ISSN 2707-1898 (print)
Український журнал інформаційних технологій
Ukrainian Journal of Information Technology
http://science.lpnu.ua/uk/ujit
https://doi.org/10.23939/ujit2024.02.074
Article received 15.10.2024 p.
Article accepted 19.11.2024 p.
UDC 004.93

Correspondence author
I. Y. Kazymyra
iryna.y.kazymyra@lpnu.ua

V. Y. Chornenkyi, I. Y. Kazymyra
Lviv Polytechnic National University, Lviv, Ukraine

ML MODELS AND OPTIMIZATION STRATEGIES FOR ENHANCING THE PERFORMANCE OF CLASSIFICATION ON MOBILE DEVICES

The paper highlights the increasing importance of machine learning (ML) in mobile applications, with mobile devices becoming ubiquitous due to their accessibility and functionality. Various ML models, including Convolutional Neural Networks (CNNs) and Vision Transformers (ViTs), are explored for their applications in real-time classification on mobile devices. The paper identifies key challenges in deploying these models, such as limited computational resources, battery consumption, and the need for real-time performance.

Central to the research is the comparison of MobileNetV2, a lightweight CNN designed for mobile applications, and Vision Transformers (ViTs), which have shown success in image recognition tasks. MobileNetV2, with its depthwise separable convolutions and residual connections, is optimized for resource efficiency, while ViTs apply self-attention mechanisms to achieve competitive performance in image classification. The study evaluates the performance of both models before and after applying optimization techniques like quantization and graph optimization.

It was discovered that quantization is one of the most effective optimization strategies for mobile environments, reducing model size by up to 74 % and improving inference speed by 44 % in ViTs. Additionally, graph optimization techniques, such as operator fusion, pruning, and node reordering, are examined for their role in reducing computational complexity and improving performance on resource-constrained devices.

Experimental results on different datasets, including MNIST and the ASL Alphabet dataset, demonstrate the significant performance improvements achieved through optimization. The study shows that post-training quantization and graph optimization can reduce model size, inference time, and CPU usage, making ML models more suitable for mobile applications. The experiments were conducted on a Xiaomi Redmi Note 8 Pro device, showcasing the practical benefits of these optimizations in real-world mobile deployments.

The research concludes that optimization techniques like quantization and graph optimization are essential for deploying ML models on mobile devices, where resource constraints and real-time performance are critical. It also provides valuable insights into how ML architectures can be optimized for mobile environments, contributing to the advancement of efficient AI-driven mobile applications.

Keywords: deep learning, convolutional neural networks, vision transformers, mobile applications.

Introduction / Вступ

Mobile devices, encompassing smartphones and tablets, have become ubiquitous in modern life. According to the International Telecommunication Union (ITU), there were over 5.3 billion unique mobile phone users globally by the end of 2022, reflecting an unprecedented level of connectivity and engagement with mobile technology [1]. The widespread adoption of mobile devices is driven by their affordability, accessibility, and the extensive range of functionalities they offer. Mobile devices are now integral to everyday activities, from communication and entertainment to productivity and health management [2].

Machine learning (ML) has increasingly been integrated into mobile applications, enabling sophisticated features and services directly on users' devices. The proliferation of on-device ML is largely attributed to advancements in mobile hardware and software frameworks designed to support such technologies. For instance, Google's TensorFlow Lite and Apple's Core ML provide specialized tools for deploying ML models efficiently on mobile platforms.

The application of ML on mobile devices ranges from real–time image and speech recognition to predictive text input and personalized recommendations. Research indicates that on-device ML can significantly enhance user experience by providing faster responses and preserving privacy through local data processing [3]. As of 2023, an increasing number of mobile applications are leveraging ML for functionalities like augmented reality (AR), health monitoring, and autonomous navigation [4].

This paper explores the influence of machine learning model characteristics – specifically Convolutional Neural Networks (CNNs) and Vision Transformers (ViTs) – on performance in mobile environments. It also examines the impact of optimization techniques, such as quantization, on these models. By understanding these factors, we aim to provide insights into how different ML architectures and optimizations can be effectively utilized for mobile applications.

The object of research – the overall performance characteristics of machine learning models on mobile devices.

The subject of research – the impact of model architecture, optimization techniques, and resource constraints on real-time applications performance, the specific application of Vision Transformers (ViT) and MobileNetV2 for hand gesture recognition, examining how these models work before and after optimization techniques like quantization and graph optimization.

The purpose of the research – is to investigate and compare the efficiency of applying Vision Transformers (ViT) and MobileNetV2 architectures, before and after optimization, in real-time recognition tasks, specifically focusing on their inference speed, memory usage, and accuracy on mobile devices.

To achieve this purpose, the following *main research objectives* are identified:

- analyze current approaches to gesture recognition using Vision Transformers (ViT) and MobileNetV2;
- evaluate the accuracy, loss, and model size for both ViT and MobileNetV2 architectures before and after quantization and graph optimization;
- investigate the performance of each model architecture on datasets, such as MNIST for ViT and ASL Alphabet for MobileNetV2, with a focus on mobile deployment;
- examine the impact of quantization and graph optimization on the models' inference time, memory usage, and CPU consumption, as well as their real-time performance on mobile devices.

Analysis of recent research and publications. Models and architectures. Transformers, originally designed for natural language processing (NLP) tasks, have revolutionnized the field of machine learning. Their ability to model long-range dependencies using self-attention mechanisms has made them the backbone of many state-of-the-art NLP models. Recently, transformers have been adapted for computer vision tasks, leading to the development of Vision Transformers (ViTs) [5]. ViTs treat images as sequences of patches, akin to word tokens in text, and apply self-attention to capture global image features.

