Jurnal Ilmu Komputer dan Informasi (Journal of Computer Science and Information)
18/1 (2025), 127-140. DOI: http://dx.doi.org/10.21609/jiki.v18i1.1443

Efficient Design and Compression of CNN Models for Rapid Character Recognition

Onesinus Saut Parulian

Department of Computer Science, Universitas Nusa Mandiri, Jakarta, Indonesia

E-mail: 14230003@nusamandiri.ac.id

Abstract

Convolutional Neural Networks (CNNs) are extensively utilized for image processing and recognition tasks; however, they often encounter challenges related to large model sizes and prolonged training times. These limitations present difficulties in resource-constrained environments that require rapid model deployment and efficient computation. This study introduces a systematic approach to designing lightweight CNN models specifically for character recognition, emphasizing the reduction of model complexity, training duration, and computational costs without sacrificing performance. Techniques such as hyperparameter tuning, model pruning, and post-training quantization (PTQ) are employed to decrease model size and enhance training speed. The proposed methods are particularly well-suited for deployment on edge computing platforms, such as Raspberry Pi, or embedded systems with limited resources. Our results demonstrate a reduction of over 80% in model size, decreasing from 43.73 KB to 6.25 KB, and a reduction of more than 45% in training time, decreasing from over 150 seconds to less than 80 seconds. This research highlights the potential for achieving a balance between efficiency and accuracy in CNN design for real-world deployment, addressing the increasing demand for streamlined deep learning models in resource-constrained environments.

Keywords: *Lightweight CNN; model optimization; efficient deep learning; character recognition;*

1. Introduction

The development and application of machine learning continue to grow, driven by the increasing demand for Artificial Intelligence (AI) solutions. Consequently, the use of deep learning is becoming more prevalent across various fields and applications. As deep learning models advance, there has been a significant rise in their parameter counts, latency, and the resources required for training, among other factors [1]. Convolutional neural networks (CNNs) are a type of artificial neural network that is widely utilized in numerous case studies, including Optical Character Recognition (OCR) [2-5]. Additionally, other variations of CNNs, such as the CTC-CNN model for speech recognition [6] and CNN-ECOC for handwritten image recognition [7], have also been employed in various applications [8-10].

Although convolutional neural networks (CNNs) are popular due to their excellent performance and versatility in addressing various problems, they face a significant challenge: achieving reliable model performance while keeping the model size manageable and optimizing training time.

In many implementations, deep learning applications often focus solely on objectives without adequately testing the effectiveness and efficiency of the constructed models. For instance, in CNN applications, using overly complex architectures for simple tasks, such as character recognition with well-structured datasets like MNIST [11], can lead to an unnecessarily large model size without improving performance.

This highlights the need for a comparison between complex models required for intricate tasks and simpler implementations that can avoid errors. Therefore, optimization techniques are crucial for eliminating unnecessary components in deep learning implementations, particularly in CNNs. This ensures that complex CNN architectures can be developed without concerns about model size and training duration, which can

be addressed prior to the model deployment process.

Several previous studies have investigated and applied various techniques to tackle optimization challenges, including model compression, model pruning [12-16], quantization [17-19], [20-21] and other approaches [22-26].

However, there has been limited research specifically focused on optimizing these indicators simultaneously for Convolutional Neural Networks (CNNs) without relying on stable existing (pre-trained) models. The goal is to reduce model size and training time while maintaining performance by integrating established techniques and methods.

A widely used and popular technique today is model pruning. Pruning involves removing unimportant parameters from a deep learning neural network to reduce the model's size and enhance its efficiency. Another commonly employed method is quantization, which lowers the computational and memory costs of inference by utilizing lower-precision data types, such as 8-bit integers (int8), instead of the standard 32-bit floating-point format (float32) for representing weights and activations.

The advantage of previous studies lies in their successful application of lightweight models to case studies using the proposed methods. However, there are areas that warrant further exploration, such as addressing the lack of comparative analysis between existing methods across various datasets utilizing architectures built on convolutional neural networks (CNNs). Additionally, it is worth investigating whether combining these methods yields better optimization results. Table 1 presents a summary of previous research related to optimization models, highlighting their advantages, disadvantages, and limitations.

Several approaches in recent research focus on enhancing lightweight CNN models to address complex datasets. These methods have been particularly successful in specific contexts, such as classification tasks with high computational efficiency, but gaps remain in their practical applications and experimental validations. For instance, GGM-VGG16, as proposed for pepper leaf disease recognition [24], combines Ghost modules, global pooling, and multi-scale convolutions, leading to an efficient 12.84 MB model.

However, training time and overall development processes lack detailed comparison metrics, limiting its application scope for time-critical scenarios. The work on lightweight CNN image recognition techniques [25] provides a theoretical basis through comprehensive reviews but lacks experimental validations, especially in optimizing model size and training time. Similarly, the utilization of knowledge distillation in AlexNet modification for image classification [22] shows memory efficiency but does not explore dataset variation or benchmark improvements pre- and post-distillation. Joint distillation methods for deepfake video detection [23] introduce effective pre-training and knowledge transfer strategies but overlook hyperparameter tuning and comparative performance evaluations with other state-of-the-art techniques.

Finally, neural architecture search combined with knowledge distillation for ISAR imaging tasks [26] achieves ultra-lightweight designs but does not examine augmentation effects or provide tuning parameters that could optimize model performance further.

While these studies have explored advanced techniques for lightweight CNN design, they focus on specialized, complex datasets such as high-resolution imagery or tasks requiring extreme computational demands. However, few address the practical challenges in simplifying models for balanced and moderately sized datasets while maintaining computational efficiency and rapid adaptability for diverse, resource-constrained environments.

