JURIKOM (Jurnal Riset Komputer), Vol. 13 No. 5, Oktober 2026
e-ISSN 2715-7393 (Media Online), p-ISSN 2407-389X (Media Cetak)
DOI 10.30865/jurikom.v15i5.10047
Hal 1280-1291
https://ejurnal.stmik-budidarma.ac.id/index.php/jurikom

Comparative Evaluation of TensorFlow Lite Quantization Variants for MobileNetV2-Based Batik Motif Classification: Accuracy, Model Size, and Inference Latency

Dewi Handayani Untari Ningsih[1], Eri Zuliarso[1,*], Jati Sasongko Wibowo[1],Dwi Budi Santoso[2], Hari Murti[2]

[1] Teknologi Informasi dan Industri, Teknik Informatika, Unisbank, Semarang, Indonesia
[2] Teknologi Informasi dan Industri , Sistem Informasi, Unisbank, Semarang, Indonesia
Email: [1]dewi_h@edu.unisbank.ac.id, [2,*]eri299@edu.unisbank.ac.id, [3]jatisw@ edu.unisbank.ac.id, [4]dbs@ edu.unisbank.ac.id, [5]harimurti@ edu.unisbank.ac.id
Email Penulis Korespondensi: eri299@edu.unisbank.ac.id[*]
Submitted **28-07-2026**; Accepted **02-10-2026**; Published **31-10-2026**

Abstract

Convolutional Neural Network (CNN) models achieve high accuracy in visual recognition, but their size and computational cost limit deployment on mobile devices, so batik motif recognition remains difficult to bring to artisans and small creative enterprises. This study evaluates how post-training quantization with TensorFlow Lite affects the classification accuracy, model size, and inference latency of a MobileNetV2 transfer-learning model for batik motif recognition, in order to identify the deployment configuration that offers the best balance among the three. The experimental workflow adapted a pre-trained MobileNetV2 network to the Batik Wastra Nusantara dataset, which covers batik designs originating from different parts of Indonesia with substantial variation in color, ornamentation, and pattern complexity, converted the trained model into Dynamic Range, Float16, and Int8 TensorFlow Lite variants, and compared them under identical conditions. Evaluation on 644 valid images from 14 motif classes produced a test accuracy of 43.08%, corresponding to 28 of the 65 test images, a macro F1-score of 41.65%, and a weighted F1-score of 40.39%. Across the four model formats the accuracy spans only three test images, from 41.54% to 46.15%, a range that lies well inside the confidence interval implied by a 65-image test subset, so the variants cannot be separated on accuracy at this sample size. The deployment metrics differ substantially: relative to the unquantized reference, the Float16 variant reduced model size from 19.04 MB to 4.29 MB (77.47%) and inference latency from 205.52 ms to 28.60 ms (86.08%), while the Int8 variant was the fastest at 14.87 ms and the Dynamic Range variant the smallest at 2.41 MB. These findings indicate that the benefit of TensorFlow Lite optimization lies in storage and latency rather than in accuracy, and that the accuracy attainable on this dataset is constrained by its limited size and the visual similarity among motifs. The study contributes the deployment measurements, namely model size and inference latency for each quantization variant, that comparable batik classification studies do not report, and identifies dataset expansion as the prerequisite for reaching production-level accuracy in AI-based digital innovation for Indonesia's creative industries.

Keywords: Batik Image; CNN; Post-Training Quantization; TensorFlow Lite; Transfer Learning.

1. INTRODUCTION

Batik is a UNESCO-recognized Indonesian cultural heritage, with motifs steeped in philosophy and regional identity [1]. Each region exhibits unique visual characteristics, such as Trutum, Mega Mendung, and Semen Romo, creating a diverse array of patterns. In the digital age, automatic recognition and classification of batik motifs is crucial for preservation, documentation, and the development of the creative industry [2].

The swift progression of AI and deep learning has enabled systems adept at discerning visual patterns. CNN is a promising technique highly effective in computer-vision tasks like object detection [3], face recognition [4], and image classification [5]. Automatic classification of these patterns is crucial for building digital archives and bolstering creative industries [6], and strengthening the nation's cultural identity through technology integration [7].

Despite remarkable progress in CNN-based image classification, most high-accuracy models [8] such as VGG-16 [9], ResNet [10], or Inception are computationally heavy and demand substantial memory resources [11]. Current model limitations hinder real-time implementation on mobile/embedded systems used by SMEs and artisans. There is a need for lightweight, precise deep-learning models. TensorFlow Lite (TFLite) is a viable solution for optimization and efficient inference on edge devices [12].

Previous studies on batik image classification have primarily focused on improving accuracy through model variation or feature extraction techniques [13], [14]. Meranggi et al. [13] reported a best test accuracy of 88.88% using a pre-trained ResNet-18 model after correcting and improving the dataset, while Sastypratiwi et al. [14] reported 100% test accuracy for a MobileNet transfer-learning model on 1,520 images from four batik classes, compared with 95% for the same architecture trained without transfer learning. Nevertheless, limited attention has been given to optimizing transfer-learning-based CNN models for deployment through TensorFlow Lite [15], [16]. Most studies remain confined to desktop experiments and do not report real-world latency or model size. This highlights a lack of research integrating transfer learning, fine-tuning, and TFLite optimization for batik recognition.