Vision Transformers (ViTs) have demonstrated significant advancements in image recognition tasks. The Vision Transformer achieves convolutional neural network (CNN)-level accuracy when trained with large-scale datasets such as JFT-300M [6]. By treating images as sequences of patches and utilizing self-attention [5] to model extensive interactions among image tokens, ViTs offer advantages such as global processing capabilities and excellent scalability with increasing dataset sizes. However, when it comes to lightweight and mobile networks, models like MobileNets [7], [8] and EfficientNets [9] continue to outperform ViTs. For example, under a parameter budget of 6 million, DeiT [10] is 3 % less accurate than MobileNetV3 [8] on the ImageNet classification task and consumes six times more FLOPs.

Lightweight CNNs have been instrumental in enabling numerous mobile vision tasks. Compared to CNNs, ViTs are generally more complex, with larger parameter counts (e.g., ViT-B/16 vs. MobileNetV3: 86 vs. 7.5 million parameters) [11]. ViTs also require significant data augmentation and regularization to prevent overfitting [10], and they need high-cost decoders for downstream tasks involving dense predictions. For instance, a ViT-based segmentation network [12] learns around 345 million parameters and achieves performance comparable to the CNN-based DeepLabv3 network [13], which only has 59 million parameters. The increased parameter count in ViT-based models is mainly due to the absence of image-specific inductive biases in CNNs.

Hybrid models that combine convolutions and transformers are gaining attention to address these challenges. Several designs have emerged to integrate the strengths of both architectures, such as ViT-C [11], CvT [14], BoTNet [15], and ConViT [16]. These models can achieve competitive performance but generally carry a heavier computational load than efficient CNNs like EfficientNet [9]. They are often outperformed by lightweight CNNs, such as MobileNetV2, when scaled down. These models also remain resource-intensive and sensitive to data augmentation.

Optimization techniques. Significant work has been done to improve the efficiency of transformers. Most approaches focus on reducing the number of tokens in the transformer block using various methods, including down-sampling [17] and pyramidal structures [18]. The proposed separable self-attention module is a drop-in replacement for multi-head attention (MHA) and can be easily integrated into any transformer-based model to improve efficiency further.

ViT and its variants have produced state-of-the-art results in image processing domains and have been the foundation for many pre-trained models, such as DINO [19] and MOCOv3 [20]. However, much research has focused on performance improvement, with relatively little attention to efficiency [10], [21]. Since multi-head self-attention (MSA) has quadratic complexity with respect to input length, transformer models like ViT can be computationally intensive. Consequently, classical transformers are not widely used in mobile and other low-resource environments.

Recently, a lightweight and general-purpose vision transformer called MobileViT has been introduced [22]. MobileViT addresses the limitations of traditional transformers by combining input-adaptive weighting and global processing with spatial inductive biases learned by CNN-based networks. MobileViT integrates inverted residual blocks from MobileNetV2 with the ViT encoder block, encoding both local and global information with fewer parameters. Despite these improvements, MobileViT remains slower than lightweight CNNs [7] in terms of both inference and training time due to the lack of dedicated device-level operations support for MSA layers. The entire ***Mv2*** block can be expressed by formula (1):

$$Mv2_{stride\ \{1|2\}}(x) \quad \begin{cases} PConv\left(DConv_{stride\ 1}\left(PConv(x)\right)\right) + x \\ PConv\left(DConv_{stride\ 2}\left(PConv(x)\right)\right) \end{cases}, \quad (1)$$

where: $Mv2_{stride} = \{112\}$ represents the MobileNetV2 block for different stride values; *PConv* refers to the pointwise convolution, which applies a 1x1 convolution to the input to adjust the depth of the feature maps; $DConv_{stride}$ denotes the depthwise convolution, which applies a spatial convolution over each input channel separately.

Besides architecture optimizations, additional steps like quantization and graph optimization techniques can be applied to achieve further improvements. Quantization can be

applied both during model creation and post-training to get more improvements based on the expected data set. Post-training quantization includes general techniques to reduce CPU and hardware accelerator latency, processing, power, and model size with little degradation in model accuracy. These techniques can be performed on an already-trained float TensorFlow model and applied during TensorFlow Lite conversion.

There are a lot of different post-training quantization techniques which can include dynamic range quantization, full integer quantization, and float16 quantization. Based on recent studies [23], integer quantization is the most aggressive and gives the most results in a smaller model and increased inferencing speed, which is valuable for low-power devices such as microcontrollers, edge, and mobile devices.

Important performance metrics for mobile and edge devices:

1. Limited Computational Resources – mobile devices are constrained by significantly lower computational power than desktop or server environments. Advanced gesture recognition algorithms, particularly those employing deep learning models such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), and Vision Transformers (ViTs), demand substantial computational resources. Optimizing these complex models to run efficiently on resource-limited devices remains a formidable challenge.
2. Battery Consumption – the high computational load associated with gesture recognition algorithms can lead to increased power consumption, thereby quickly depleting the battery life of mobile devices. Ensuring the energy-efficient operation of recognition systems is critical to maintaining user satisfaction and device usability.
3. Real-Time Performance – gesture recognition systems must provide responses in real-time to ensure a seamless user experience. Achieving low-latency performance is challenging due to the computational demands of the algorithms and potential delays introduced by network dependencies, particularly in scenarios where computations are offloaded to cloud services.