To fill this gap, our research emphasizes efficient CNN designs specifically tailored to character recognition tasks with balanced datasets. Through hyperparameter tuning, model pruning, and post training quantization, we target real-world challenges by achieving substantial reductions in model size and training duration, advancing the field of lightweight CNNs for real-world applicability. This research aims to enhance techniques and methods applicable to Convolutional Neural Networks (CNNs) to reduce model size and accelerate training time while preserving the performance of the constructed CNN model.

The structure of this paper is as follows: Section 2 introduces method of this research; Section 3 provides results; Section 4 provides result visualizations. Section 5 compares and analyzes the experimental results among indicators and discusses the findings and limitations of the research; Section 6 conclude the research.

2. Method

This study employs a systematic approach to optimize Convolutional Neural Network (CNN) architectures for efficient image classification tasks. The methodology is divided into three main phases: data preprocessing, CNN model construction, and model optimization, as illustrated in Figure 1. Two distinct datasets,

MNIST [11] and Braille Character, were selected for training and validation. Each dataset was processed through a series of CNN models and optimized using techniques designed to reduce model size and training duration.

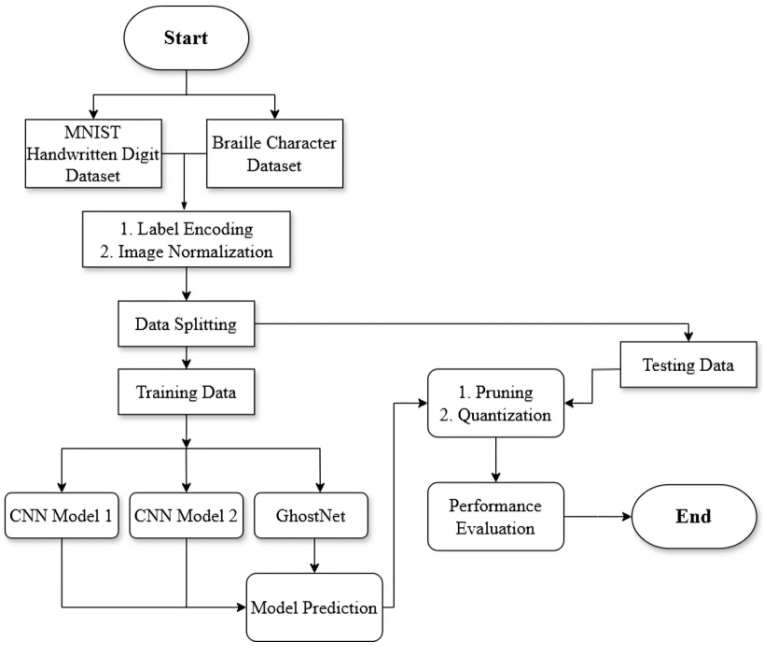

Figure 1. Proposed Method.

The datasets utilized in this study are (i) MNIST[11], a standard dataset for digit recognition, and (ii) a custom Braille character dataset, which presents a more complex classification challenge. Both datasets were preprocessed to ensure uniformity in image dimensions and pixel scaling, thereby enabling efficient input handling within the convolutional neural network (CNN) architecture. The images were normalized to reduce training variability and facilitate faster convergence. Three CNN model architectures were constructed for each dataset. Figure 2 and 3 are example images of datasets.

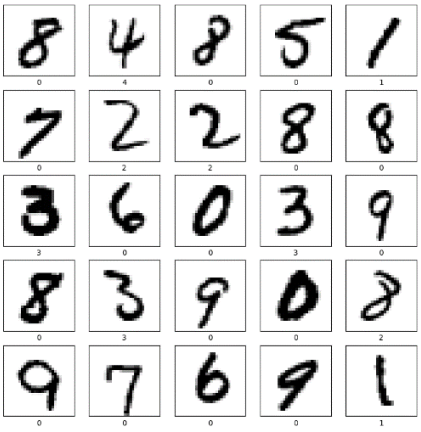

Figure 2. MNIST dataset

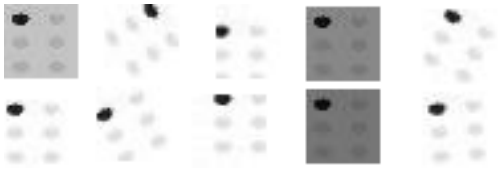

Figure 3. Braille character dataset

The CNNs were designed with varying depths and configurations to balance accuracy and computational efficiency. Two custom architectures (Models 1 and 2; see Appendices A and B) and the GhostNet architecture [24] were evaluated on accuracy, inference time, and model complexity. Lightweight models with reduced training time were achieved using hyperparameter tuning, low magnitude pruning, and post training quantization.

Pruning techniques were applied to iteratively remove redundant or less significant weights from the CNN models. This reduction in parameter count aimed to lower memory usage and computational load while retaining core model performance. Pruning levels were carefully tuned to balance model efficiency and accuracy retention. The process is formulated as equation 1.

$$W' = prune(W, s) \quad (1)$$

As formulated in equation 1, W is the original set of weights in the model, s is the sparsity level

(0.1 for 10% in this case), sparsity is number of zero-valued elements in a tensor. W' is the pruned set of weights after pruning process. In sequence this is the steps of pruning process in this research:

1. First Step: Initialize Model - Define Pruning Parameters - Apply Pruning - Compile Pruned Model.
2. Second Step: Strip Pruning Components - Convert to TFLite - Save Model - Measure Model Size.

This study used low magnitude pruning in Keras with pruning schedule value set to constant sparsity=0.1, begin step=0, and frequency=100. After pruning process, the pruned model is converted and exported using TFLite and represented as Kilobyte (KB) as formulated in equation 2.

$$Model\ Size = \frac{Size\ of\ tflite\_model}{1024}\ KB \qquad (2)$$

Post Training Quantization was performed to compress model weights, using 8-bit integers (int8) in place of 32-bit floating-point representations. This technique significantly reduces memory requirements and enables faster computation during inference. Quantization was implemented after model training, with evaluations conducted to ensure no substantial accuracy loss occurred due to lower precision. Pruning and Quantization process to the model is formulated as Figure 4.

