ISSN 2320 - 2602
Volume 13 No. 8, August 2024
International Journal of Advances in Computer Science and Technology
Available Online at http://www.warse.org/IJACST/static/pdf/file/ijacst021382024.pdf
https://doi.org/10.30534/ijacst/2024/021382024

A Model for Reduction of Time and Space Complexity on Edge Devices

Taylor, Onate Egerton[1], Bumotu Braye Christy[2], Anireh, Vincent Ike Emeka[3]
[1]Department of Computer Science, Rivers State University, Nigeria, taylor.onate@ust.edu.ng
[2]Department of Computer Science, Rivers State University, Nigeria, anireh.ike@ust.edu.ng
[3]Department of Computer Science, Rivers State University, Nigeria, meetbraye@gmail.com

Received Date : June 4 , 2024 Accepted Date : July 29, 2024 Published Date : August 07, 2024

ABSTRACT

In the realm of edge computing, a paradigm emphasizing decentralized computational tasks, the interplay between time and space complexity holds immense significance. Time complexity denotes the duration required for an algorithm's execution, while space complexity concerns the memory or storage demand throughout the process. The evaluation entails a comparative analysis between a conventional non-quantized model and its quantized counterpart, focusing on accuracy, memory utilization, and runtime. The non-quantized model exhibits commendable learning performance, achieving a 96% accuracy rate during training but experiencing a marginal decrease to 90% in testing. Conversely, the quantized model sustains competitive accuracy, attaining 98% in both training and testing phases. The architecture of the quantized model, characterized by diminished numerical precision, emerges as a pivotal factor in minimizing both memory footprint and computational requirements. Graphical analyses unveil that despite a slight increase in loss during validation, the quantized model displays robust learning and generalization capabilities from the training dataset. The comparative analysis emphasizes the benefits of quantization, emphasizing decreased memory utilization (3kb), faster runtime, and, in specific cases, improved accuracy (96%). This thesis provides valuable perspectives on the effectiveness of quantization in optimizing Convolutional Neural Network (CNN) models for deployment on edge devices with limited resources. The evaluation metrics employed include memory usage reduction, runtime speed, and accuracy enhancement 96%.

Key words : Time and space complexity, edge devices, convolutional neural network, Domain Knowledge

1. INTRODUCTION

Edge devices refer to the various hardware devices that are connected to the Internet of Things (IoT) ecosystem, which are located on the "edge" of the network. These devices are usually small, low-cost, and have limited processing capabilities. They are designed to perform simple tasks like data collection, processing, and transmission. The data collected by these devices is then sent to the cloud or other central servers for further analysis and processing. Edge devices have become increasingly popular in recent years due to their ability to process data locally, reducing latency and improving overall performance. Edge devices can be broadly categorized into two types: sensors and actuators. Sensors are devices that collect data from the environment, such as temperature, humidity, or light. They are often used in applications like home automation, industrial monitoring, and healthcare. Actuators, on the other hand, are devices that act upon the environment, such as turning on a light, opening a valve, or activating a motor. Actuators are commonly used in home automation, industrial control, and robotics [1].

Edge computing has garnered significant attention for its potential to reduce latency, enhance privacy, and improve overall system efficiency by processing data closer to its source. A critical aspect in edge computing is reducing time and space complexity on edge devices to enable efficient processing. Several studies have focused on developing models and algorithms to address this challenge. For example, [2] introduced ICONet, a lightweight network with reduced time complexity suitable for implementation on edge devices. Similarly, [3] discussed optimizing Deep Neural Networks (DNNs) through model partitioning to enhance performance on edge devices by dividing the model into smaller sub-modules that can run in parallel.

One of the most significant advantages of edge devices is their ability to perform real-time analysis and decision-making. By processing data locally, these devices can quickly respond to changes in the environment without the need for centralized processing. This is particularly important in applications like industrial automation and robotics, where delays can lead to safety hazards or decreased efficiency. Edge computing is becoming increasingly important as more and more devices are connected to the internet, and the amount of data generated continues to grow. There are several challenges associated with edge computing, including limited processing power, memory, and storage. These limitations can make it difficult to process large amounts of data or perform complex computations. Additionally, edge devices are often located in harsh environments, which can lead to reliability and maintenance issues. To address these challenges, researchers are exploring new hardware architectures, software optimizations, and communication protocols [4].

Time and space complexity are two important factors to consider when developing algorithms for edge devices. Edge devices are typically resource-constrained, meaning that they have limited processing power, memory, and energy. Therefore, it is essential to optimize algorithms for both time and space complexity to ensure efficient and effective performance. Time complexity refers to the amount of time it takes for an algorithm to execute as a function of the input size. As edge devices have limited processing power, it is important to minimize the time complexity of algorithms to reduce the amount of time taken to process data. This can be achieved by using algorithms that have a low time complexity, such as those with a linear or logarithmic time complexity [5].