CNN architecture. For experiment purposes, we have used models with 2 different architectures and 3 datasets. For the CNN architecture representative, MobileNet2 was used. MobileNetV2 is a lightweight deep learning model developed by Google, specifically designed for mobile and edge devices. It employs depthwise separable convolutions, which break down standard convolution operations into simpler components, significantly reducing the number of parameters and computational complexity. The architecture features linear bottlenecks, where the last layer of each block is a linear layer instead of a non-linear activation function, enhancing information preservation and gradient flow during training. Additionally, MobileNetV2 incorporates residual connections to facilitate the learning of identity mappings, helping to mitigate the vanishing gradient problem and improve feature extraction [7].

ViT architecture. For the ViT architecture, a default structure was used. The core of the architecture consists of multiple transformer encoder layers, each featuring a self-attention mechanism that allows the model to assess the importance of different patches relative to one another. Following the self-attention layer, the output is processed through a feed-forward neural network, with normalization and residual connections enhancing training stability and performance. A special classification token is included in the sequence of patch embeddings, aggregating information for the final classification task. The output corresponding to this classification token is then passed through a linear layer to produce class probabilities. Overall, ViT represents a significant shift from traditional convolutional neural networks (CNNs) by treating images as sequences of patches, leveraging self-attention mechanisms to achieve competitive performance across various image classification benchmarks while offering a flexible and scalable architecture [5].

Post–Training Quantization Scheme. Models were also passed through post-training optimizations. Full integer quantization is an optimization strategy that converts 32-bit floating-point numbers (such as weights and activation outputs) to the nearest 8-bit fixed-point numbers using the following formula (2).

$$real_{value} = (int8_{value} + zero_{point}) \times scale, \qquad (2)$$

where: $real_{value}$ is the original floating-point value that the integer value represents after quantization; $int8_{value}$ is the quantized 8-bit integer value; $zero_{value}$ defines shift for the range of the quantized values to ensure that 0 in floating-point maps correctly to an integer value; $scale$ is a scaling factor that maps the integer value back to its original floating-point value.

After applying the full integer quantization formula, the model undergoes optimizations that improve its efficiency while preserving as much accuracy as possible. During full integer quantization, the original 32-bit floating-point weights and activation outputs are approximated by 8-bit integer values. This conversion reduces memory usage and computational demands, enabling faster inference and lower power consumption, particularly on devices with limited hardware resources like mobile phones or edge devices. The zero point ensures that the floating-point value of 0 is accurately represented within the integer range, which can include both positive and negative numbers depending on the model. This is critical for maintaining numerical stability and accuracy after quantization. The zero point allows the integer values to shift in such a way that floating-point operations like multiplication and addition still produce meaningful results when mapped back to their floating-point equivalents. After converting the floating-point values to integers, matrix multiplications are performed in the integer domain, which is much more efficient on certain hardware platforms. Once these operations are completed, the scale factor is used to revert the results back to their approximate floating-point equivalents for subsequent layers or final outputs.

Graph Optimization in Deep Learning. Graph optimization in deep learning refers to a suite of techniques used to improve the computational efficiency and performance of a model by restructuring and optimizing the computational graph. A computational graph is a directed acyclic graph (DAG) where nodes represent operations (e. g., matrix multiplications, convolutions) and edges represent the flow of data (tensors) between these operations. Each edge carries the result of one operation to be used as input for the

next. Optimizing this graph is crucial for reducing the computational complexity, memory usage, and execution time, which becomes especially important when deploying deep learning models on resource-constrained devices like mobile phones, embedded systems, or other edge devices.

At its core, graph optimization seeks to minimize the number of operations, eliminate redundancies, and maximize the parallelism of computations, all while maintaining the accuracy and performance of the model. Consider a deep learning model as a sequence of mathematical operations, where each operation depends on the results of previous ones. Graph optimization identifies opportunities to fuse, prune, or reorder these operations, thereby improving efficiency.

For example, the time complexity of a model is often determined by the number of operations in its computational graph, denoted by $T(n)$, where n represents the size of the input or number of layers. Reducing the number of redundant nodes (pruning) or combining operations (fusion) can decrease $T(n)$, improving the overall execution time. These optimizations directly affect the model's inference time, making it more suitable for real-time applications on mobile devices.

Mathematically, we can represent the impact of graph optimization techniques on performance by formula (3):

$$T_{optimized}(n) = \alpha \cdot T(n) + \beta \cdot M(n), \quad (3)$$

where: $T(n)$ – is the original time complexity of the model; $M(n)$ – is the memory complexity; α and β are the constants representing the optimization factors for computation and memory, respectively.

Through graph optimization, the constants α/alphaα and β/betaβ are reduced, leading to improved performance in both time and memory usage.

Techniques for Graph Optimization. One of the most impactful techniques for graph optimization is operator fusion. Operator fusion involves the combination of sequential operations into a single, more efficient one, reducing the overhead associated with memory accesses and improving the throughput of computation. Consider a scenario where a convolutional layer is followed by batch normalization and an activation function. Instead of executing these three operations in separate steps, operator fusion merges them into a single computational kernel. This not only reduces the number of memory accesses but also makes better use of hardware resources by executing the operations in parallel.

The benefits of operator fusion can be expressed as a reduction in latency (4):

$$L_{fused} = \frac{L_{sequential}}{k}, \quad (4)$$

where: $L_{sequential}$ – is the latency of executing operations sequentially; k – represents the number of operations fused into a single kernel.

By reducing k, the total execution time decreases, which is particularly important for mobile devices where low latency is crucial.

Another key technique is graph pruning, which involves removing redundant or less significant nodes and edges from the computational graph. This method simplifies the model and reduces the number of computations required for inference. Pruning can be performed in multiple ways: by removing individual weights, entire neurons, or even whole layers, depending on their contribution to the model's performance. For instance, if certain weights contribute minimally to the final output, they can be set to zero without significantly impacting model accuracy. This leads to a sparse representation of the model, which is more efficient in terms of both computation and memory usage.