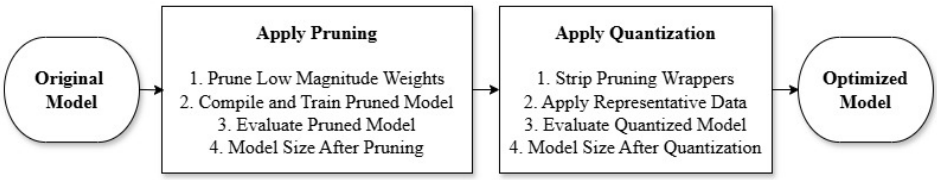

Figure 4. Pruning and Quantization Process Flow

Figure 4 outlines the pruning and quantization process flow for optimizing a model, detailing each step from preparation to the result of quantization. This structured flow ensures a systematic approach to reducing the model size and optimizing performance. The main stages are described as follows:

1. Original Model: Represents the initial model before any optimization steps are applied.

2. Apply Pruning:

- Prune Low Magnitude Weights: Removes weights with low magnitude to reduce the model size.
- Compile and Train Pruned Model: Compiles and trains the model after pruning to ensure it maintains performance.
- Evaluate Pruned Model: Assesses the performance of the pruned model.
- Model Size After Pruning: Measures the size of the model after pruning to evaluate the reduction achieved.

3. Apply Quantization:

- Strip Pruning Wrappers: Removes any pruning-related artifacts from the model.
- Apply Representative Data: Uses representative data to fine-tune the quantization process, ensuring accuracy is preserved.
- Evaluate Quantized Model: Assesses the performance of the quantized model.

4. Optimized Model: Represents the final optimized model ready for deployment, having undergone both pruning and quantization to reduce size and improve performance.

In this study, a hyperparameter tuning strategy is introduced to enhance training efficiency by manually configuring key parameters such as early stopping, learning rate scheduling, and batch size adjustments as detailed in Table 1. These techniques collectively aim to reduce training time and optimize performance across epochs.

Table 1. Hyperparameter and callbacks

ReduceLROnPlateau	This callback halves the learning rate (factor=0.5) if the monitored metric (val_loss) stagnates for 2 epochs (patience), with a minimum limit of 1e-6, ensuring adaptive response to training plateaus.
Batch Size Configuration	A batch size of 128 was chosen to ensure efficient training updates, balancing computational efficiency and training speed based on dataset and resource constraints.
Early Stopping	Early stopping halts training if validation loss shows no improvement for 3 epochs (patience=3), preventing overfitting and restoring the best model state (restore_best_weights=True).

These mechanisms enable the model to modify specific aspects of its training behavior according to predefined parameters. Although this setup

necessitates careful tuning based on the dataset, it fosters efficient and stable training, ultimately producing a robust final model suitable for resource-constrained deployments. While initial parameters must be customized for the specific problem, this approach strikes a balance between manual control and training efficiency.

The models were evaluated based on their accuracy, size, and training time. Optimization techniques were applied both individually and in combination to identify the most effective strategies for reducing computational burden. A comparative analysis was conducted against unoptimized models to quantify improvements in terms of size reduction and training time acceleration. Performance metrics, including parameter count, inference latency, and accuracy, were recorded for each optimized model.

3. Results

The experimental results presented in Table 2 provide a comparative result of model performance before and after the application of pruning and quantization. Additionally, the impact of the proposed hyperparameter tuning approach is evaluated in comparison to three baseline models: Model 1, Model 2, and GhostNet.

Table 2. Result of optimizing dataset 1 (MNIST)

Models	Process	Params	Size	Time	Training Accuracy	Training Loss	Testing Accuracy	Testing Loss
Model 1 [Appendix A]	No Optimization	834	43.73 KB	185s	94.14%	0.19	94.73%	0.16
	After Pruning	1644	7.01 KB	152s	95.53%	0.14	96.13%	0.12
	After Pruning + Quantization	834	6.25 KB	153s	95.89%	0.12	96.29%	0.12
Model 2 [Appendix B]	No Optimization	121,930	1473 KB	225s	99.82%	0.0055	99.15%	0.037
	After Pruning	243,701	479 KB	227s	99.89%	0.0027	99.02%	0.054
	After Pruning + Quantization	121,930	242 KB	204s	99.89%	0.0034	99.13%	0.06
GhostNet [Appendix C]	No Optimization	14,954	380 KB	754s	99.13%	0.0287	98.51%	0.0451
	After Pruning	27,175	62 KB	571s	99.61%	0.0126	98.68%	0.0394
	After Pruning + Quantization	14,954	41 KB	543s	99.61%	0.0112	98.76%	0.0487
Proposed Method [Appendix A]	No Optimization	834	43 KB	50s	95.67%	0.13	95.22%	0.14
	After Pruning	1,644	7.01 KB	49s	95.65%	0.13	93.97%	0.18
	After Pruning + Quantization	834	6.25 KB	80s	96.15%	0.12	95.52%	0.13

The proposed method demonstrates significant model compression while maintaining accuracy. For instance, in Model 1, the parameter count remained consistent at 834 after applying pruning and quantization, while the model size was reduced from 43.73 KB to 6.25 KB. The compression achieved with the proposed method illustrates notable efficiency in model size. Additionally, this method results in reduced training and inference times. The optimized model required only 49.15 seconds, compared to the initial times of 185 seconds, 225 seconds, and 754 seconds for Models 1, 2, and 3, respectively. This indicates faster training without a substantial compromise in accuracy. The accuracy of the models after pruning and quantization remained stable. For Model 2, the training accuracy reached 99.89% post-optimization, with only a slight increase in loss. Testing accuracy exhibited a similar trend, maintaining high levels (e.g., 96.29% in Model 1). This supports the conclusion that the optimization strategy effectively preserves model performance while minimizing resource usage.