Research on the classification of batik motifs in Indonesia also shows a positive trend [17]. Syafi'i and Khomsah classified Banyumasan batik motifs by combining CNN and VGG16 with Canny edge detection, using 572 original images that were augmented into 5,148 images across 11 classes. Their four CNN variants reached 95–99%

Copyright © 2026 The **Dewi Handayani Untari Ningsih**, Page 1280
This Journal is licensed under a Creative Commons Attribution 4.0 International License

JURIKOM (Jurnal Riset Komputer), Vol. 13 No. 5, Oktober 2026
e-ISSN 2715-7393 (Media Online), p-ISSN 2407-389X (Media Cetak)
DOI 10.30865/jurikom.v15i5.10047
Hal 1280-1291
https://ejurnal.stmik-budidarma.ac.id/index.php/jurikom

accuracy and VGG16 reached 100% without edge detection, whereas the same models declined to 91–94% and 99% respectively once Canny edge detection was applied. Integrating Canny edge detection therefore lowered predictive performance, indicating that this preprocessing technique was less effective for capturing the intricate characteristics of batik, so that combining Canny edge filtering with a CNN architecture is not an optimal choice for identifying batik motifs. Other studies have likewise identified batik motifs through CNN-based deep learning approaches [18].

MobileNetV2 is ideal for mobile devices due to its efficiency and parameter reduction, enabled by depthwise separable convolutions and inverted residuals [19]. Nevertheless, further optimizations regarding precision, memory footprint, and latency remain essential to maximize its performance on resource-constrained hardware [20].

TensorFlow Lite (TFLite) provides a framework for deploying machine-learning models on smartphones, Internet of Things devices, and other edge platforms [21], [22]. It supports model conversion and optimization techniques intended to reduce storage requirements and inference latency. Quantization converts model parameters from high-precision floating-point representations into lower-precision formats, such as Float16 or Int8, thereby reducing model size and computational cost [23]. Dynamic-range quantization compresses model weights while retaining floating-point calculations for selected operations, whereas Float16 quantization reduces weight precision with relatively limited accuracy degradation. Full integer quantization converts both weights and activations into integer representations and can provide faster inference, although the loss of numerical precision may reduce classification accuracy. Other optimization strategies include pruning, which removes weights with low contribution to reduce network complexity [24], and hardware delegates such as GPU or NNAPI, which can accelerate inference on supported devices [25].

Building on prior work on quantized CNN implementation with TensorFlow Lite, this study develops a transfer-learning framework for efficient batik motif classification [26]. MobileNetV2 was adapted to the Batik Wastra Nusantara dataset through transfer learning and partial fine-tuning, then optimized using TensorFlow Lite quantization for comparative deployment evaluation [27]. After model training, post-training quantization generated multiple TensorFlow Lite implementations with different numerical precisions, including Dynamic, Float16, and Int8, to assess deployment efficiency [14]. The model variants were evaluated based on classification accuracy, storage requirements, and inference latency. The comparison was designed to determine the most balanced deployment option by considering both predictive capability and computational cost, rather than selecting a model solely because it was smaller or faster.

This study supports mobile-based cultural heritage applications by enabling lightweight on-device batik recognition. It helps artisans catalog, document, and promote motifs offline, which reduces internet dependency, shortens response times, enhances data privacy, and works efficiently in low-infrastructure environments.

Accordingly, the objective of this research is to optimize a MobileNetV2 transfer-learning model for batik motif classification and to determine the TensorFlow Lite quantization variant that provides the best balance between classification accuracy, model size, and inference latency on mobile devices. Because MobileNetV2 is itself a mobile-oriented architecture, this study quantifies the additional efficiency obtainable by quantizing an already lightweight backbone; it does not compare lightweight networks against heavyweight architectures such as VGG-16 or ResNet, which are cited above only to motivate the deployment problem. To meet this objective, the study contributes an integrated experimental framework that combines dataset validation, transfer learning, fine-tuning, multiclass evaluation, TensorFlow Lite conversion, and deployment-oriented benchmarking [28], [29]. Unlike studies that evaluate classification accuracy alone, this work examines the relationship among predictive performance, storage efficiency, and inference latency.

2. RESEARCH METHODOLOGY

2.1 Research Stages

The research was carried out in six sequential stages: designing the classification model architecture, training the CNN, applying transfer learning and partial fine-tuning, acquiring and augmenting the batik image data, converting and optimizing the trained model with TensorFlow Lite, and evaluating both classification performance and on-device inference behaviour. Each stage is described in the following subsections.