To further enhance the efficiency of edge devices, researchers have explored techniques such as task offloading and deep reinforcement learning. [6] proposed a novel task offloading algorithm based on deep reinforcement learning to optimize task scheduling, transmit power of IoT devices, and computing resource allocation of edge servers, aiming to reduce latency and energy consumption. Additionally, [7] suggested partitioning and distributing layer information across multiple edge devices to reduce computation and data load on individual devices, thereby improving overall efficiency.

Moreover, deploying neural network models on edge devices has been a focus of recent research efforts [8] highlighted the increasing popularity of deploying neural network models on edge devices to reduce response time and enhance data privacy. By automating quantization and retraining, models can be optimized for edge deployment, as discussed by [8]. Furthermore, [9] emphasized the importance of layer optimization and parameter reduction in models like Tiny-YOLOv3 to reduce

2. LITERATURE REVIEW

In their study, [10] introduce an innovative multistage pruning method that effectively decreases the complexity of CNN models while maintaining performance levels similar to other pruning strategies. A pre-existing Convolutional Neural Network (CNN) model is utilised as a baseline reference for ECG categorization. The proposed technique, with a sparsity of 60%, obtains an accuracy of 97.7% and an F1 score of 93.59% for ECG classification tasks. The accuracy and F1 Score have improved by 3.3% and 9% respectively, compared to the typical strategy of trimming with fine-tuning. In comparison to the baseline model, we also attain a reduction of 60.4% in the complexity of the run-time.

[11] have developed a tri-design approach for implementing MOT (Multiple Object Tracking) on edge devices. This approach utilises aggressive data reduction, model compression, and ultra-low-power hardware innovation to produce an algorithm that is both hardware-aware and ultra-lightweight. The authors validate the efficacy of the proposed tri-design through comprehensive experiments. Our technique on Alveo U50 FPGAs outperforms the state-of-the-art MOT baseline in several

In their study, [12] examine three primary research domains related to on-device computation: quantization, pruning, and network architecture design. The three strategies facilitate the deployment of a DNN model on edge devices for real-time computation and storage, primarily by reducing computation and space complexity. Furthermore, these techniques have the potential to enable the use of deep neural networks (DNNs) in industrial Internet of Things (IoT) devices.

[13] presents an extensive examination of different methods for improving the computational efficiency of deep learning inference on edge devices. The authors explore various methodologies, including network pruning, quantization, and knowledge distillation. The report also presents a comparative analysis of the efficacy of different methodologies on diverse edge devices.

[4] presents an innovative method to decrease the time and space complexity in edge devices. The authors present a novel method that decreases the time and space requirements of edge devices while upholding a high level of accuracy.

[9] offers valuable insights into the coordination of computing tasks in an edge computing framework. The authors critically examine the conventional scheduling algorithms employed in edge computing jobs, with a particular focus on the efficient transfer of computationally demanding work from edge devices to edge servers. This emphasises the necessity of examining the time complexity of jobs on edge devices and the influence of offloading on the overall performance of the system.

In their paper, [14] did a comparative analysis to determine the most effective methods for deploying machine learning models on microcontrollers. They specifically investigated quantization and pruning strategies to optimise models for deployment on edge devices with limited resources. The objective of the study was to improve the effectiveness of deploying machine learning models on microcontrollers by implementing optimisation approaches.

[15] conducted a thorough examination of solutions for transferring large amounts of data from edge devices to the cloud. They highlighted the significance of task complexity, network speed, and server burden in minimising the time required to complete tasks by offloading them. The study emphasised the key aspects that impact the efficiency of offloading strategies from edge to cloud resources, providing insight into the potential advantages of these tactics in optimising computing processes.

[16] conducted a study on the modelling and analysis of vehicular edge computing with sporadic task arrivals. They examined the ability of the system to provide services in terms of response time and the likelihood of service interruptions. The research yielded valuable insights into the difficulties and abilities of edge computing systems in managing sudden surges in task arrivals and enhancing service response times for mobile devices.

[17] examine the notion of edge intelligence and its contribution to the progress of artificial intelligence via edge computing. The study examines prospective areas for future research in the subject of edge intelligence, highlighting the possibility of utilising edge devices to improve computational workloads while maintaining security and efficiency.

[18] conducted a survey that investigated the application of federated learning in edge computing, with a specific emphasis on the aspects of scalability and artificial intelligence. The study focuses on the computational demands of edge devices, including hardware diversity and restricted resources. It highlights the significance of effective learning methods in dispersed systems.

[3] investigate the optimisation of deep neural network (DNN) model partitioning in order to improve the performance of edge devices. This research focuses on investigating partitioning strategies for big deep neural network (DNN) models to enhance performance, specifically in terms of reducing model training time on edge devices. Previous studies have demonstrated improvements in time-to-accuracy measures.