Before optimization, Model 1 achieved a training accuracy of 94.14% and a testing accuracy of 94.73%. After applying pruning and quantization, both training and testing accuracies improved to 95.89% and 96.29%, respectively, while also reducing the model size and training time. For Model 2, the training accuracy increased to 99.89%, accompanied by a significant reduction in model size from 1,473 KB to 242 KB following optimization. Despite its larger initial parameter count, GhostNet also benefited from pruning and quantization, achieving a testing accuracy of 98.76% and reducing its model size from 380 KB to 41.38 KB.

The proposed approach demonstrated consistent improvements across all metrics, particularly in reducing training time and maintaining high accuracy. It outperformed the baseline models in both model compression and training efficiency, making it suitable for resource-constrained environments. The combination of pruning, quantization, and hyperparameter tuning proved effective for optimizing CNN models. The results indicate that it is possible to achieve compact, high-performing

models that maintain competitive accuracy while significantly reducing computational requirements.

The Braille dataset is processed using the same methodology to provide an alternative perspective on the proposed method and to validate that the solution is effective for different datasets within the same study case as detailed in Table 3.

Table 3. Result of optimizing dataset 2 (Braille)

Models	Process	Params	Size	Time	Training Accuracy	Training Loss	Testing Accuracy	Testing Loss
Model 1 [Appendix D]	No Optimization	587,834	6973 KB	105s	98.97%	0.041	95.19%	0.22
	After Pruning	1,171,537	2291 KB	47s	98.97%	0.024	92.95%	0.3
	After Pruning + Quantization	587,834	**1151 KB**	82s	97.94%	0.06	95.19%	0.35
Model 2 [Appendix E]	No Optimization	587,834	4678 KB	55s	98.63%	0.04	93.91%	0.27
	After Pruning	1,171,537	2292	63s	**99.54%**	0.01	93.27%	0.44
	After Pruning + Quantization	587,834	1152 KB	37.99s	99.31%	0.01	93.58%	0.5
GhostNet [Appendix C]	No Optimization	88,256	1185 KB	60s	99.88%	0.09	44%	36.46
	After Pruning	173,213	349 KB	59.21s	99.77%	0.06	42%	109.81
	After Pruning + Quantization	88,256	184 KB	33.49s	100%	0.04	42.27%	158
Proposed Method [Appendix D]	No Optimization	587,834	6973 KB	105s	98.97%	0.041	95.19%	0.22
	After Pruning	1,171,537	2292	95.47s	99.54%	0.02	91.35%	0.52
	After Pruning + Quantization	587,834	1151 KB	**42.34s**	98.97%	0.02	91.03%	0.96

The result from Braille dataset shows that the GhostNet architecture is not suitable for this dataset, as it resulted in overfitting, where the training performance is good but the testing performance is poor.

4. Visualizations

Results visualization for dataset 1 are shown in Figure 5- 8.