2.2 Model Architecture Design

A CNN is a hierarchical deep-learning model that learns spatial feature representations through multiple convolutional and pooling stages [30]. CNNs work by extracting features from an image through a series of filters (kernels) applied to images of Indonesian batik motifs [31]. Figure 1 illustrates the proposed model architecture. The study employs transfer learning followed by partial fine-tuning of a lightweight convolutional network, with subsequent TensorFlow Lite optimization to support efficient inference on mobile devices.

This Journal is licensed under a Creative Commons Attribution 4.0 International License

JURIKOM (Jurnal Riset Komputer), Vol. 13 No. 5, Oktober 2026
e-ISSN 2715-7393 (Media Online), p-ISSN 2407-389X (Media Cetak)
DOI 10.30865/jurikom.v15i5.10047
Hal 1280-1291
https://ejurnal.stmik-budidarma.ac.id/index.php/jurikom

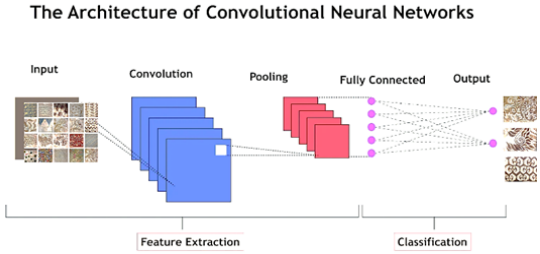


Figure 1. CNN Architecture Batik Motif

A CNN progressively learns discriminative visual features using convolutional, activation, and pooling layers, whereas fully connected layers map the extracted representations to class labels. Effective optimization using gradient descent-based algorithms and appropriate regularization strategies further improves convergence and model generalization [32],[33].

Formula for convolution Eq(1):

$$Y\,(i,j) = \sum_m \sum_n X\,(i+m, j+n).K\,(m,n) \quad (1)$$

Here, $Y(i,j)$ is the output pixel value at position (i,j), calculated by multiplying the input image pixel $X(i+m,j+n)$ with the kernel weight $K(m,n)$.

ReLU activation function Eq.(2):

$$\text{ReLU}\,(x) = \max\,(0,x) \quad (2)$$

This function is used to provide non-linearity to the network and help the model in learning more complex representations.

2.3 CNN Model Training

Model training was performed to enable the configured CNN to learn feature representations that support accurate batik motif recognition. The MobileNetV2 model was trained on preprocessed batik images using a gradient-based optimizer to minimize categorical cross-entropy loss. This process enabled discriminative feature learning while preparing the model for TensorFlow Lite conversion and deployment on resource-constrained mobile and edge devices [34], [35]**.**

Cross-Entropy loss function formula Eq.(3):

$$L(y,\hat{y}) = -\sum_{i=1}^{C} y_i.\log\,(\hat{y}_i) \quad (3)$$

The term $L(y,\hat{y})$represents the loss computed over Cbatik classes, where $y_i \in \{0,1\}$denotes the ground-truth label and $\hat{y}_i$indicates the model-assigned confidence for the i-th class.

2.4 Transfer Learning and Fine-Tuning

Training proceeded in two phases. First, MobileNetV2 retained its pre-trained convolutional parameters while the added classifier was optimized on the batik dataset. Next, the final 30 layers were released and updated with a smaller learning rate. This procedure adapted ImageNet-derived representations to batik-specific patterns while limiting computational cost and data demand. Figure 2 presents the complete training and refinement workflow.

Transfer learning was applied by reusing a MobileNetV2 network pre-trained on ImageNet as the feature extractor and by fine-tuning its upper layers on the batik dataset. MobileNetV2 was selected instead of other lightweight backbones such as EfficientNet-B0 or SqueezeNet because of its depthwise separable convolutions and inverted residual structure, which keep the parameter count low enough for mobile deployment.

Each 224 × 224 RGB image is transformed into high-level feature representations by stacked depthwise separable convolution operations. These representations are condensed through global average pooling and subsequently mapped to 14 output categories, corresponding to the 14 batik motif classes retained after dataset validation, using a dense prediction module with softmax activation.

The training process used the Adam optimizer with an initial learning rate of 1e-4, which was lowered during fine-tuning to allow smoother convergence. The classification head was trained for 12 epochs with a batch size of 16, followed by 10 epochs of partial fine-tuning of the last 30 layers. Early Stopping and ReduceLROnPlateau were applied to limit overfitting and to keep the learning process stable; the complete configuration is listed in Table 1.