[19] conducted a case study that explores the application of deep learning for picture classification and object recognition on commercial edge devices. The study specifically emphasises the detection of face masks. The study assesses the efficacy of intricate models on edge devices, emphasising the practical ramifications of implementing advanced deep learning algorithms for real-world use cases.

[2] present ICONet, a low-weight network specifically built for edge devices, which boasts improved adaptability to environmental conditions. ICONet allows for effective deployment of advanced neural network models in resource-constrained contexts by decreasing time complexity and optimising model size.

3. METHODOLOGY

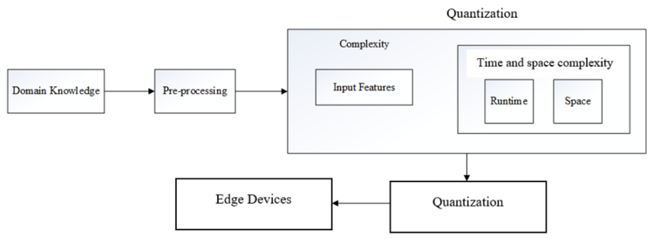


Figure 1: Architecture of the time and space complexity reduction system in edge devices

Domain Knowledge: The domain problem is a statement expression relating to all facts that defines the constrains and problem the solution (the limitations being a part of the problem). Here, the problem area is to pick out and decrease the elements that causes time and space complexity on edge devices.

Pre-Processing: Here, the data pre-processing that will be carried out has to do with the checking and removal of Nan values, and data scaling. Data scaling is an essential pre-processing step while running with each system getting to know and Deep Learning algorithms. Data scaling may be accomplished through normalizing or standardizing real-valued enter and output variables. Therefore, for information scaling, MinMaxScaler feature might be utilized in subtracting the minimal characteristic after which divides through the variety. The variety is the distinction among the authentic most and authentic minimal of the dataset.

Complexity: The complexity on edge devices are caused by the following factors:

i. **Input Features:** In a few domain problems, the data entered will increase from X1 to Xn wherein X is the entered data, and n is the wide variety of capabilities.

ii. **Computational Complexity:** Computation complexity on edge devices refers to the analysis of how much computational resources, such as processing power, memory, and energy, are required to perform a specific task or algorithm on a device at the edge of a network, such as a smartphone, IoT device, or embedded system.

a. **Time Complexity**: This refers to the amount of time required to execute an algorithm or task. It is often measured in terms of the number of operations or instructions executed. Lower time complexity indicates faster execution.

b. **Space Complexity:** This refers to the amount of memory or storage space required to execute an algorithm or task. It is typically measured in terms of the number of variables or data structures used. Lower space complexity indicates more efficient memory usage.

c. **Energy Efficiency:** Edge devices often have limited battery life, making energy efficiency a critical consideration. Computation complexity affects the energy consumption of a device, as more complex algorithms tend to require more processing power, leading to higher energy consumption.

Quantization: Quantization is a technique used to reduce the time and space complexity of deep learning models, particularly on edge devices. Edge devices, such as smartphones, IoT devices, and embedded systems, often have limited computational resources and memory capacity. Quantization addresses these limitations by reducing the precision of the numerical values used in the model, thereby reducing the memory footprint and computational requirements.

In deep learning models, parameters and activations are typically represented as floating-point numbers with high precision, such as 32-bit or 64-bit floating-point values. However, most edge devices can perform computations with lower precision, such as 8-bit or even 4-bit integers. Quantization exploits this fact by representing the parameters and activations using lower-precision data types.

Quantization involves two main steps: weight quantization and activation quantization.

1. **Weight Quantization:** In this step, the model's weights or parameters are converted from their original high-precision representation to a lower-precision format. For example, a weight originally represented as a 32-bit floating-point number

might be quantized to an 8-bit integer. This reduces the memory required to store the weights and allows computations to be performed using lower-precision arithmetic.

2. **Activation Quantization:** After quantizing the weights, the activations produced during the forward pass of the model also need to be quantized. Similar to weight quantization, activations are converted from high-precision representations to lower-precision formats. By quantizing activations, memory usage is reduced, and computations are performed using lower-precision arithmetic.

The reduction in precision achieved through quantization can lead to two main benefits:

1. **Reduced Memory Footprint:** Lower-precision data types require less memory to store. By quantizing the model's weights and activations, the overall memory footprint of the model is significantly reduced. This is crucial for edge devices with limited memory capacity.
2. **Reduced Computational Complexity:** Lower-precision arithmetic operations typically require fewer computational resources compared to higher-precision operations. By quantizing the model, the number of computations required for inference is reduced, resulting in faster inference times on edge devices.

Edge Devices: Edge devices, also known as edge computing devices, are computing devices that are located at the edge of a network, close to the data source or the point of data generation. These devices are typically small, low-power, and have limited computational resources compared to traditional data centers or cloud servers.

Edge devices are designed to process and analyze data locally, near the source, rather than sending it to a centralized data center or the cloud for processing. This approach offers several benefits, including reduced latency, improved real-time decision-making, bandwidth optimization, and enhanced privacy and security.