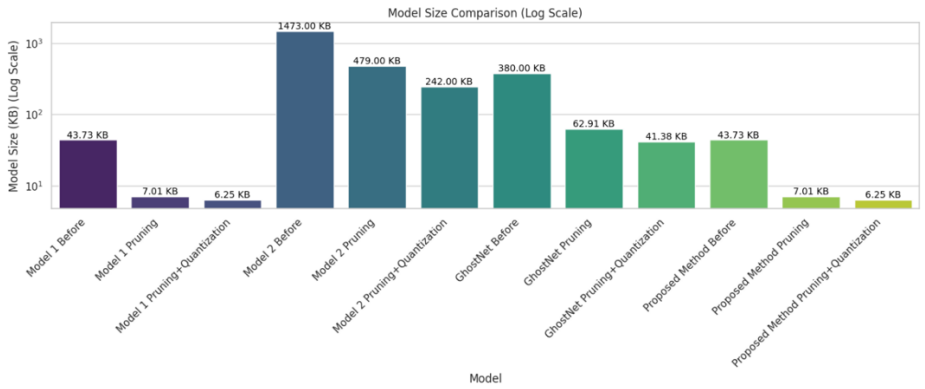

Figure 5. Model Size Comparison MNIST dataset

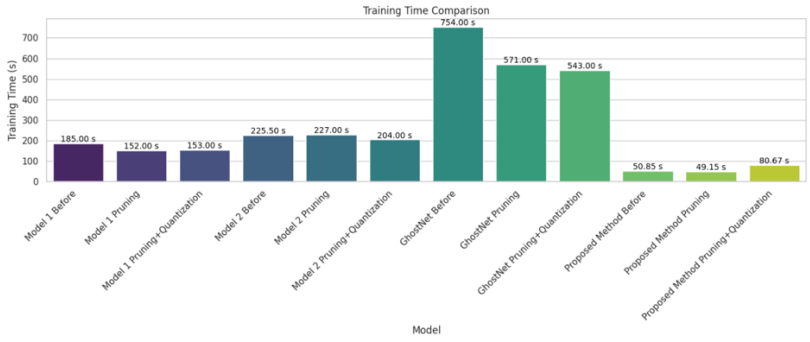

Figure 6. Training Time Comparison MNIST dataset

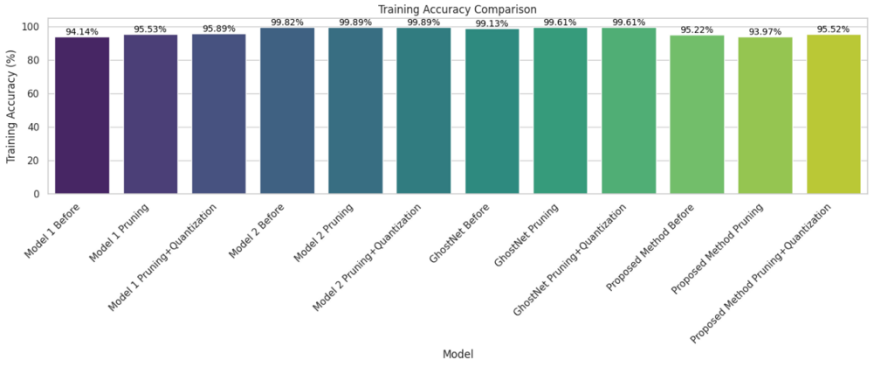

Figure 7. Training accuracy Comparison MNIST dataset

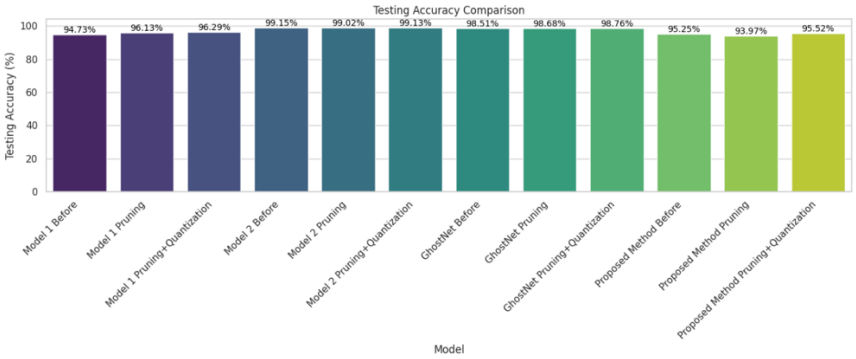

Figure 8. Testing accuracy Comparison MNIST dataset

5. Analysis and Discussions

This section combines the analysis and discussion of the results to provide a holistic view of the findings and their implications. The proposed optimization strategies, including hyperparameter tuning, pruning, and quantization, significantly reduced model size and training time while maintaining robust accuracy.

The architectural design of the models was tailored to achieve a balance between accuracy and computational efficiency. The lightweight CNNs, particularly Model 1 and Model 2, benefited from the proposed optimizations, with notable parameter reductions. For instance, Model 1's parameters were reduced to 834, and its size compressed from 43.73 KB to 6.25 KB. Similarly, Model 2's size dropped from 1473 KB to 242 KB. Despite these reductions, the testing accuracy of Model 1 remained high at 95.52%, nearly matching the uncompressed version.

Interestingly, the quantized version of Model 1 exhibited slightly higher accuracy compared to the non-quantized model, which is atypical for post-training quantization. This improvement can be attributed to the pruning and hyperparameter tuning processes, which likely removed noisy parameters and enhanced generalization, thereby making the quantized model better suited for inference on the testing dataset. Such results emphasize the synergy between pruning and quantization in improving performance beyond mere compression.

The proposed method significantly shortened training time compared to baseline models. Model 1's training duration decreased from 185 seconds to 80.67 seconds—a 56% reduction. These results stem from tuning parameters like batch size and implementing learning rate schedules, which stabilized the training process and improved convergence. Additionally, the integration of pruning-specific callbacks during training increased computational overhead temporarily, but this was offset by improved efficiency in subsequent epochs.

GhostNet, while inherently efficient, suffered from overfitting on Dataset 2, performing well in training but failing to generalize during testing. In contrast, Models 1 and 2, with simpler architectures, demonstrated robust performance, particularly after hyperparameter tuning and pruning. Early stopping and learning rate scheduling were instrumental in controlling overfitting by dynamically adapting the training process. The consistency in training (95.89% accuracy) and testing performance (95.52% accuracy) of Model 1 highlights the effectiveness of these strategies.

The substantial reduction in model size and complexity makes the proposed method ideal for deployment in resource-constrained environments. For example, Model 1's compressed size of 6.25 KB and Model 2's 242 KB make them suitable for edge devices such as mobile sensors and embedded systems. This aligns with the study's objectives of addressing memory and computational limitations while retaining high performance.

While the proposed method showed promising results, manual hyperparameter tuning requires dataset-specific adjustments and limits scalability to more complex datasets. Future work should explore automated optimization techniques, such as reinforcement learning or meta-optimization, to generalize the approach. Additionally, validating the method on high-resolution, imbalanced, or noisy datasets could further test its robustness.

By integrating pruning, quantization, and tailored hyperparameter tuning, this study achieved an effective balance between resource efficiency and model performance. These findings provide a pathway for deploying lightweight CNNs in real-world applications while addressing scalability challenges in future work.

6. Conclusions

This study presents an optimization method for convolutional neural networks (CNNs) aimed at creating efficient architectures by utilizing pruning, quantization, and hyperparameter adjustments. The results demonstrate that the proposed method effectively reduces model size and training time while maintaining high accuracy, addressing the demand for lightweight models in resource-constrained environments such as the Internet of Things (IoT) and edge devices. Although the method has proven effective, future research could focus on large models and datasets to try the workflow applicability for real word challenges. Overall, this study provides a practical and efficient approach to optimizing CNNs, striking a balance between computational efficiency and model performance.

References

[1] G. Menghani, "Efficient Deep Learning: A Survey on Making Deep Learning Models Smaller, Faster, and Better," 2023. doi: 10.1145/3578938.

[2] S. Alghyaline, "Optimised CNN Architectures for Handwritten Arabic Character Recognition," *Computers, Materials and Continua*, vol. 79, no. 3, pp. 4905–4924, 2024, doi: 10.32604/cmc.2024.052016.

[3] M. Sinthuja, C. G. Padubidri, G. S. Jayachandra, M. C. Teja, and G. S. P. Kumar, "Extraction of Text from Images Using Deep Learning," *Procedia Comput Sci*, vol. 235, no. 2023, pp. 789–798, 2024, doi: 10.1016/j.procs.2024.04.075.

