreases, indicating that the model is improving. Likewise, the validation loss also reduces as the model is trained.

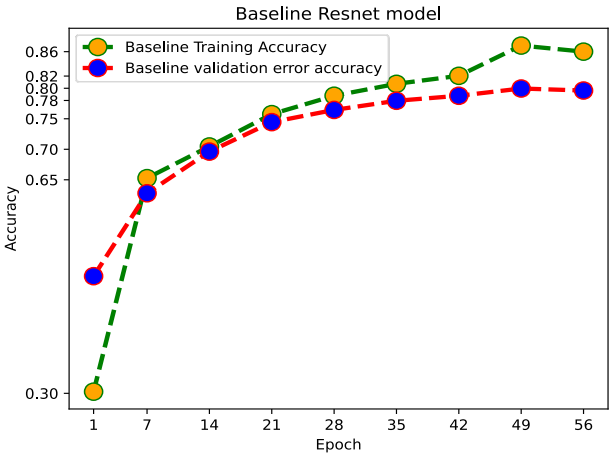

Fig. 2. Baseline Resnet34 model training accuracy and validation error accuracy

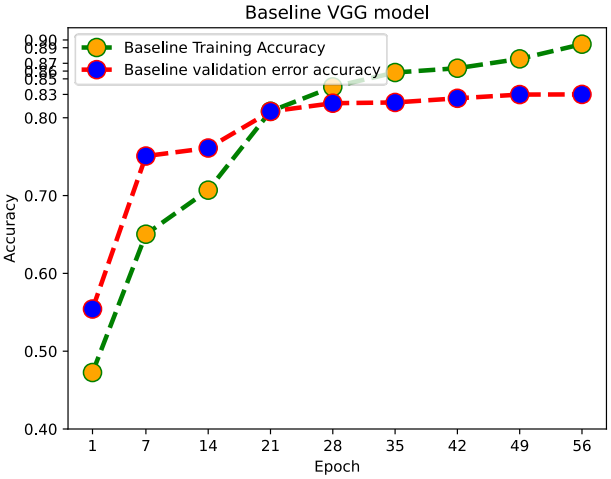

Fig. 3. Baseline VGG16 model training accuracy and validation error accuracy

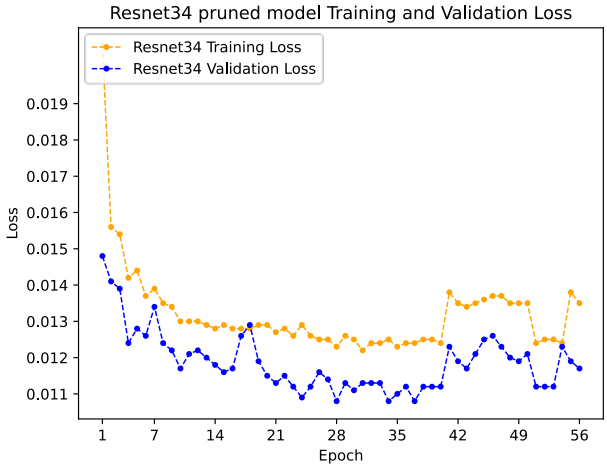

Fig. 4. Training loss and Validation Loss after pruning for Resnet34 model.

As shown in Figure 5 Validation accuracy increases as epochs grows. A higher validation accuracy indicates that the model performs well on the validation dataset.

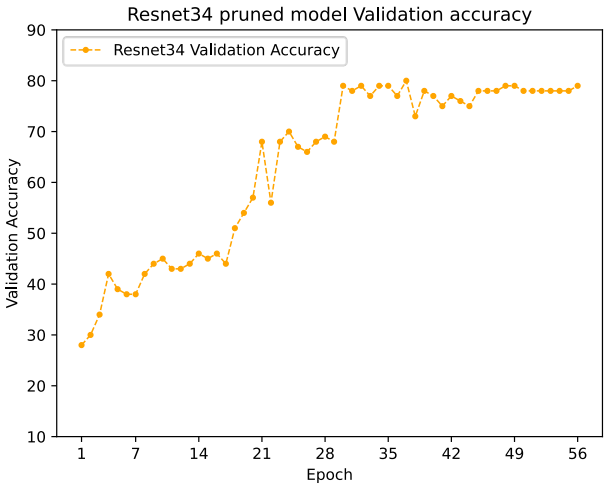

Fig. 5 Validation accuracy after pruning for Vgg16 model.