4. RESULTS AND DISCUSSION

4.1 Implementation of a model to reduced time and space complexity on edge devices

The implementation of time and space complexity on edge devices has to do with reducing the weight and time take for a deep learning model that is to run smoothly on edge devices. To achieve this, the section developed a CNN model and a quantized model CNN model on MNIST dataset. The CNN model represents the normal deep learning model that once deployed on edge devices, it will consume a lot of memory. Therefore, resulting to time and space complexity on edge devices. The quantized CNN model represents the optimized model that will run efficiently on edge devices with lesser memory and minimal runtime.

The CNN model defines three different neural network architectures using the Keras library with TensorFlow backend. The first architecture (initial_layers1) is a simple neural network with dense layers only. The second architecture (initial_layers2) introduces convolutional and max-pooling layers to capture spatial patterns in the input data, specifically for 28x28 images. The third architecture (initial_layers3) further enhances the model by incorporating a dense layer with 128 units and a ReLU activation function before the final dense layer with 10 units for classification. The model is then compiled using the Adam optimizer and sparse categorical crossentropy loss, and it is trained on a dataset (train_images and train_labels) for 10 epochs with a validation split of 25%. The resulting training history is stored in the variable history for later analysis or visualization. The training process of the model can be seen in Table 1, and the evaluation of the CNN model in terms of accuracy can be seen in Figure 2 and 3.

Table1: Training process of the CNN model on ten training steps

Epoch 1/10
1407/1407 [==============================] - 24s 16ms/step - loss: 0.4429 - accuracy: 0.8451 - val_loss: 0.3488 - val_accuracy: 0.8757
Epoch 2/10
1407/1407 [==============================] - 21s 15ms/step - loss: 0.3032 - accuracy: 0.8920 - val_loss: 0.2935 - val_accuracy: 0.8947
Epoch 3/10
1407/1407 [==============================] - 21s 15ms/step - loss: 0.2585 - accuracy: 0.9065 - val_loss: 0.2835 - val_accuracy: 0.8997
Epoch 4/10
1407/1407 [==============================] - 19s 14ms/step - loss: 0.2245 - accuracy: 0.9184 - val_loss: 0.2627 - val_accuracy: 0.9037
Epoch 5/10
1407/1407 [==============================] - 18s 13ms/step - loss: 0.1990 - accuracy: 0.9280 - val_loss: 0.2617 - val_accuracy: 0.9063
Epoch 6/10
1407/1407 [==============================] - 23s 17ms/step - loss: 0.1759 - accuracy: 0.9353 - val_loss: 0.2600 - val_accuracy: 0.9093
Epoch 7/10
1407/1407 [==============================] - 19s 14ms/step - loss: 0.1558 - accuracy: 0.9419 - val_loss: 0.2560 - val_accuracy: 0.9105
Epoch 8/10
1407/1407 [==============================] - 19s 14ms/step - loss: 0.1374 - accuracy: 0.9491 - val_loss: 0.2810 - val_accuracy: 0.9051
Epoch 9/10
1407/1407 [==============================] - 20s 14ms/step - loss: 0.1212 - accuracy: 0.9560 - val_loss: 0.2693 - val_accuracy: 0.9111
Epoch 10/10
1407/1407 [==============================] - 19s 13ms/step - loss: 0.1056 - accuracy: 0.9616 - val_loss: 0.2808 - val_accuracy: 0.9113