[4] G. Ahmed *et al.*, "Recognition of Urdu Handwritten Alphabet Using Convolutional Neural Network (CNN)," *Computers, Materials and Continua*, vol. 73, no. 2, pp. 2967–2984, 2022, doi: 10.32604/cmc.2022.029314.

[5] S. H. Ali and M. B. Abdulrazzaq, "KurdSet: A Kurdish Handwritten Characters Recognition Dataset Using Convolutional Neural Network," *Computers, Materials and Continua*, vol. 79, no. 1, pp. 429–448, 2024, doi: 10.32604/cmc.2024.048356.

[6] W. T. Sung, H. W. Kang, and S. J. Hsiao, "Speech Recognition via CTC-CNN Model," *Computers, Materials and Continua*, vol. 76, no. 3, pp. 3833–3858, 2023, doi: 10.32604/cmc.2023.040024.

[7] M. B. Bora, D. Daimary, K. Amitab, and D. Kandar, "Handwritten Character Recognition from Images using CNN-ECOC," *Procedia Comput Sci*, vol. 167, no. 2019, pp. 2403–2409, 2020, doi: 10.1016/j.procs.2020.03.293.

[8] C. S. Wei, S. L. Wang, N. T. Foo, and D. A. Ramli, "A CNN based handwritten numeral recognition model for four arithmetic operations," *Procedia Comput Sci*, vol. 192, pp. 4416–4424, 2021, doi: 10.1016/j.procs.2021.09.218.

[9] S. Arooj, S. Altaf, S. Ahmad, H. Mahmoud, and A. S. N. Mohamed, "Enhancing sign language recognition using CNN and SIFT: A case study on Pakistan sign language," *Journal of King Saud University - Computer and Information Sciences*, vol. 36, no. 2, p. 101934, 2024, doi: 10.1016/j.jksuci.2024.101934.

[10] S. D. Pande *et al.*, "Digitization of handwritten Devanagari text using CNN transfer learning – A better customer service support," *Neuroscience Informatics*, vol. 2, no. 3, p. 100016, 2022, doi: 10.1016/j.neuri.2021.100016.

[11] Li Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]," *IEEE Signal Process Mag*, vol. 29, no. 6, pp. 141–142, Nov. 2012, doi: 10.1109/MSP.2012.2211477.

[12] H. Louati, A. Louati, E. Kariri, and S. Bechikh, "Optimizing Deep Learning for Computer-Aided Diagnosis of Lung Diseases: An Automated Method Combining Evolutionary Algorithm, Transfer Learning, and Model Compression," *Computer Modeling in Engineering & Sciences*, vol. 138, no. 3, pp. 2519–2547, 2024, doi: 10.32604/cmes.2023.030806.

[13] A. Li, M. Markovic, P. Edwards, and G. Leontidis, "Model pruning enables localized and efficient federated learning for yield forecasting and data sharing," *Expert Syst Appl*, vol. 242, no. November 2023, p. 122847, 2024, doi: 10.1016/j.eswa.2023.122847.

[14] K. Vidya, P. Ramesh, H. Viknesh, and S. Devanand, "Compressed Deepfake Detection using Spatio-Temporal Approach with Model Pruning," *Procedia Comput Sci*, vol. 230, pp. 436–444, 2023, doi: 10.1016/j.procs.2023.12.099.

[15] C. G. Pachon, J. O. Pinzon-Arenas, and D. Ballesteros, "FlexiPrune: A Pytorch tool for flexible CNN pruning policy selection," *SoftwareX*, vol. 27, no. June, p. 101858, 2024, doi: 10.1016/j.softx.2024.101858.

[16] L. E. Pommé, R. Bourqui, R. Giot, J. Vallet, and D. Auber, "NetPrune: A sparklines visualization for network pruning," *Visual Informatics*, vol. 7, no. 2,

pp. 85–99, 2023, doi: 10.1016/j.visinf.2023.04.001.

[17] J. N. Kolf, J. Elliesen, N. Damer, and F. Boutros, "MixQuantBio: Towards extreme face and periocular recognition model compression with mixed-precision quantization," *Eng Appl Artif Intell*, vol. 137, no. PB, p. 109114, 2024, doi: 10.1016/j.engappai.2024.109114.

[18] X. Yang, E. del Rey Castillo, Y. Zou, and L. Wotherspoon, "UAV-deployed deep learning network for real-time multi-class damage detection using model quantization techniques," *Autom Constr*, vol. 159, no. December 2023, p. 105254, 2024, doi: 10.1016/j.autcon.2023.105254.

[19] L. Wei, Z. Ma, and C. Yang, "Activation Redistribution Based Hybrid Asymmetric Quantization Method of Neural Networks," *Computer Modeling in Engineering & Sciences*, vol. 138, no. 1, pp. 981–1000, 2024, doi: 10.32604/cmes.2023.027085.

[20] F. He, K. Ding, D. Yan, J. Li, J. Wang, and M. Chen, "A Novel Quantization and Model Compression Approach for Hardware Accelerators in Edge Computing," *Computers, Materials and Continua*, vol. 80, no. 2, pp. 3021–3045, 2024, doi: 10.32604/cmc.2024.053632.

[21] M. Goswami, S. Mohanty, and P. K. Pattnaik, "Optimization of machine learning models through quantization and data bit reduction in healthcare datasets," *Franklin Open*, vol. 8, no. April, p. 100136, 2024, doi: 10.1016/j.fraope.2024.100136.

[22] A. Kuldashboy, S. Umirzakova, S. Allaberdiev, R. Nasimov, A. Abdusalomov, and Y. I. Cho, "Efficient image classification through collaborative knowledge distillation: A novel AlexNet modification approach," *Heliyon*, vol. 10, no. 14, p. e34376, 2024, doi: 10.1016/j.heliyon.2024.e34376.