The impact of pruning on the size of the model can be described by the following formula (5):

$$S_{pruned} = S_{original} - \sum_i W_i, \quad (5)$$

where: $S_{original}$ is the size of the original model; W_i are the pruned weights.

This equation highlights the reduction in model size after pruning, which translates into faster inference times and lower memory consumption, both critical for mobile applications.

Finally, node reordering plays a pivotal role in optimizing the execution order of operations within the computational graph. By reordering operations, we aim to minimize data dependencies and maximize parallelism, ensuring that independent computations are performed simultaneously. This optimization allows the hardware (whether it be a CPU, GPU, or neural processing unit) to process data more efficiently, reducing execution stalls and improving cache utilization.

The benefit of node reordering can be modelled as an increase in parallelism cap with the following relationship (6):

$$P_{reordered} = \frac{P_{original}}{d}, \quad (6)$$

where: $P_{reordered}$ represents the new level of parallelism after node reordering; $P_{original}$ refers to the initial level of parallelism before node reordering; d is the degree of data dependency between operations.

By reducing d we increase the potential for parallel execution, thus improving the overall efficiency of the model.

Research results and their discussion / Результати досліджень та їх обговорення

For our practical implementation, we focused on optimizing a Vision Transformer (ViT) model using the MNIST dataset. The MNIST dataset is a widely used benchmark in the machine-learning community and consists of grayscale images of handwritten digits (0–9). Each image is 28×28 pixels in size, represented by grayscale values ranging from 0 to 255. The training set contains 60 000 images, while the test set includes 10,000 images. Given the relatively simple structure of the MNIST dataset, it provides a good foundation for testing and optimizing our model for tasks such as digit recognition.

We trained the initial ViT model on the MNIST dataset, which contained 870 937 parameters, and we monitored its performance (Fig. 1) regarding training and validation accuracy and loss. The model converged quickly, achieving near-perfect accuracy on both the training and validation sets, with losses dropping significantly in the early epochs. This indicated that the model was highly effective at learning the representations of handwritten digits.

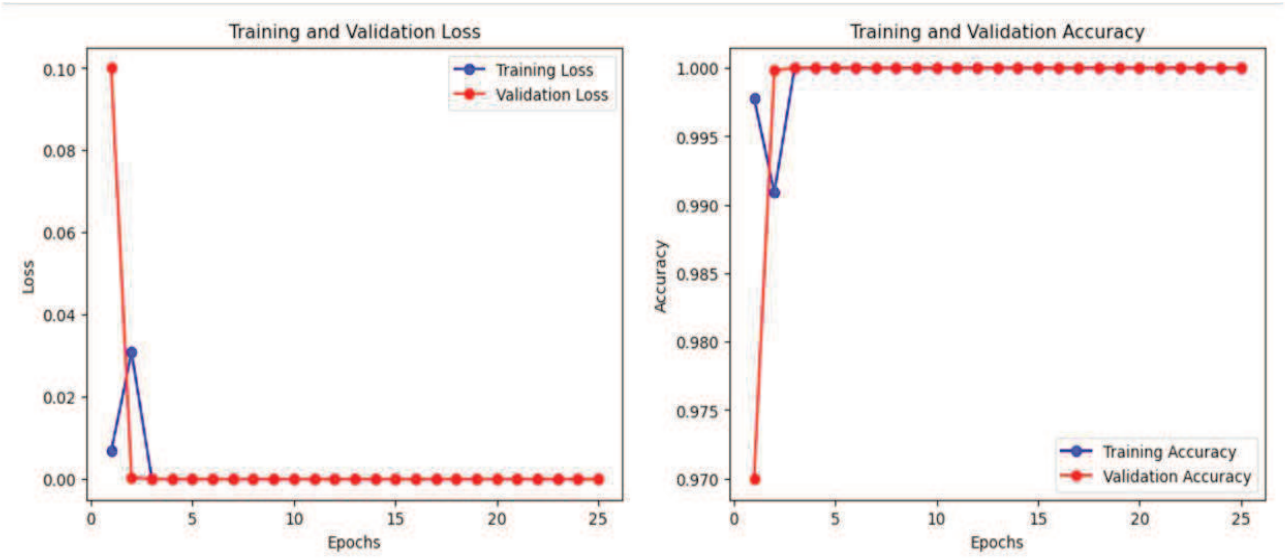

Fig. 1. Training and validation loss (left) and accuracy (right) over 25 epochs for the ViT model on the MNIST dataset / Втрати під час навчання та перевірки (ліворуч) і точності (праворуч) протягом 25 епох для моделі ViT на наборі даних MNIST

Model Conversion to TensorFlow Lite (TFLite). Once the ViT model was trained, we converted it into TensorFlow Lite (TFLite) format. This conversion is essential for deploying deep learning models on mobile and embedded devices, where computational resources are often limited. Testing the TFLite model on a real device allowed us to evaluate its performance in terms of inference speed, memory consumption, and CPU utilization.

While the model performed well in its initial form, it was still relatively large in size (3.5 MB) and could benefit from further optimization to improve its deployment feasibility on mobile devices.

Quantization for Model Size Reduction and Speed Optimization. To enhance the efficiency of the ViT model, we applied quantization. Quantization reduces the precision of the model's weights from 32-bit floating-point values to 8-bit integers, resulting in a significantly smaller model size and faster inference times. After quantization, the model size dropped from 3.5 MB to just 899 KB, making it much more suitable for mobile devices with limited storage.

Although quantization often results in a slight increase in loss, the impact on accuracy was minimal.