Copyright © 2026 The **Dewi Handayani Untari Ningsih**, Page 1282
This Journal is licensed under a Creative Commons Attribution 4.0 International License

JURIKOM (Jurnal Riset Komputer), Vol. 13 No. 5, Oktober 2026
e-ISSN 2715-7393 (Media Online), p-ISSN 2407-389X (Media Cetak)
DOI 10.30865/jurikom.v15i5.10047
Hal 1280-1291
https://ejurnal.stmik-budidarma.ac.id/index.php/jurikom

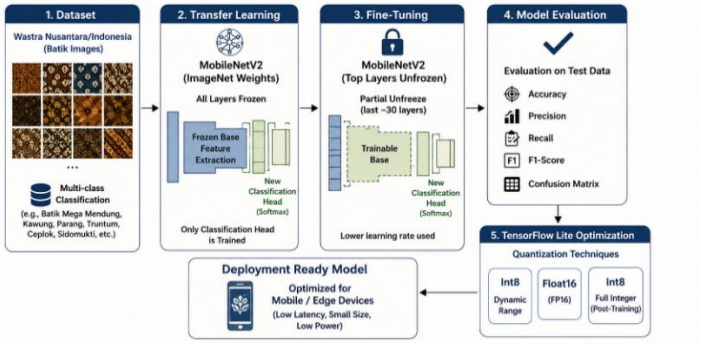


Figure 2. Transfer learning and Fine Tuning process flow

2.5 Data Acquisition and Augmentation

The dataset used in this study is the publicly available Batik Wastra Nusantara collection obtained from Kaggle and described in Section 3.1; no images were collected by the authors. Augmentation was performed using rotation, flipping, zooming, and brightness shifting to improve generalization [36].
Brightness Augmentation Formula Eq.(4):

$$I_{aug(x,y)}=\alpha.I(x,y)+\beta \quad (4)$$

Where α and β is a factor that increases brightness and contrast.

2.6 TensorFlow Lite Conversion and Optimization

Model converted to TensorFlow Lite with quantization [37],[38]:

Post-training quantization reduces the numerical precision of model parameters after training rather than during it [39]. The integer mapping actually applied in this study is given in Eq. (5) below.
Pruning and hardware delegates were not applied in this study; the optimization scope is limited to post-training quantization, although pruning remains a complementary route to further compression [23].
Performance analysis covered classification outcomes and deployment cost, represented by accuracy, precision, recall, F1-score, file size, and inference delay.

Optimizations are performed to ensure CNN models can run on resource-constrained devices such as smartphones [41]. Deployment optimization comprised three stages: model validation, post-training quantization into Dynamic Range, Float16, and Int8 TensorFlow Lite variants, and comparative benchmarking using classification accuracy, model size, and inference latency to identify the most efficient deployment configuration.

Deployment preparation was organized into three sequential stages to optimize the trained CNN for resource-constrained edge devices:

- Model Preparation
  The trained MobileNetV2 model was benchmarked to establish a baseline before TensorFlow Lite optimization.
- Post-Training Quantization
  Post-training quantization generated Dynamic Range, Float16, and Int8 TensorFlow Lite models by reducing FP32 precision, improving storage efficiency and inference performance.
  For integer quantization, a floating-point value $w$is mapped to a quantized value q as follows in Eq. (5):

$$q = \text{clip}\left(\text{round}\left(\frac{w}{s}\right) + z,\ q_{\min}, q_{\max}\right), \quad (5)$$

  where s denotes the scale factor, z represents the zero-point, and $q_{\min}$and $q_{\max}$define the numerical range of the target integer format. The approximate reconstructed value is expressed as Eq. (6):

$$\hat{w} = s(q - z). \quad (6)$$

  Dynamic Range quantizes weights, Float16 reduces parameter precision, and Int8 quantizes both weights and activations using representative calibration samples.
- Comparative Benchmarking
  TensorFlow and TensorFlow Lite models were benchmarked under identical conditions using classification accuracy, model size, and inference latency to identify the optimal deployment configuration.

2.7 Model Evaluation and Device Testing

Once training was completed, the model was tested on unseen data to quantify its classification performance. The assessment included the following measures:

Accuracy: Indicates the percentage of samples classified correctly among all predictions

$$\boldsymbol{Accuracy} = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} \quad (7)$$

Copyright © 2026 The **Dewi Handayani Untari Ningsih**, Page 1283
This Journal is licensed under a Creative Commons Attribution 4.0 International License

JURIKOM (Jurnal Riset Komputer), Vol. 13 No. 5, Oktober 2026
e-ISSN 2715-7393 (Media Online), p-ISSN 2407-389X (Media Cetak)
DOI 10.30865/jurikom.v15i5.10047
Hal 1280-1291
https://ejurnal.stmik-budidarma.ac.id/index.php/jurikom

Precision, Recall, and F1-score: Complement overall accuracy by providing a more detailed evaluation of class-specific prediction performance.
The converted TensorFlow Lite models were then executed to verify that inference speed and classification accuracy remained acceptable under resource constraints. All accuracy and latency figures reported in Section 3 were obtained on a single evaluation platform [authors to specify: processor, RAM, operating system, and TensorFlow / TensorFlow Lite runtime versions], with latency measured as single-image inference over the test subset. The reported latencies therefore characterise the relative differences among model formats rather than absolute performance on a particular smartphone.

Inference testing using TensorFlow Lite Eq. (8):

$$\hat{y} = \text{Model}\ (X_{input}) \tag{8}$$

where:

$\hat{Y}$ is the class prediction result for the input image X_{input}

TensorFlow Lite optimization was applied to the trained network to improve execution efficiency on mobile devices without substantially compromising predictive performance [40].