As shown in Figure 6 as the model is trained, the training loss decreases, indicating that the model is improving. Likewise, the validation loss also reduces as the model is trained. The reduced validation loss indicates model is not overfitting.

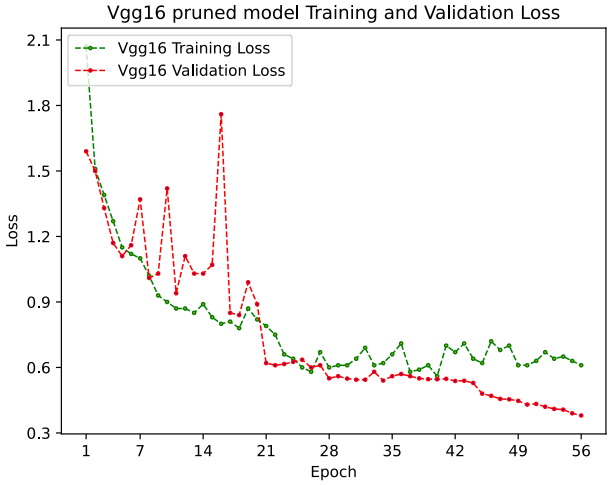

Fig. 6. Training loss and Validation Loss after pruning for Vgg16 model.

Figure 7 indicates model likely to perform well on unseen data. The lower validation accuracy demonstrates that the model may not perform as well and may need further tuning or adjustments.

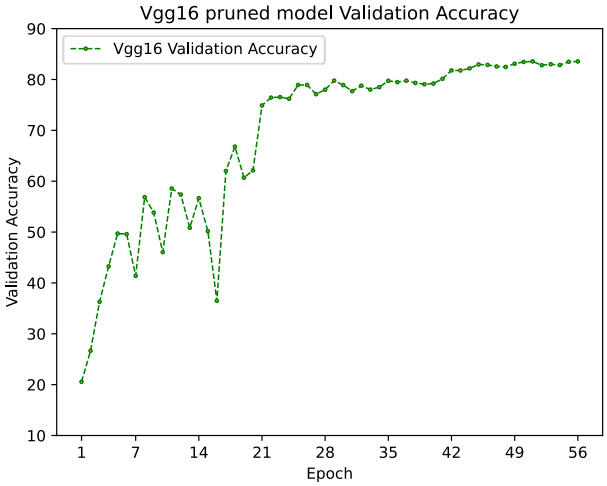

Fig. 7. Validation accuracy after pruning for Vgg16 model.

Our experimental results show that considering the 56th epoch, Resnet34 achieved an accuracy of 78.563% before pruning and 79% of accuracy after pruning. Hence from the overall result we observed for the Resnet34 model, compared to the baseline after pruning, there is a marginal reduction of accuracy (<1%) on the CIFAR 10 dataset. Resnet34 model archives 86% of training accuracy and 79% of validation error accuracy for the baseline model.

Vgg16 achieved an accuracy of 83.55% before pruning and 83.55% of accuracy after pruning for 56th epoch. Hence Vgg16 maintains the same accuracy even after pruning the model compared to the baseline model. The Vgg16 model achieves 89.45% of training accuracy and 83% of validation error accuracy for the baseline model.

Next, we perform k-means clustering and 8-bit quantization reduces the memory footprint and speeds up the inference of the model by maintaining 78.01% of accuracy for Resnet34 model and 82.34% of accuracy for Vgg16 model. Quantization reduces around 80x unique parameters.

Figure 8 indicates the reduction in the number of parameters in each layer of the Vgg16 model. More number parameter reduction leads to reduced complexity and faster inference in a compact optimized model.