The quantized ViT model retained near-perfect accuracy (Fig. 2), with only a small increase in loss (from 3.93 e-08 to 2.49 e-06), which is negligible given the substantial gains in efficiency (Fig. 3). Moreover, the inference time improved dramatically. The inference time decreased from 16 milliseconds in the original model to 10 milliseconds in the quantized version for the 95th percentile, showcasing the benefits of quantization for speed improvements.

Graph Optimization for Further Efficiency. In our model, we employed a combination of these advanced graph optimization techniques to enhance the model's performance, particularly for deployment on mobile devices. To maximize the benefits of these optimizations, we used TensorFlow Lite's operator fusion, graph pruning, and node reordering alongside mixed-precision quantization based on the nuclear norm of the attention maps and output features.

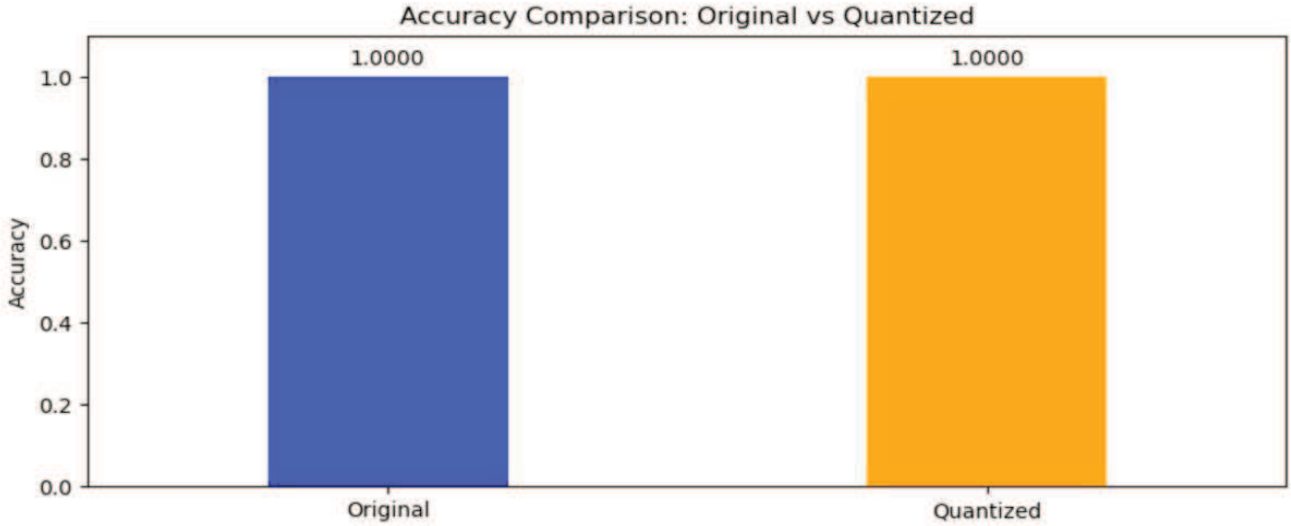

Fig. 2. Accuracy comparison between the original and quantized ViT models / Порівняння точності вихідної та квантизованої моделей ViT

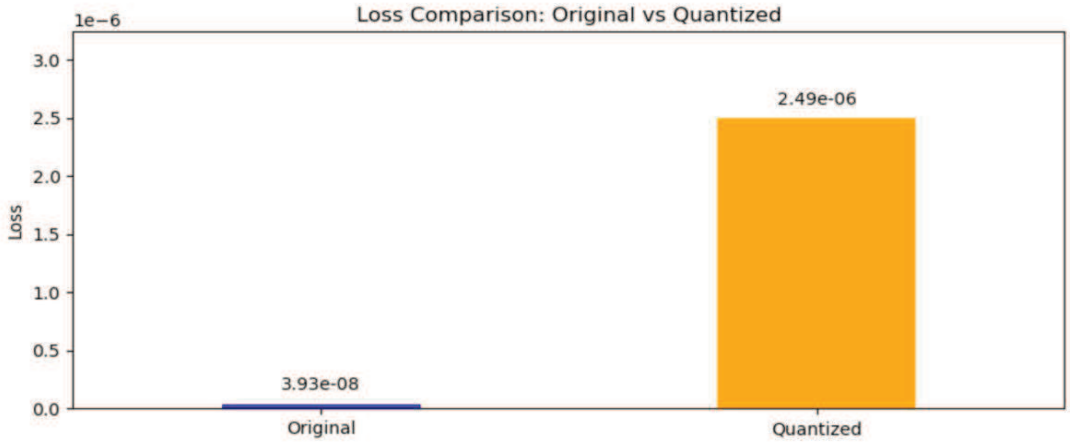

Fig. 3. Loss comparison between the original and quantized ViT models / Порівняння втрат між початковою та квантизованою моделями ViT

Among the optimization techniques, we found operator fusion to be particularly effective in reducing the latency of our model. By fusing operations such as convolution, batch normalization, and activation into a single kernel, we were able to significantly lower the number of memory accesses and improve throughput. This led to an impressive reduction in both execution time and memory footprint, making the model more suitable for mobile deployment.

These optimizations should provide significant advantages for mobile use by reducing model size, speeding up inference, lowering power consumption, and improving hardware utilization, all while maintaining acceptable accuracy levels.

For comparison and variability, we also tested the MobileNetV2 architecture using the ASL Alphabet dataset (Fig. 4). This dataset comprises 87 000 images, each of size 200×200 pixels, and includes 29 classes representing different hand gestures in the American Sign Language alphabet. Testing MobileNetV2 alongside our optimized, quantized ViT model allowed us to evaluate how these two different architectures performed on the gesture recognition task. By comparing their performance in terms of accuracy, inference speed, and resource usage, we gained valuable insights into the strengths and weaknesses of each architecture, providing a comprehensive understanding of how well these models adapt to real-world tasks on mobile devices.