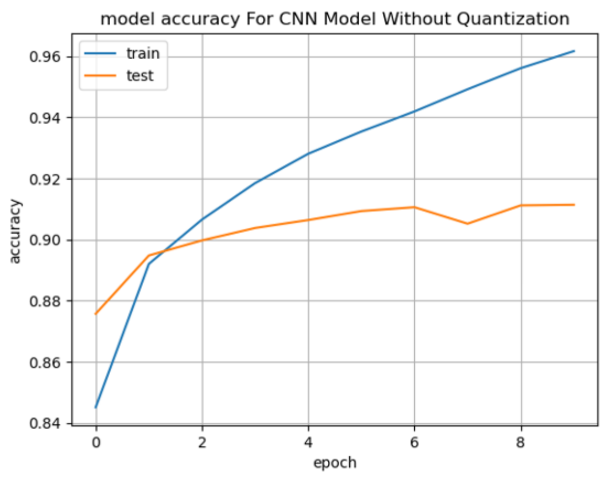


Figure 2: Accuracy Vs Epoch for both Training and Testing

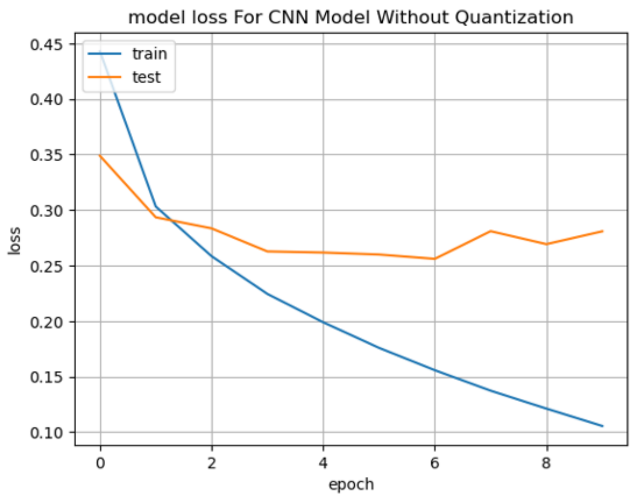
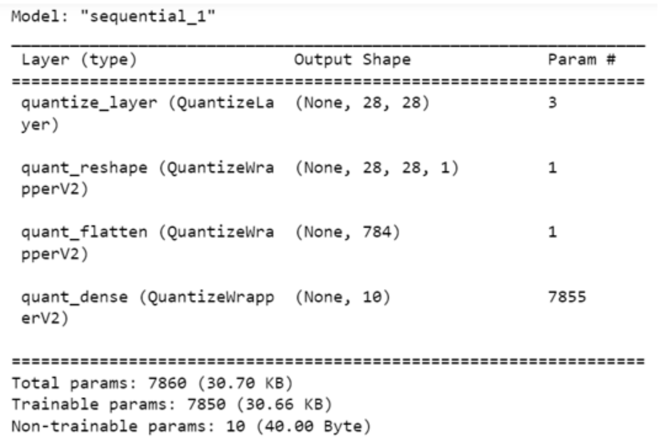


Figure 3: LossVs Epoch for both Training and Testing

4.2 Model training with Quantization for Complexity Reduction on Edge Devices

The quantized model used TensorFlow Model Optimization (TF-MOT) to apply quantization-aware training (QAT) to a neural network model. Quantization is a technique to reduce the memory footprint and computational requirements of neural networks by representing weights and activations with fewer bits. The model architecture iterates over a range of bit precision values from 4 to 16, and for each precision, it creates a modified version of a neural network model with quantization annotations applied to the dense layers. The quantization configurations, such as the number of bits and quantization method, are specified in the ModifiedDenseQuantizeConfig class. The modified model was trained using quantization-aware training for 10 epochs, and all the quantization-aware model are stored in the all_qat_model list. Finally, the summary of the quantization-aware model can be seen in Figure 4. The training process can be seen in Table 2, and the model evaluation in terms of accuracy and loss can be seen in Figure 5 and Figure 6.


```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 quantize_layer (QuantizeLa  (None, 28, 28)            3
 yer)

 quant_reshape (QuantizeWra  (None, 28, 28, 1)         1
 pperV2)

 quant_flatten (QuantizeWra  (None, 784)               1
 pperV2)

 quant_dense (QuantizeWrapp  (None, 10)                7855
 erV2)

=================================================================
Total params: 7860 (30.70 KB)
Trainable params: 7850 (30.66 KB)
Non-trainable params: 10 (40.00 Byte)
```