3. RESULT AND DISCUSSION

3.1 Batik Motif Dataset Overview

The Batik Wastra Nusantara collection contains images representing diverse Indonesian batik motifs that are grouped into predefined categories (Figure 3). After data validation and cleaning, a total of 644 valid images from 14 selected batik motif classes were retained for experimentation, with approximately 30–50 images per class. Stratified sampling produced 515 images for training, 64 for validation, and 65 for testing, ensuring that each batik motif category was proportionally represented throughout the experimental subsets.

A major challenge of this dataset lies in its relatively limited number of samples and the high visual similarity among several batik motifs. Many classes share comparable geometric patterns, repetitive ornaments, textures, and color compositions, making class discrimination more difficult. These characteristics increase the complexity of the classification task and may affect the generalization capability of CNN-based models.

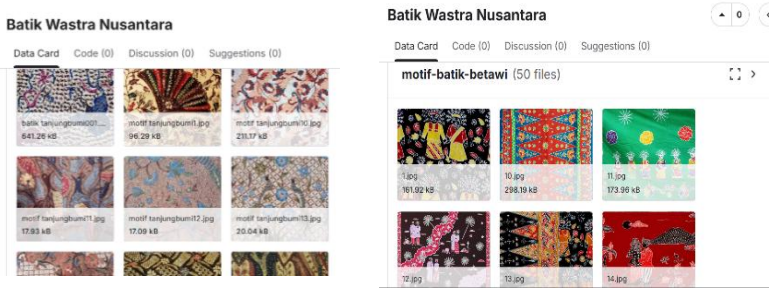
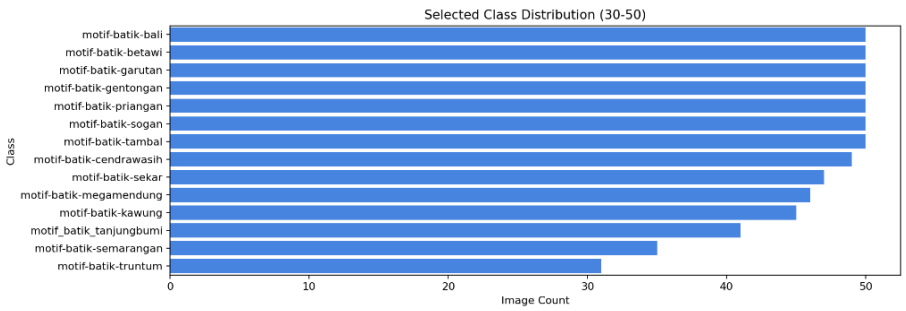


Figure 3. Wastra Nusantara batik dataset: (https://www.kaggle.com/datasets/dwibudisantoso/batik-wastra-nusantara)

3.2 Data Distribution by Class

The pipeline filters the dataset to 30-50 images per class, resulting in 14 selected batik motifs. During validation, one invalid image (18.jpg) was found in the *motif-batik-cendrawasih* class.

Figure 4 illustrates the distribution of images across the selected batik motif classes after the filtering process. The dataset contains 14 classes with a relatively balanced number of samples, ranging from 31 to 50 images per class. Most classes, including *motif-batik-bali*, *motif-batik-betawi*, *motif-batik-garutan*, and *motif-batik-gentongan*, contain approximately 50 images, while *motif-batik-truntum* has the smallest number of samples. This relatively balanced distribution reduces class imbalance during training, promoting more consistent feature extraction across batik motif classes while minimizing bias toward categories with a larger number of samples.



Copyright © 2026 The **Dewi Handayani Untari Ningsih**, Page 1284
This Journal is licensed under a Creative Commons Attribution 4.0 International License

JURIKOM (Jurnal Riset Komputer), Vol. 13 No. 5, Oktober 2026
e-ISSN 2715-7393 (Media Online), p-ISSN 2407-389X (Media Cetak)
DOI 10.30865/jurikom.v15i5.10047
Hal 1280-1291
https://ejurnal.stmik-budidarma.ac.id/index.php/jurikom

Figure 4. Dataset Distribution

A stratified data split generated 515 training images (79.97%), 64 validation images (9.94%), and 65 testing images (10.09%). Preserving the original class distribution supported reliable model optimization and objective performance evaluation on independent test samples. Because the test subset accounts for only about 10% of the data, the resulting accuracy is sensitive to a small number of prediction changes, and the reported values should therefore be read together with the class-level results in Section 4.

3.3 Transfer Learning Configuration and Fine-Tuning

A MobileNetV2 architecture pre-trained on ImageNet was adopted as the feature extraction network because of its lightweight computational design. The batik images were standardized to 224 × 224 pixels before training with 16-image mini-batches and a random seed of 42 to maintain reproducible experimental conditions.