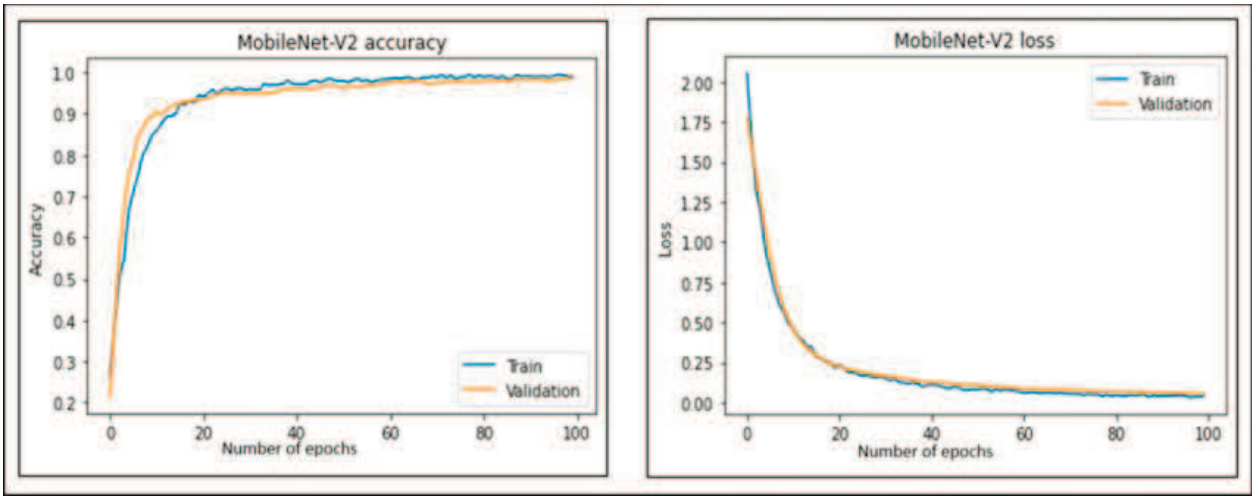

Fig. 4. MobileNetV2 training and validation accuracy (left) and loss (right) over 100 epochs on the ASL Alphabet dataset / Точність навчання та перевірки MobileNetV2 (ліворуч) і втрати (праворуч) протягом 100 епох на наборі даних ASL Alphabet

Table 1. Models and datasets used for experiments / Моделі та набори даних, використані для експериментів

Model	Architecture	Dataset
MobileNetV2	CNN	ImageNet
Quantized MobileNetV2	CNN	ImageNet
MobileNetV2	CNN	ASL Alphabet
MobTransformer	ViT	MNIST
Quantized MobTransformer	ViT	MNIST
Graph-optimized Quantized MobTransformer	ViT	MNIST

Measuring models' performance. To better understand the influence of optimization techniques, experiments were conducted with 6 different models (see Table 1).

Model size is one of the most critical characteristics of ML models for mobile devices. Mobile devices typically have limited computational resources compared to desktop or server environments. Therefore, models must be lightweight to ensure they can be deployed effectively without consuming excessive memory or processing power. Smaller models not only save memory but also reduce latency or inference speed, which is crucial for real-time applications such as image recognition or natural language processing. The inference speed can be influenced by various factors, including the model architecture, the optimization techniques used (such as quantization and pruning), and the underlying hardware capabilities.

To better understand which models suit our specific needs, we evaluated several parameters important for the usage of mobile phones: model physical size, random-access memory (RAM) and central processing unit (CPU) usage, and inference time.

All experiments were conducted on an Android device – Xiaomi Redmi Note 8 Pro, which was equipped with a Mediatek MT6785 CPU (8 cores) and 6 GB of RAM. Metrics measured with built in profilers in Android Studio integrated development environment over 1200 iterations. The models were integrated into the mobile app using TensorFlow Lite without graphics processing unit usage. The results are presented in Table 2.

Table 2. Results of the experiments / Результати експериментів

Model	Physical size	RAM usage, Mb	CPU usage, %	Inference time, percentile/ms	Average model load time, ms
MobileNet v2 + ImageNet	14 Mb	140–155	18–22	0.95 / 53 0.9 / 46 0.5 / 37 0.1 / 32	968
Quantized MobileNet v2 + ImageNet	3.6 Mb	140–155	18–20	0.95 / 65 0.9 / 58 0.5 / 48 0.1 / 47	950
MobileNet v2 + ASL Alphabet	3.3 Mb	120–135	12–16	0.95 / 50 0.9 / 46 0.5 / 36 0.1 / 33	947
ViT + MNIST	3.5 Mb	118–145	7–14	0.95 / 16 0.9 / 16 0.5 / 12 0.1 / 7	932
Quantized ViT + MNIST	899 Kb	109–140	7–13	0.95 / 10 0.9 / 10 0.5 / 5 0.1 / 3	906
Graph optimized, Quantized ViT + MNIST	899 Kb	105–130	6–12	0.95 / 9 0.9 / 8 0.5 / 5 0.1 / 3	904

Discussion of research results. As a result of these optimizations, the model's inference speed improved significantly across all tested confidence levels. The inference time was reduced by approximately 44 % for the 95th percentile, from 16 ms in the regular ViT model to 9 ms in the graph-optimized, quantized version, and 58 % for the 50th percentile, respectively. This reduction in time was accompanied by a decrease in memory usage and CPU load, making the model far more efficient for deployment on mobile devices. The optimized model demonstrated a lower RAM footprint (from 145 MB to 130 MB maximum) and CPU usage (dropping from 14 % to 12 % maximum), which translates into reduced energy consumption – an essential factor for battery-powered mobile devices. Less noticeable improvements were observed in the MobileNetV2 model in regards to inference time, still giving a noticeable reduction in the model size.