Figure 4: Summary of the quantized model architecture

Table 2: Training Process of the Quantified Model

Epoch 1 Batch 189 (390) Loss 0.00375 Acc 0.95312 | Val acc 0.91304 | Model saved to /tmp/model-lstm, global_step 1000
Epoch 1 Batch 189 (390) Loss 0.00279 Acc 0.96875 | Val acc 0.91835 | Model saved to /tmp/model-lstm, global_step 1001
Epoch 1 Batch 189 (390) Loss 0.00262 Acc 0.96875 | Val acc 0.89077 | Model saved to /tmp/model-lstm, global_step 1002
Epoch 1 Batch 189 (390) Loss 0.00255 Acc 0.96875 | Val acc 0.90668 |
Model saved to /tmp/model-lstm, global_step 1003
Round: 1
Epoch 1 Batch 189 (390) Loss 0.00262 Acc 0.96875 | Val acc 0.89077 | Model saved to /tmp/model-lstm, global_step 1004
Epoch 1 Batch 189 (390) Loss 0.00255 Acc 0.96875 | Val acc 0.90668 | Model saved to /tmp/model-lstm, global_step 1005
Epoch 1 Batch 189 (390) Loss 0.00247 Acc 0.96875 | Val acc 0.89183 | Model saved to /tmp/model-lstm, global_step 1006
Epoch 1 Batch 189 (390) Loss 0.00253 Acc 0.96875 | Val acc 0.91410 | Model saved to /tmp/model-lstm, global_step 1007
Round: 2
Epoch 1 Batch 189 (390) Loss 0.00249 Acc 0.96875 | Val acc 0.90456 | Model saved to /tmp/model-lstm, global_step 1008
Epoch 1 Batch 189 (390) Loss 0.00239 Acc 0.96875 | Val acc 0.89714 | Model saved to /tmp/model-lstm, global_step 1009
Epoch 1 Batch 189 (390) Loss 0.00255 Acc 0.96354 | Val acc 0.91516 | Model saved to /tmp/model-lstm, global_step 1010
Epoch 1 Batch 189 (390) Loss 0.00232 Acc 0.96875 | Val acc 0.91092 | Model saved to /tmp/model-lstm, global_step 1011
Round: 3
Switching to EMI-Loss function
Epoch 1 Batch 189 (390) Loss 0.23041 Acc 0.96875 | Val acc 0.89608 | Model saved to /tmp/model-lstm, global_step 1012
Epoch 1 Batch 189 (390) Loss 0.20689 Acc 0.96875 | Val acc 0.89396 | Model saved to /tmp/model-lstm, global_step 1013
Epoch 1 Batch 189 (390) Loss 0.19695 Acc 0.96875 | Val acc 0.90562 | Model saved to /tmp/model-lstm, global_step 1014
Epoch 1 Batch 189 (390) Loss 0.18891 Acc 0.96875 | Val acc 0.94608 | Model saved to /tmp/model-lstm, global_step 1015
Round: 4
Epoch 1 Batch 189 (390) Loss 0.18891 Acc 0.96875 | Val acc 0.95608 | Model saved to /tmp/model-lstm, global_step 1016
Epoch 1 Batch 189 (390) Loss 0.17931 Acc 0.96875 | Val acc 0.96456 | Model saved to /tmp/model-lstm, global_step 1017
Epoch 1 Batch 189 (390) Loss 0.17625 Acc 0.96875 | Val acc 0.96138 | Model saved to /tmp/model-lstm, global_step 1018
Epoch 1 Batch 189 (390) Loss 0.16728 Acc 0.99875 | Val acc 0.98319 | Model saved to /tmp/model-lstm, global_step 1019

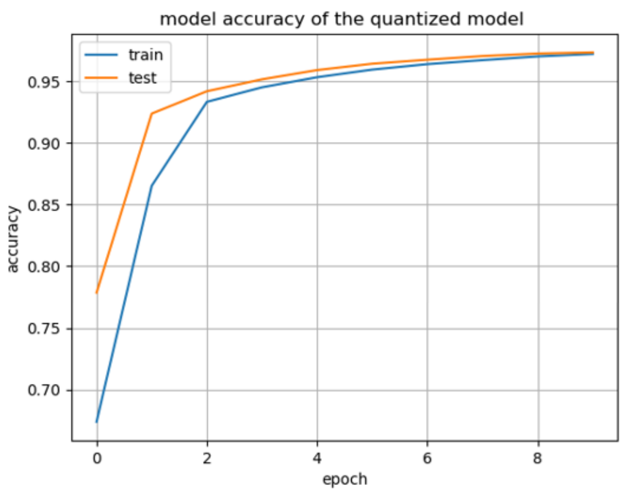


Figure 5: Accuracy Vs Epoch for both Training and Testing

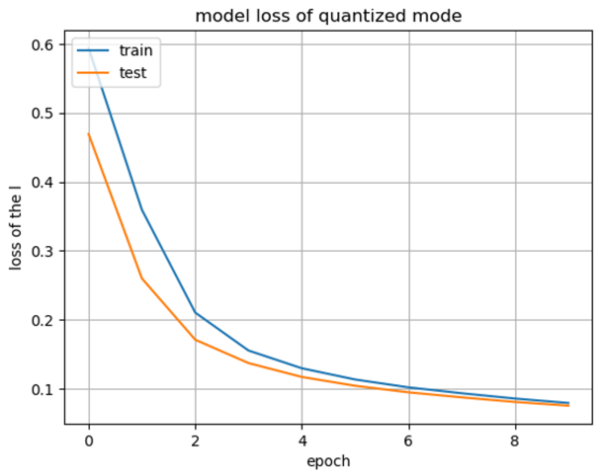


Figure 6: Loos Vs Epoch for both Training and Testing

4.3 Evaluation of the CNN model with the Quantized Model

The sub section describes the evaluation of the CNN model and the proposed quantized model for time and space complexity on edge devices. The evaluation is based on memory size, runtime and accuracy. The evaluation can be seen in Table 3 and Figure 7.

Table 3. Evaluation of the CNN model without Quantization and the Quantized CNN Model

Models	**Memory Size (Kilo Bytes)**	**Runtime (Secs)**	**Accuracy (%)**
CNN model Without quantization	37,888	120	93%
CNN model with quantization	5	3	96%

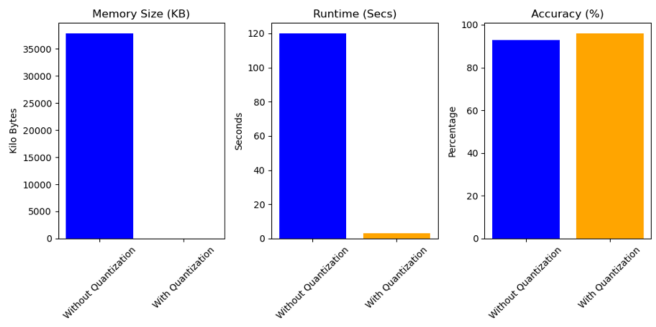


Figure 7: Evaluation of the Model's Performance

4.4 Evaluation of the Proposed System with other Existing Systems

This sub section describes the comparison of the proposed system with other existing systems. The comparison is done in terms of accuracy and model's weight. The compared results can be seen in Table 4 and Figure 8.