The initial optimization phase preserved the pre-trained MobileNetV2 feature extractor, with learning restricted to the classification head for 12 epochs. Subsequently, parameter updates were extended to the final 30 layers for 10 epochs with a reduced learning rate. This refinement improved the representation of batik-specific visual patterns while retaining the transferable knowledge obtained from ImageNet. Table 1 presents the corresponding experimental settings.

Table 1. Experimental Settings for MobileNetV2 Transfer Learning

Parameter	Value
Base architecture	MobileNetV2
Initial weights	ImageNet
Input size	224 × 224 pixels
Batch size	16
Random seed	42
Classification head training	12 epochs
Fine-tuning	10 epochs
Number of unfrozen layers	Last 30 layers
First stage	Frozen backbone
Second stage	Partial fine-tuning
Final model task	Classification of 14 batik motif classes

Table 1 outlines the experimental training configuration. The classification framework employed a pre-trained MobileNetV2 architecture with an input size of 224 × 224 pixels. Training comprised an initial feature extraction phase using frozen layers for 12 epochs, followed by partial fine-tuning of the last 30 layers for 10 epochs to distinguish 14 batik motif classes.

3.4 CNN Model Testing

3.4.1 Test Results per Class

Classification results show that model performance differs significantly across classes. Some motifs are recognized well, while others remain difficult to distinguish, as summarized in Table 2.

Table 2. Class-Level Performance for Selected Motif Classes

Batik Motif Class	F1-Score	Interpretation
Megamendung Motif	0.9091	Excellent
Kawung Motif	0.8333	Good
Semarangan Motif	0.7500	Good
Bali Motif	Low	Difficult to recognize consistently
Tambal Motif	Low	Frequently confused with other motifs

Table 2 shows that Megamendung achieved the highest F1-score (0.9091), followed by Kawung (0.8333) and Semarangan (0.7500), indicating that these motifs possess distinctive visual characteristics that the model learns more easily. In contrast, Bali and Tambal motifs remained difficult to classify, reflecting greater intra-class variation and stronger visual similarity with other motifs, which increases the likelihood of misclassification. Motif complexity and inter-class visual resemblance therefore emerged as the key factors influencing the class-level outcomes. These class-level values must be read with caution. The test subset contains 65 images distributed over 14 classes, so each class is represented by only four to five images, and a single prediction can shift a class F1-score by more than ten percentage points. The values in Table 2 are consequently indicative of the relative difficulty of the motifs rather than precise estimates of per-class performance. [Authors: the complete 14-class classification report and the corresponding confusion matrix should be added so that the macro and weighted F1-scores in Table 3 can be verified.]

Copyright © 2026 The **Dewi Handayani Untari Ningsih**, Page 1285
This Journal is licensed under a Creative Commons Attribution 4.0 International License

JURIKOM (Jurnal Riset Komputer), Vol. 13 No. 5, Oktober 2026
e-ISSN 2715-7393 (Media Online), p-ISSN 2407-389X (Media Cetak)
DOI 10.30865/jurikom.v15i5.10047
Hal 1280-1291
https://ejurnal.stmik-budidarma.ac.id/index.php/jurikom

Megamendung achieved the highest classification performance among all evaluated batik motifs. The unique combination of cloud-like motifs, smooth contours, and layered decorative patterns forms distinctive feature representations, leading to improved discrimination among batik motif classes.

Kawung motif also produced favorable classification results because its repetitive geometric elements generated consistent visual characteristics. The regularity of these patterns supported stable feature extraction throughout network training.

In contrast, Bali and Tambal motifs exhibited relatively low classification performance. Bali motifs encompass a wide range of artistic variations in ornamentation and composition, whereas Tambal motifs are formed by combining multiple decorative patterns within a single design. These characteristics increase visual overlap with other batik classes, making accurate discrimination more challenging for the model

3.4.2 Overall Model Evaluation Results

Following transfer learning and partial fine-tuning, evaluation was performed on the test dataset using previously unseen batik images. The evaluation produced a test accuracy of 43.08%, corresponding to 28 correctly classified images out of the 65 in the test subset, accompanied by macro and weighted F1-scores of 41.65% and 40.39%, respectively. Because the test subset is small, this estimate carries a wide 95% confidence interval of approximately twelve percentage points in either direction, and the differences of a few percentage points reported later in this section must be interpreted in that light. These values demonstrate that the network successfully captured several motif-specific characteristics, although classification consistency across unseen samples remained moderate. The main contributing factors were the relatively small dataset and the high visual similarity among several motif classes. The overall evaluation results are summarized in Table 3.

Table 3. Overall Model Evaluation Results

Evaluation Metric	Value
Test Loss	1.8749
Test Accuracy	0.4308
Test Accuracy (%)	43.08%
Macro F1-Score	0.4165
Weighted F1-Score	0.4039

Table 3 summarizes the overall performance of the model on the test dataset. The model achieved a test accuracy of 43.08%, with macro and weighted F1-scores of 41.65% and 40.39%, respectively. These results indicate moderate classification performance and show that the model still has limited generalization ability across the batik motif classes

3.4.3 Model Conversion Results to TensorFlow Lite

The TensorFlow model was then converted into several TensorFlow Lite variants, namely dynamic quantization, float16 quantization, and int8 quantization (Table 4).