For both models, post-training optimizations helped to reduce the physical size by 74 %, which can positively influence the overall rate of app download [23], and slightly, about 3 %, improve the initial model loading time during application launch.

These improvements not only enable real-time performance for lightweight applications such as gesture recognition and image classification, but they also set the stage for deploying larger, more complex deep learning models on mobile platforms. The combination of quantization and graph optimization ensures that the model operates within the strict resource constraints of mobile hardware while maintaining high accuracy and responsiveness. These gains in efficiency will likely become even more pronounced as we scale the model and dataset, making graph optimization a key factor in ensuring practical and effective deep-learning solutions for mobile and embedded environments.

The scientific novelty of the obtained research results is the proposed approach to the use of machine learning models together with optimization techniques that lead to improved classification performance on mobile devices.

The practical significance of the research results – the research demonstrates the effectiveness of the machine

learning model optimizations and their potential impact on the development and deployment of applications in mobile and embedded environments.

Conclusions / Висновки

The research results indicate that quantization and graph optimization can lead to significant reductions in model size and inference time, making it more suitable for mobile and embedded deployment. While there may be some trade-offs in inference time for certain models, the overall efficiency in terms of CPU usage and the drastic reduction in physical size make optimized models a compelling choice for mobile applications. The performance of the models also highlights the importance of dataset selection, as it can impact both accuracy and inference speed. Ultimately, the choice between optimization techniques should consider the specific application requirements, including the need for speed, resource constraints, and the nature of the data being processed.

Several experiments can be conducted to discover further areas for improvement in machine learning models for mobile deployment. These include comparing various quantization techniques, exploring alternative lightweight architectures, and optimizing hyperparameters to assess their impact on performance. Evaluating models on diverse datasets and under real-world conditions can provide insights into robustness and generalization.

References

1. ITU/UN tech agency (2024, May 19). *Measuring Digital Development – Facts and Figures 2023*. International Telecommunication Union. https://www.itu.int/hub/publication/d–ind–ict_mdd–2023–1/
2. Statista (2024, June 14). *Topic: US smartphone market*. https://www.statista.com/topics/2711/us–smartphone–market/#topicOverview
3. Brand, L. (2023). Towards improved user experience for artificial intelligence systems. In *Engineering Applications of Neural Networks* (pp. 33–44). Cham. https://doi.org/10.1007/978–3–031–34204–2_4
4. Li, Y., Dang, X., Tian, H., Sun, T., Wang, Z., Ma, L., Klein, J., & Bissyande, T. F. (2022). AI-driven mobile apps: An explorative study. *ArXiv*. https://doi.org/10.48550/arxiv.2212.01635
5. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. *ArXiv*. https://doi.org/10.48550/arxiv.1706.03762
6. Sun, C., Shrivastava, A., Singh, S., & Gupta, A. K. (2017). Revisiting unreasonable effectiveness of data in deep learning era. In *2017 IEEE International Conference on Computer Vision (ICCV)* (pp. 843–852). https://doi.org/10.48550/arXiv.1707.02968
7. Sandler, M., Howard, A. G., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 4510–4520). https://doi.org/10.1109/CVPR.2018.00474
8. Howard, A. G., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., & Adam, H. (2019). Searching for mobilenetv3. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (pp. 1314–1324). https://doi.org/10.1109/ICCV.2019.00140
9. Tan, M., & Le, Q. V. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. *ArXiv*. https://doi.org/10.48550/arXiv.1905.11946
10. Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., & Jegou, H. (2021). Training data–efficient image transformers & distillation through attention. In *Proceedings of the 38th International Conference on Machine Learning*, 10347–10357.
11. Xiao, T., Singh, M., Mintun, E., Darrell, T., Dollar, P., & Girshick, R. (2021). Early convolutions help transformers see better. *ArXiv*. https://doi.org/10.48550/arXiv.2106.14881
12. Ranftl, R., Bochkovskiy, A., & Koltun, V. (2021). Vision transformers for dense prediction. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)* (pp. 12159–12168). https://doi.org/10.1109/ICCV48922.2021.01196
13. Chen, L., Papandreou, G., Schroff, F., & Adam, H. (2017). Rethinking atrous convolution for semantic image segmentation. *ArXiv*, abs/1706.05587.
14. Wu, H., Xiao, B., Codella, N., Liu, M., Dai, X., Yuan, L., & Zhang, L. (2021). Cvt: Introducing convolutions to vision transformers. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)* (pp. 22–31). https://doi.org/10.1109/ICCV48922.2021.00009
15. Srinivas, A., Lin, T.-Y., Parmar, N., Shlens, J., Abbeel, P., & Vaswani, A. (2021). Bottleneck transformers for visual recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 16519–16529). https://doi.org/10.1109/CVPR46437.2021.01625
16. d'Ascoli, S., Touvron, H., Leavitt, M., Morcos, A., Biroli, G., & Sagun, L. (2021). Convit: Improving vision transformers with soft convolutional inductive biases. *International Conference on Machine Learning*, 2286–2296. https://doi.org/10.1088/1742–5468/ac9830
17. Ryoo, M., Piergiovanni, A. J., Arnab, A., Dehghani, M., & Angelova, A. (2021). Tokenlearner: Adaptive space–time tokenization for videos. *Advances in Neural Information Processing Systems*, 34.
18. Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., & Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 10012–10022). https://doi.org/10.1109/ICCV48922.2021.00986
19. Caron, M. (2021). Emerging properties in self-supervised vision transformers. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)* (pp. 9630–9640). https://doi.org/10.1109/ICCV48922.2021.00951
20. Kitaev, N., Kaiser, Ł., & Levskaya, A. (2020). Reformer: The efficient transformer. *ArXiv*. https://doi.org/10.48550/arXiv.2001.04451
21. Mehta, S., & Rastegari, M. (2021). MobileViT: Light-weight, general–purpose, and mobile–friendly vision transformer. *ArXiv*. https://doi.org/10.48550/arXiv.2110.02178
22. Wu, H., Judd, P., Zhang, X., Isaev, M., & Micikevicius, P. (2020). Integer quantization for deep learning inference: Principles and empirical evaluation. *ArXiv*. https://doi.org/10.48550/arXiv.2004.09602
23. Wan, L. (2014). A study of factors affecting mobile application download. *Journal of Digital Convergence, 12*, 189–196. https://doi.org/10.14400/JDC.2014.12.7.189