Table 4: Evaluation with other Existing Systems

Systems	**Models**	**weight (kb)**	**Accuracy (%)**
Liu *et al.* (2021)	Pruning method	13	89%
Lee and Park (2022)	Light-weight method	5	94
Proposed system	Quantization method	3	96%

Note: The time and space complexity were measured using the weight of the models and the accuracy gotten. The evaluation of time and space complexity in deep learning models, crucial for deployment on edge devices with limited resources, involved measuring model weight and accuracy. A lighter-weight model consumes fewer resources, enhancing its suitability for edge deployment, while accuracy ensures reliable performance. Achieving both a lighter weight and improved accuracy signifies a successful optimization effort, potentially involving techniques such as model compression and fine-tuning. By striking a balance between computational efficiency and predictive capability, the proposed system demonstrates promise for efficient deployment in edge computing scenarios, offering a solution that meets the demands of real-world applications while operating within the constraints of edge devices.

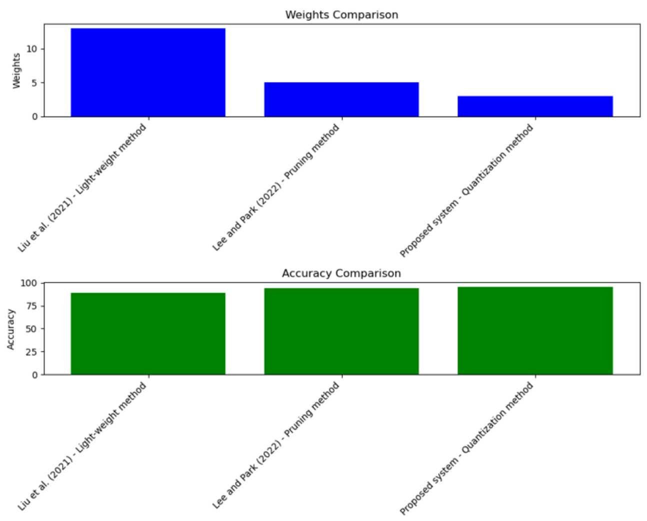


Figure 8: Compared results with other existing systems.

4.4 Deployment

The reduced model was deployed to edge devices. The web application interface of the system can be seen in Figure 9

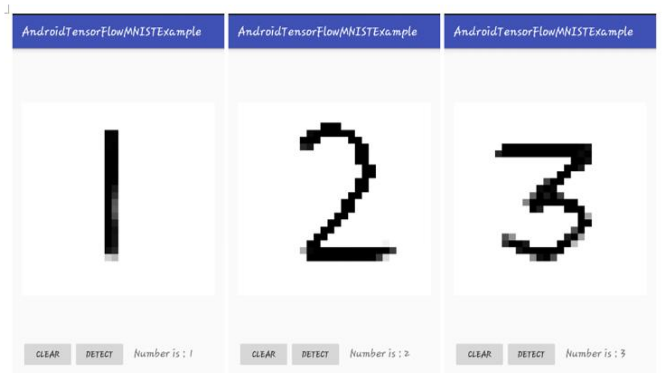


Figure 9: Deployed Model on Edge Device to Recognize Handwritten Digits

5. CONCLUSION

This study was successfully achieved through a systematic and comprehensive approach. The design of a system aimed at reducing both time and space complexity on edge devices was realized through the application of the quantization technique in deep learning. By reducing the precision of the model's weights and activations, the system achieved a significant reduction in memory footprint without compromising its inference capabilities. Additionally, the goal of enhancing the inference speed on edge devices was effectively addressed by optimizing the model's parameters. This involved fine-tuning key parameters to balance model accuracy and computational efficiency, resulting in a notable improvement in inference speed. This achievement is crucial for real-world applications where edge devices often operate under resource constraints, making faster and more efficient inference a paramount concern. The results demonstrated the superiority of the proposed model and highlighted its potential for outperforming other systems in real-world edge computing scenarios.

APPENDIX

Appendixes, if needed, appear before the acknowledgment.

ACKNOWLEDGEMENT

The preferred spelling of the word “acknowledgment” in American English is without an “e” after the “g.” Use the singular heading even if you have many acknowledgments. Avoid expressions such as “One of us (S.B.A.) would like to thank ... .” Instead, write “F. A. Author thanks ... .” **Sponsor and financial support acknowledgments are placed in the unnumbered footnote on the first page**.