Table 4. Model Size Comparison

Model Format	Model Size	Size Reduction
TensorFlow SavedModel	19.04 MB	–
TFLite Dynamic	2.41 MB	87.34%
TFLite Float16	4.29 MB	77.47%
TFLite Int8	2.60 MB	86.34%

TFLite Dynamic produces the smallest file size, at 2.41 MB. This represents an 87.34% reduction compared to SavedModel. TFLite Int8 also produces a very small file size, at 2.60 MB. Meanwhile, TFLite Float16 has a file size of 4.29 MB. While larger than the other two variants, it is still significantly smaller than the original TensorFlow model. The reduced model size provides immediate benefits for mobile devices, including reduced storage requirements, shorter model loading times, lower memory usage, and lighter app distribution. Two qualifications apply to Table 4. First, the 19.04 MB reference is a TensorFlow SavedModel, which stores graph metadata and variable files that a TensorFlow Lite FlatBuffer does not, so the reduction percentages partly reflect a difference in container format rather than in parameter precision alone; an unquantized FP32 TensorFlow Lite model would be the strictly like-for-like baseline. Second, the smallest file is produced by the Dynamic Range variant rather than by the variant recommended in Section 3.6, which shows that file size alone does not determine the preferred deployment configuration.

3.5 Accuracy and Inference Latency Evaluation

Each model variant was evaluated to measure its classification accuracy and inference latency; the results are presented in Table 5.

Table 5. Comparison of Classification Accuracy and Inference Latency

Model	Accuracy	Inference Latency (ms)	Latency Reduction (%)

Copyright © 2026 The **Dewi Handayani Untari Ningsih**, Page 1286
This Journal is licensed under a Creative Commons Attribution 4.0 International License

JURIKOM (Jurnal Riset Komputer), Vol. 13 No. 5, Oktober 2026
e-ISSN 2715-7393 (Media Online), p-ISSN 2407-389X (Media Cetak)
DOI 10.30865/jurikom.v15i5.10047
Hal 1280-1291
https://ejurnal.stmik-budidarma.ac.id/index.php/jurikom

TensorFlow	46.15%	205.52	—
TFLite Dynamic	44.62%	60.42	70.60%
TFLite Float16	46.15%	28.60	86.08%
TFLite Int8	41.54%	14.87	92.77%

Based on Table 5, TensorFlow Lite conversion reduced inference latency substantially in every variant, while the accompanying accuracy differences were small. Relative to the unquantized TensorFlow reference, TFLite Dynamic reduced latency by 70.60%, TFLite Float16 by 86.08%, and TFLite Int8 by 92.77%. The corresponding accuracies are 44.62%, 46.15%, and 41.54%. Every value in the accuracy column is an exact multiple of one sixty-fifth, because the test subset contains 65 images: 30 correct predictions for TensorFlow and for Float16, 29 for Dynamic Range, and 27 for Int8. The entire spread across the four formats is therefore three images. At this sample size the 95% confidence interval around any of these values is approximately twelve percentage points wide in either direction, so the differences are not statistically distinguishable from zero. The statement that Float16 preserves the reference accuracy should accordingly be understood to mean that it produced an identical set of predictions on these 65 images, not that Float16 quantization has been shown to be lossless in general. Selection of a deployment configuration should therefore rest on the latency and size measurements, which differ by factors rather than by single images. On that basis TFLite Float16 is the preferred configuration for batik image classification on mobile and edge devices, since it attains an 86.08% latency reduction while reproducing the reference predictions exactly, with Int8 reserved for cases in which the further saving of 13.73 ms outweighs the risk of accuracy degradation.

3.6 Comparative Evaluation of Deployment Performance

Table 6. Comparative Characteristics of TensorFlow and TensorFlow Lite Models

Model	Advantages	Limitations	Recommended Use
TensorFlow	Unquantized reference implementation	Largest model size and highest inference latency	Server-based deployment or experimental evaluation
TFLite Dynamic	Smallest model size	One fewer correct prediction than the reference (not statistically significant)	Lightweight applications with limited storage
TFLite Float16	Reproduces the reference predictions while providing low inference latency	Larger model size than Dynamic and Int8	Mobile devices with Float16 support
TFLite Int8	Fastest inference latency	Largest observed accuracy drop (three fewer correct predictions); may blur fine batik texture	Highly resource-constrained devices

Table 6 compares the main advantages, limitations, and recommended deployment scenarios of each model format. The TensorFlow model serves as the unquantized reference but requires the greatest storage and computation. TFLite Dynamic prioritizes compactness, while TFLite Int8 provides the fastest inference. As noted in Section 3.5, the accuracy separating these formats spans only three test images and does not support a ranking on predictive performance; Table 6 is therefore a comparison of deployment characteristics, with accuracy treated as approximately constant across formats.