В. Я. Чорненький, І. Я. Казимира

Національний університет "Львівська політехніка", м. Львів, Україна

МОДЕЛІ МАШИННОГО НАВЧАННЯ ТА ОПТИМІЗАЦІЙНІ СТРАТЕГІЇ ДЛЯ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ КЛАСИФІКАЦІЇ НА МОБІЛЬНИХ ПРИСТРОЯХ

У роботі розглянуто підходи до вдосконалення моделей машинного навчання у разі їх застосування для класифікації у мобільних додатках, вплив оптимізаційних технік на підвищення ефективності класифікації в реальному часі на мобільних пристроях.

Основну увагу в дослідженні приділено порівнянню MobileNetV2, згорткової нейронної мережі, розробленої для мобільних додатків, і візуальних трансформерів (ViT), які продемонстрували успіх у завданнях розпізнавання зображень. Замість стандартних згорткових операцій MobileNetV2 використовує глибинні відокремлені згортки, що істотно зменшує кількість обчислень та параметрів моделі, а також використовує залишкові зв'язки, які дають змогу зберігати інформацію із попередніх шарів, покращуючи навчання моделі. ViT використовує механізм самоуваги для виявлення глобальних залежностей між частинами зображення, що дає змогу враховувати як локальні, так і глобальні ознаки без використання згорткових шарів. Зображення у ViT розділяють на патчі фіксованого розміру і кожен патч обробляють як "слово" в тексті у звичайних трансформерах, що спрощує роботу з великими зображеннями.

У статті оцінено продуктивність обидвох моделей до і після застосування оптимізаційних технік, таких як квантизація – процес, який знижує точність коефіцієнтів моделі з 32-бітної до 8-бітної, що істотно зменшує розмір самої моделі та підвищує швидкість класифікації. Встановлено, що квантизація – одна із найефективніших оптимізаційних стратегій для мобільних середовищ, оскільки зменшує розмір моделі до 74 % і збільшує швидкість класифікації до 44 % у ViT. Крім того, досліджено роль технік оптимізації графа, таких як злиття операторів, обрізання та зміна послідовності виконання операцій, у зменшенні обчислювальної складності та підвищенні продуктивності на пристроях із обмеженими ресурсами. Ці техніки оптимізують виконання операцій у межах обчислювального графа, мінімізуючи використання пам'яті та підвищуючи паралелізм, що є критичним для додатків у реальному часі на мобільних пристроях.

Здійснено експерименти та досліджено результати на різних наборах даних, зокрема MNIST і ASL Alphabet, що демонструють істотне підвищення продуктивності, досягнуте завдяки оптимізації. Дослідження показує, що післятренувальна квантизація та оптимізація графа можуть зменшити розмір моделі, час класифікації та використання центрального процесора, роблячи моделі машинного навчання придатнішими для мобільних додатків. Експерименти, здійснені на пристрої Xiaomi Redmi Note 8 Pro з операційною системою Android та використанням TensorFlow Lite для інтеграції, продемонстрували практичні переваги такої оптимізації у реальних мобільних середовищах.

Виявлено, що такі оптимізаційні техніки, як квантизація та оптимізація графа, важливі для розгортання моделей машинного навчання на мобільних пристроях, для яких обмеження ресурсів і продуктивність у реальному часі мають вирішальне значення. Ці техніки дають змогу істотно зменшити розмір моделі та час класифікації, не жертвуючи точністю, що уможливлює практичне використання моделей глибинного навчання у мобільних додатках.

Ключові слова: глибинне навчання, згорткова нейронна мережа, візуальні трансформери, мобільні застосунки.

Інформація про авторів:

Чорненький Володимир Ярославович, аспірант, кафедра автоматизованих систем управління.
E-mail: volodymyr.y.chornenkyi@lpnu.ua; https://orcid.org/0009–0000–0569–6623

Казимира Ірина Ярославівна, канд. техн. наук, доцент, кафедра автоматизованих систем управління.
E-mail:iryna.y.kazymyra@lpnu.ua; https://orcid.org/0009–0000–0569–6623

Цитування за ДСТУ: Чорненький В. Я., Казимира І. Я. Моделі машинного навчання та оптимізаційні стратегії для підвищення ефективності класифікації на мобільних пристроях. Український журнал інформаційних технологій. 2024, т. 6, № 2. С. 74–82.

Citation APA: Chornenkyi, V. Y., & Kazymyra, I. Y. (2024). ML models and optimization strategies for enhancing the performance of classification on mobile devices. *Ukrainian Journal of Information Technology, 6*(2), 74–82. https://doi.org/10.23939/ujit2024.02.074