[23] X. Xu, S. Tang, M. Zhu, P. He, S. Li, and Y. Cao, "A novel model compression method based on joint distillation for deepfake video detection," *Journal of King Saud University - Computer and Information Sciences*, vol. 35, no. 9, p. 101792, 2023, doi: 10.1016/j.jksuci.2023.101792.

[24] Y. Fu, L. Guo, and F. Huang, "A lightweight CNN model for pepper leaf disease recognition in a human palm background," *Heliyon*, vol. 10, no. 12, p. e33447, 2024, doi: 10.1016/j.heliyon.2024.e33447.

[25] Y. Liu, J. Xue, D. Li, W. Zhang, T. K. Chiew, and Z. Xu, "Image recognition based on lightweight convolutional neural network: Recent advances," *Image Vis Comput*, vol. 146, no. April, p. 105037, 2024, doi: 10.1016/j.imavis.2024.105037.

[26] H. Yang, Y. sheng Zhang, C. bin Yin, and W. zhe Ding, "Ultra-lightweight CNN design based on neural architecture search and knowledge distillation: A novel method to build the automatic recognition model of space target ISAR images," *Defence Technology*, vol. 18, no. 6, pp. 1073–1095, 2022, doi: 10.1016/j.dt.2021.04.014.

Appendix A: Simple CNN Model 1

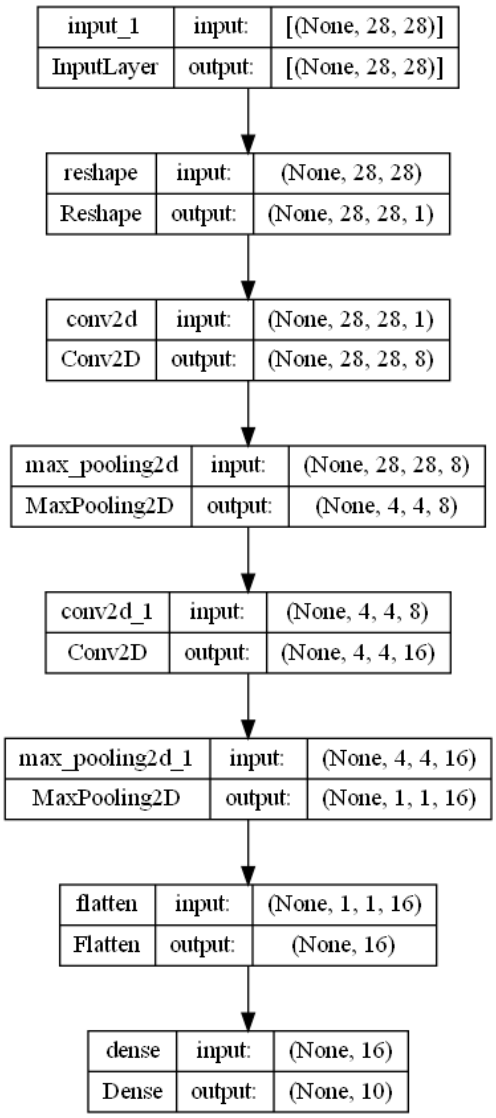

Figure A. Simple CNN Model 1 for MNIST dataset

Appendix B: Simple CNN Model 2

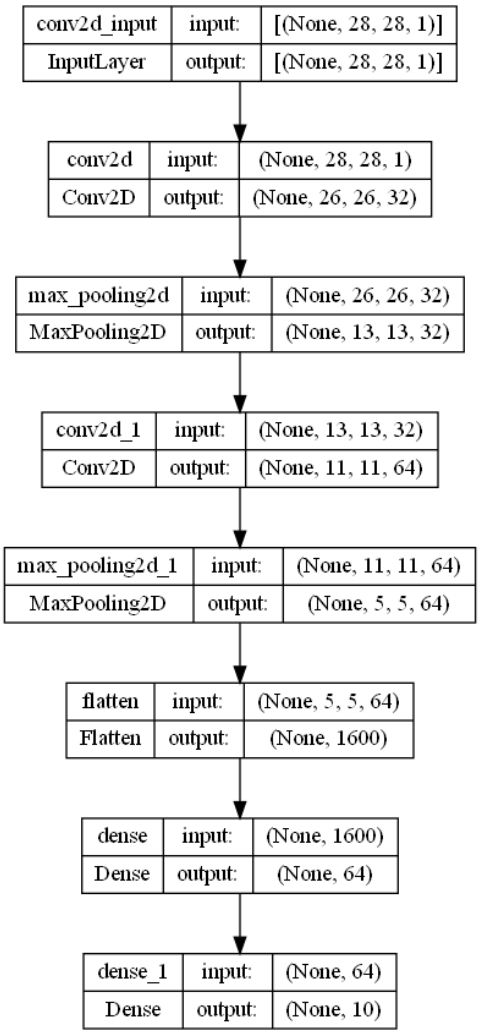

Figure B. Simple CNN Model 2 for MNIST dataset

Appendix C: GhostNet Model

No	Name	Type	Shape
1	input_1	InputLayer	[(None, 28, 28, 1)]
2	conv2d	Conv2D	(None, 28, 28, 16)
3	batch_normalization	BatchNormalization	(None, 28, 28, 16)
4	re_lu	ReLU	(None, 28, 28, 16)
5	conv2d_1	Conv2D	(None, 28, 28, 16)
6	batch_normalization_1	BatchNormalization	(None, 28, 28, 16)
7	re_lu_1	ReLU	(None, 28, 28, 16)
8	depthwise_conv2d	DepthwiseConv2D	(None, 28, 28, 16)
9	batch_normalization_2	BatchNormalization	(None, 28, 28, 16)