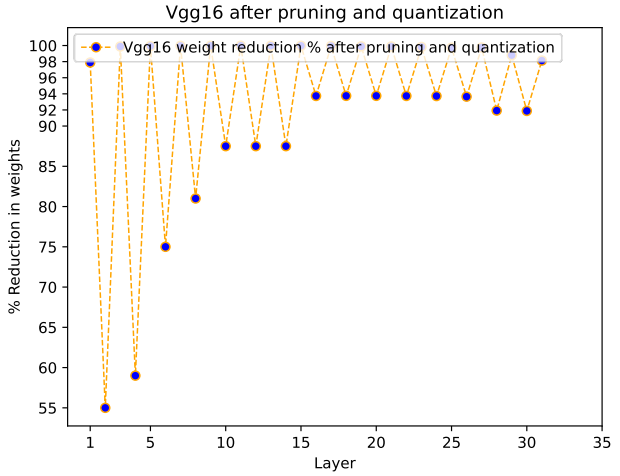

Fig. 8. Percentage of weight reduction for Vgg16 after pruning and quantization.

5. Conclusion

Model optimization is crucial when working with resource constraint devices, such as mobile devices or Internet of Things (IoT) devices, because these devices have limited computational resources and storage capabilities. Optimizing the model can reduce its size of the model and make it more efficient in terms of memory usage and computation time. We developed a compact model to optimize the time and space requirement of the deep learning model. Furthermore, we further explored the capability of filter pruning and 8-bit quantization to reduce the model complexity with marginal loss in accuracy.

To evaluate the proposed system, we considered the Resnet34 model and compared it with the baseline after pruning; there is a marginal reduction of accuracy (<1%) on the CIFAR 10 dataset. Resnet34 model archives 86% of training accuracy and 79% of validation error accuracy for the baseline model. Vgg16 maintains the same accuracy even after pruning the model compared to the baseline model. The Vgg16 model achieves 89.45% of training accuracy and 83% of validation error accuracy for the baseline model. After the quantization, the Resnet34 model maintains 78.01% accuracy and 82.34% accuracy for the Vgg16 model. Further quantization reduces around 80x the unique parameters of the model. Hence enables easy deployment of the CNN model on resource-constraint IoT devices. Our findings suggest that both the pruned models demonstrate promising results in maintaining accuracy and reducing model size through quantization. These results open up possibilities for deploying complex CNN models on IoT devices with limited resources, paving the way for efficient and practical real-world applications in computer vision. In future research, we want to explore the potential of transfer learning on various datasets and practical deployment of the pruned and quantized models on actual resource constrained IoT devices. Exploring model combination techniques, such as model stacking or knowledge distillation, might lead to even more compact and accurate models.

References

[1] R. Singh and S. S. Gill, "Edge AI: A survey," *Intern. Things Cyber.-Phys. Sys.*, vol. 3, pp. 71–92, Mar.2023, doi: 10.1016/j.iotcps.2023.02.004.

[2] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge AI: On-Demand Accelerating Deep Neural Network Inference via Edge Computing," *IEEE Trans. Wirel. Commun.*, vol. 19, no. 1, pp. 447–457, Jan. 2020, doi: 10.1109/TWC.2019.2946140.

[3] J. Chen and X. Ran, "Deep Learning With Edge Computing: A Review," *Proceed. IEEE*, vol. 107, no. 8, pp. 1655–1674, Aug. 2019, doi: 10.1109/JPROC.2019.2921977.

[4] W. Y. B. Lim *et al.*, "Federated Learning in Mobile Edge Networks: A Comprehensive Survey," *IEEE Communic. Surv. Tutor.*, vol. 22, no. 3, pp. 2031–2063, 2020, doi: 10.1109/COMST.2020.2986024.

[5] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "DeepDecision: A Mobile Deep Learning Framework for Edge Video Analytics," in *IEEE INFOCOM 2018 - IEEE Conf. Comp. Communic.*, IEEE, Apr. 2018, pp. 1421–1429. doi: 10.1109/INFOCOM.2018.8485905.