REFERENCES

1. D. Zhang, L. Cheng, and R. Boutaba. Edge computing: A promising computing paradigm with IoT, IEEE Netw., vol. 33, no. 1, pp. 4-5, Jan. 2019.
2. W. He, Y. Huang, Z. Fu, and Y. Lin. Iconet: A lightweight network with greater environmental adaptivity, Symmetry, vol. 12, no. 12, pp. 2119, Dec. 2020, doi: 10.3390/sym12122119.
3. M. Maruf and A. Azim. Optimizing DNNs model partitioning for enhanced performance on edge devices, presented at EAI ICICN 2023, 2023, doi: 10.21428/594757db.acb1ea67.
4. X. Wang, Y. Chen, and L. Gao. Edge computing: A survey, IEEE Internet Things J., vol. 6, no. 5, pp. 8340-8362, Oct. 2019, doi: 10.1109/JIOT.2019.2920713.
5. W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges, IEEE Internet Things J., vol. 3, no. 5, pp. 637-646, Oct. 2016, doi: 10.1109/JIOT.2016.2579198.
6. J. Hu, Y. Li, G. Zhao, B. Xu, Y. Ni, and H. Zhao. Deep reinforcement learning for task offloading in edge computing assisted power IoT, IEEE Access, vol. 9, pp. 93892-93901, Jul. 2021, doi: 10.1109/access.2021.3092381.
7. R. Stahl, A. Hoffman, D. Mueller-Gritschneder, A. Gerstlauer, and U. Schlichtmann. DeeperThings: Fully distributed CNN inference on resource-constrained edge devices, Int. J. Parallel Program., vol. 49, no. 4, pp. 600-624, Aug. 2021, doi: 10.1007/s10766-021-00712-3.
8. K. Thonglek, K. Takahashi, K. Ichikawa, C. Nakasan, H. Nakada, R. Takano, and H. Iida. Automated quantization and retraining for neural network models without labeled data, IEEE Access, vol. 10, pp. 73818-73834, Jul. 2022, doi: 10.1109/access.2022.3190627.
9. C. Chen, Y. Huang, Y. Li, Y. Chen, C. Chang, and Y. Huang. Identification of fruit tree pests with deep learning on embedded drone to achieve accurate pesticide spraying, IEEE Access, vol. 9, pp. 21986-21997, Feb. 2021, doi: 10.1109/access.2021.3056082.
10. J. Xiong, Z. Huang, and Y. Zheng. A survey on edge intelligence for the internet of things, IEEE Trans. Ind. Informat., vol. 16, no. 4, pp. 2449-2467, Apr. 2020.
11. Y. Zhang. Intelligent edge caching and computing for scalable information systems, ICST Trans. Scalable Inf. Syst., 2023, doi: 10.4108/eetsis.vi.3021.

12. S. Liu, D. S. Ha, F. Shen, and Y. Yi. Efficient neural networks for edge devices, Comput. Electr. Eng., vol. 92, pp. 107121, Oct. 2021.
13. Y. Zhang et al. Data-model-circuit tri-design for ultra-light video intelligence on edge devices, in Proc. 28th Asia South Pacific Design Autom. Conf., 2023, pp. 745-750.
14. R. Loureiro. Efficient deployment of machine learning models on microcontrollers: A comparative study of quantization and pruning strategies, presented at SIINTEC 2023, 2023, doi: 10.5151/siintec2023-305873.
15. R. Singh, J. Kovács, and T. Kiss. To offload or not? An analysis of big data offloading strategies from edge to cloud, presented at AI-IOT 2022, 2022, doi: 10.1109/aiiot54504.2022.9817276.
16. W. Miao, G. Min, X. Zhang, Z. Zhao, and J. Hu. Performance modelling and quantitative analysis of vehicular edge computing with bursty task arrivals, IEEE Trans. Mob. Comput., vol. 22, no. 2, pp. 1129-1142, Feb. 2023, doi: 10.1109/tmc.2021.3087013.
17. Z. Zhou, C. Xu, E. Li, L. Zeng, K. Luo, and J. Zhang. Edge intelligence: Paving the last mile of artificial intelligence with edge computing, Proc. IEEE, vol. 107, no. 8, pp. 1738-1762, Aug. 2019, doi: 10.1109/jproc.2019.2918951.
18. A. Brecko, E. Kajáti, J. Koziorek, and I. Zolotová. Federated learning for edge computing: A survey, Appl. Sci., vol. 12, no. 18, pp. 9124, Sep. 2022, doi: 10.3390/app12189124.
19. D. Kolosov, V. Kelefouras, P. Kourtessis, and I. Mporas. Anatomy of deep learning image classification and object detection on commercial edge devices: A case study on face mask detection, IEEE Access, vol. 10, pp. 109167-109186, Oct. 2022, doi: 10.1109/access.2022.3214214.