10	re_lu_2	ReLU	(None, 28, 28, 16)
11	conv2d_2	Conv2D	(None, 28, 28, 16)
12	batch_normalization_3	BatchNormalization	(None, 28, 28, 16)
13	re_lu_3	ReLU	(None, 28, 28, 16)
14	concatenate	Concatenate	(None, 28, 28, 32)
15	max_pooling2d	MaxPooling2D	(None, 14, 14, 32)
16	conv2d_3	Conv2D	(None, 14, 14, 32)
17	batch_normalization_4	BatchNormalization	(None, 14, 14, 32)
18	re_lu_4	ReLU	(None, 14, 14, 32)
19	depthwise_conv2d_1	DepthwiseConv2D	(None, 14, 14, 32)
20	batch_normalization_5	BatchNormalization	(None, 14, 14, 32)
21	re_lu_5	ReLU	(None, 14, 14, 32)
22	conv2d_4	Conv2D	(None, 14, 14, 32)
23	batch_normalization_6	BatchNormalization	(None, 14, 14, 32)
24	re_lu_6	ReLU	(None, 14, 14, 32)
25	concatenate_1	Concatenate	(None, 14, 14, 64)
26	max_pooling2d_1	MaxPooling2D	(None, 7, 7, 64)
27	conv2d_5	Conv2D	(None, 7, 7, 64)
28	batch_normalization_7	BatchNormalization	(None, 7, 7, 64)
29	re_lu_7	ReLU	(None, 7, 7, 64)
30	depthwise_conv2d_2	DepthwiseConv2D	(None, 7, 7, 64)
31	batch_normalization_8	BatchNormalization	(None, 7, 7, 64)
32	re_lu_8	ReLU	(None, 7, 7, 64)
33	conv2d_6	Conv2D	(None, 7, 7, 64)
34	batch_normalization_9	BatchNormalization	(None, 7, 7, 64)
35	re_lu_9	ReLU	(None, 7, 7, 64)
36	concatenate_2	Concatenate	(None, 7, 7, 128)
37	global_average_pooling2d	GlobalAveragePooling2D	(None, 128)
38	dense	Dense	(None, 10)
39	input_1	InputLayer	[(None, 28, 28, 1)]
40	conv2d	Conv2D	(None, 28, 28, 16)

Table A. GhostNet Model to be applied to both datasets

Appendix D: CNN Model 1 for Braille Dataset

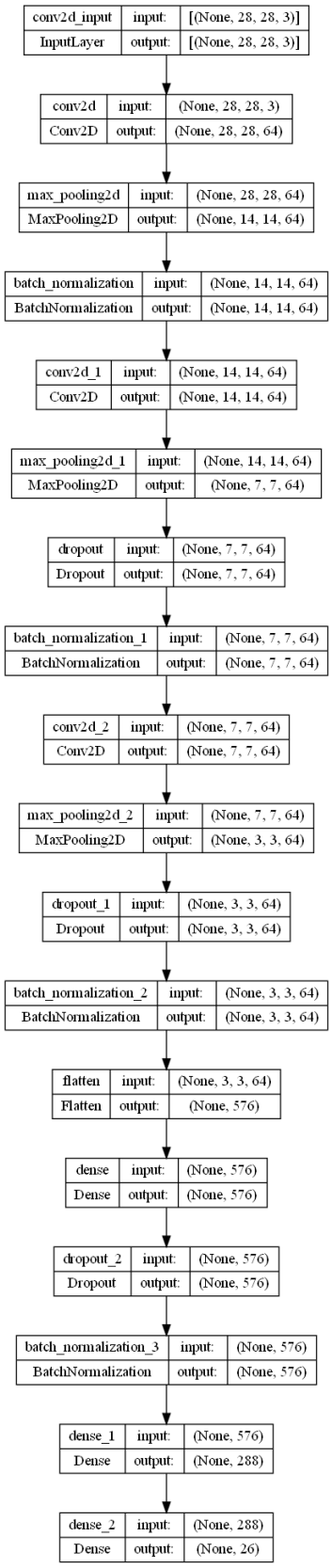

Figure C. CNN Model 1 using Adam optimizer for Braille dataset

Appendix E: CNN Model 2 for Braille Dataset

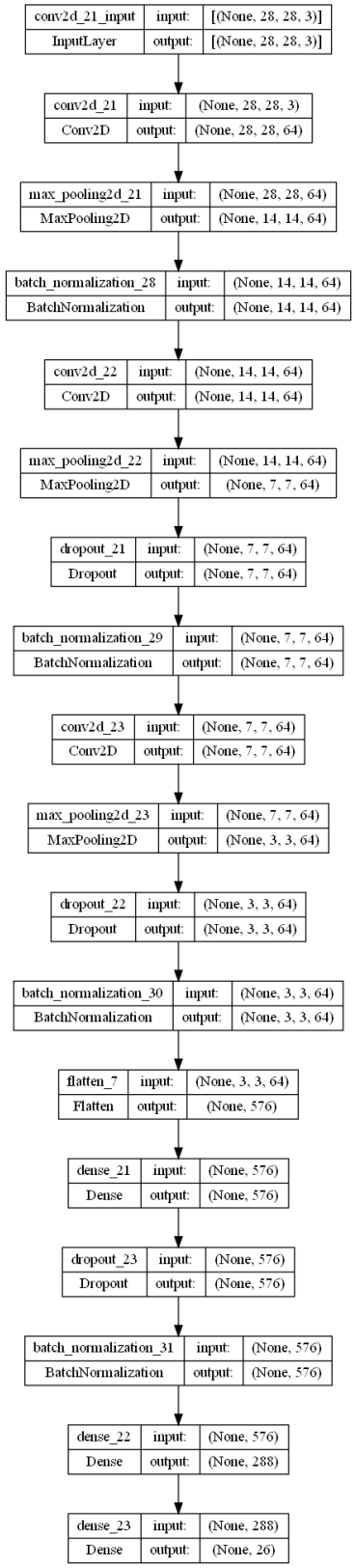

Figure D. CNN Model 2 using RMSprop optimizer for Braille dataset