[6] A. Alrawais, A. Alhothaily, C. Hu, and X. Cheng, "Fog Computing for the Internet of Things: Security and Privacy Issues," *IEEE Internet Comput.*, vol. 21, no. 2, pp. 34–42, Mar. 2017, doi: 10.1109/MIC.2017.37.

[7] R. Wang, J. Lai, Z. Zhang, X. Li, P. Vijayakumar, and M. Karuppiah, "Privacy-Preserving Federated Learning for Internet of Medical Things Under Edge Computing," *IEEE J. Biomed. Health Inform.*, vol. 27, no. 2, pp. 854–865, Feb. 2023, doi: 10.1109/JBHI.2022.3157725.

[8] Y. I. Alzoubi, V. H. Osmanaj, A. Jaradat, and A. Al-Ahmad, "Fog computing security and privacy for the Internet of Thing applications: State-of-the-art," *Secur. Priv.*, vol. 4, no. 2, Mar. 2021, doi: 10.1002/spy2.145.

[9] H. Li, K. Ota, and M. Dong, "Learning IoT in Edge: Deep Learning for the Internet of Things with Edge Computing," *IEEE Netw.*, vol. 32, no. 1, pp. 96–101, Jan. 2018, doi: 10.1109/MNET.2018.1700202.

[10] M. G. S. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, "Machine Learning at the Network Edge: A Survey," *ACM Comput. Surv.*, vol. 54, no. 8, pp. 1–37, Nov. 2022, doi: 10.1145/3469029.

[11] Md. M. H. Shuvo, S. K. Islam, J. Cheng, and B. I. Morshed, "Efficient Acceleration of Deep Learning Inference on Resource-Constrained Edge Devices: A Review," *Proceed. IEEE*, vol. 111, no. 1, pp. 42–91, Jan. 2023, doi: 10.1109/JPROC.2022.3226481.

[12] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "Model Compression and Acceleration for Deep Neural Networks: The Principles,

Progress, and Challenges," *IEEE Sign. Process. Mag.*, vol. 35, no. 1, pp. 126–136, Jan. 2018, doi: 10.1109/MSP.2017.2765695.

[13] T. Choudhary, V. Mishra, A. Goswami, and J. Sarangapani, "A comprehensive survey on model compression and acceleration," *Artif. Intell. Rev*, vol. 53, no. 7, pp. 5113–5155, Oct. 2020, doi: 10.1007/s10462-020-09816-7.

[14] J. Liu *et al.*, "Discrimination-aware Network Pruning for Deep Model Compression," *IEEE Trans. Pattern Anal. Mach. Intell.*, pp. 1–1, 2021, doi: 10.1109/TPAMI.2021.3066410.

[15] M. A. Rumi, X. Ma, Y. Wang, and P. Jiang, "Accelerating Sparse CNN Inference on GPUs with Performance-Aware Weight Pruning," in *Proceed. ACM Int. Conf. Par. Archit. Compilat. Techniq.*, New York, NY, USA: ACM, Sep. 2020, pp. 267–278. doi: 10.1145/3410463.3414648.

[16] S. Naveen, M. R. Kounte, and M. R. Ahmed, "Low Latency Deep Learning Inference Model for Distributed Intelligent IoT Edge Clusters," *IEEE Access*, vol. 9, pp. 160607–160621, 2021, doi: 10.1109/ACCESS.2021.3131396.

[17] J.-H. Luo, H. Zhang, H.-Y. Zhou, C.-W. Xie, J. Wu, and W. Lin, "ThiNet: Pruning CNN Filters for a Thinner Net," *IEEE Trans. Pattern Anal Mach. Intell.*, vol. 41, no. 10, pp. 2525–2538, Oct. 2019, doi: 10.1109/TPAMI.2018.2858232.

[18] L. Wang and K.-J. Yoon, "Knowledge Distillation and Student-Teacher Learning for Visual Intelligence: A Review and New Outlooks," *IEEE Trans. Pattern Anal Mach. Intell.*, vol. 44, no. 6, pp. 3048–3068, Jun. 2022, doi: 10.1109/TPAMI.2021.3055564.

[19] S. Naveen and M. R. Kounte, "Key Technologies and challenges in IoT Edge Computing," in *2019 Third Int. Conf. I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, IEEE, Dec. 2019, pp. 61–65. doi: 10.1109/I-SMAC47947.2019.9032541.

[20] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, Oct. 2021, doi: 10.1016/j.neucom.2021.07.045.

[21] F. Tung and G. Mori, "Deep Neural Network Compression by In-Parallel Pruning-Quantization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 3, pp. 568–579, Mar. 2020, doi: 10.1109/TPAMI.2018.2886192.

[22] S. Naveen and M. R. Kounte, "Machine Learning at Resource Constraint Edge Device Using Bonsai Algorithm," in *2020 Third Inter. Conf. Adv. Electron., Comput. Communic. (ICAECC)*, IEEE, Dec. 2020, pp. 1–6. doi: 10.1109/ICAECC50550.2020.9339514.

[23] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," Oct. 2015. [Online]. Available: https://arxiv.org/abs/1510.00149.

[24] K. Paupamah, S. James, and R. Klein, "Quantisation and Pruning for Neural Network Compression and Regularisation," in *2020 Int. SAUPEC/RobMech/PRASA Conf.*, IEEE, Jan. 2020, pp. 1–6. doi: 10.1109/SAUPEC/RobMech/PRASA48453.2020.9041096.

[25] S. Naveen and M. R. Kounte, "Memory optimization at Edge for Distributed Convolution Neural Network," *Trans. Emerg. Telecommunic. Technol.*, vol. 33, no. 12, Dec. 2022, doi: 10.1002/ett.4648.

[26] L. Guerra and T. Drummond, "Automatic Pruning for Quantized Neural Networks," in *2021 Dig. Im. Comput.: Techniq. Applic. (DICTA)*, IEEE, Nov. 2021, pp. 01–08. doi: 10.1109/DICTA52665.2021.9647074.

[27] S. Jung *et al.*, "Learning to Quantize Deep Networks by Optimizing Quantization Intervals With Task Loss," in *2019 IEEE/CVF Conf. Comp. Vision Patt. Recogn. (CVPR)*, IEEE, Jun. 2019, pp. 4345–4354. doi: 10.1109/CVPR.2019.00448.

[28] J. Kim, "Quantization Robust Pruning With Knowledge Distillation," *IEEE Access*, vol. 11, pp. 26419–26426, 2023, doi: 10.1109/ACCESS.2023.3257864.

[29] S. Yang *et al.*, "APQ: Automated DNN Pruning and Quantization for ReRAM-Based Accelerators," *IEEE Trans. Parallel Distr. Sys.*, vol. 34, no. 9, pp. 2498–2511, Sep. 2023, doi: 10.1109/TPDS.2023.3290010.

[30] C.-Y. Chang, K.-C. Chou, Y.-C. Chuang, and A.-Y. Wu, "E-UPQ: Energy-Aware Unified Pruning-Quantization Framework for CIM Architecture," *IEEE J. Emerg. Sel. Top Circ. Sys.*, vol. 13, no. 1, pp. 21–32, Mar. 2023, doi: 10.1109/JETCAS.2023.3242761.

[31] Y. Zhang, X. Wang, X. Jiang, Y. Yang, Z. Shen, and Z. Jia, "PQ-PIM: A pruning–quantization joint optimization framework for ReRAM-based processing-in-memory DNN accelerator," *J. Sys. Architec.*, vol. 127, Art. no. 102531, Jun. 2022, doi: 10.1016/j.sysarc.2022.102531.

[32] P. Hu, X. Peng, H. Zhu, M. M. S. Aly, and J. Lin, "OPQ: Compressing Deep Neural Networks with One-shot Pruning-Quantization," May 2022, [Online]. Available: https://doi.org/10.48550/arXiv.2205.11141