On that basis TFLite Float16 offers the best trade-off: it reproduces the reference predictions exactly while reducing latency by 86.08% and model size by 77.47%. TFLite Int8 is preferable when latency is the binding constraint, although full integer quantization may reduce the precision with which subtle batik textures and pattern details are represented, and a larger test set would be needed to establish whether the accuracy difference observed here is real.

3.7 Relative Efficiency Analysis

Table 7. Relative Efficiency Improvement of TensorFlow Lite Models

Model	Inference Speed Improvement (vs. TensorFlow)	Accuracy Change vs. TensorFlow (65-image test subset)
TFLite Dynamic	3.40× faster	−1 image (−1.53 pp; not significant)
TFLite Float16	7.19× faster	0 images (identical predictions)
TFLite Int8	13.82× faster	−3 images (−4.61 pp; not significant)

Table 7 summarizes the relative efficiency of the TensorFlow Lite models compared with the unquantized TensorFlow model. The speed-up factors are large and unambiguous, ranging from 3.40× to 13.82×. The accuracy

Copyright © 2026 The **Dewi Handayani Untari Ningsih**, Page 1287
This Journal is licensed under a Creative Commons Attribution 4.0 International License

JURIKOM (Jurnal Riset Komputer), Vol. 13 No. 5, Oktober 2026
e-ISSN 2715-7393 (Media Online), p-ISSN 2407-389X (Media Cetak)
DOI 10.30865/jurikom.v15i5.10047
Hal 1280-1291
https://ejurnal.stmik-budidarma.ac.id/index.php/jurikom

column, by contrast, expresses differences of between zero and three test images and is reported for completeness rather than as a basis for ranking the variants on predictive performance.

TFLite Float16 provided the best overall balance, achieving inference roughly seven times faster while reproducing the reference predictions. Although TFLite Int8 was the fastest, the choice between the two should be governed by the latency budget of the target application and confirmed on a larger evaluation set before either is adopted in production.

3.8 Consistency Analysis of the Reported Accuracy Results

Two different accuracy values, 43.08% and 46.15%, appear in the raw experimental outputs of this study. The discrepancy requires explanation, because both were produced by the same unquantized model on the same test subset.

Table 8. Comparison of the Reported Accuracy Values

Result Source	Accuracy	Evaluation Purpose
run_summary.json	43.08%	28 of 65 test images; evaluation through the training and validation data pipeline, reported as the primary classification result of this study
tf_vs_tflite_benchmark.csv	46.15%	30 of 65 test images; internal reference value of the deployment benchmark script, used only for comparison among model formats

Table 8 shows that the two values were produced by the same trained model on the same 65-image test subset and that they differ by exactly two images, 28 correct predictions against 30. The difference therefore cannot be attributed to a difference in evaluation purpose; it originates in the evaluation procedure, specifically in the image-loading and preprocessing path. The primary evaluation reuses the TensorFlow data pipeline applied during training, whereas the benchmark script loads and rescales each image independently, and small differences in resizing interpolation or input scaling are sufficient to change the predicted label for a small number of borderline images. [Authors: verify that both paths apply identical resizing and preprocessing, and report a single corrected figure if the two converge.] Pending that verification, the two values are used consistently throughout this article as follows. The figure of 43.08% is reported as the classification accuracy of the study, in the abstract, in Section 3.4.2, and in the conclusion. The figure of 46.15% is used only as the internal reference of the deployment benchmark in Sections 3.5 to 3.7, where each TensorFlow Lite variant is compared against it and never against 43.08%. Given the width of the confidence interval at a sample size of 65, the two values are in any case statistically indistinguishable.

3.9 Comparison with Previous Studies

To position the present results within the existing literature, Table 9 compares this study with previous batik image classification research in terms of method, dataset scale, reported accuracy, and the deployment metrics that were made available.

Table 9. Comparison with Previous Batik Image Classification Studies

Study	Method and dataset	Reported accuracy	Model size and inference latency
Syafi'i and Khomsah [17]	CNN (M1–M4) and VGG16 with Canny edge detection; 572 original images augmented to 5,148 images, 11 classes	Without Canny: 95–99% (CNN), 100% (VGG16). With Canny: 91–94% (CNN), 99% (VGG16)	Not reported
Meranggi et al. [13]	Pre-trained ResNet-18 with dataset improvement and patching; 598 and 621 images, 5 classes	66.14% on the original dataset; 88.88% on the improved dataset	Not reported
Sastypratiwi et al. [14]	MobileNet with transfer learning, converted to TensorFlow Lite and deployed in an Android application; 1,520 images, 4 classes	95% without transfer learning; 100% with transfer learning	Not measured; only training time (1,275 s) is reported
This study	MobileNetV2 transfer learning with TensorFlow Lite quantization; 644 images, 14 classes	43.08% (28/65) on the test set; 46.15% (30/65) as the internal benchmark reference	Float16: 4.29 MB and 28.60 ms; Int8: 2.60 MB and 14.87 ms

Table 9 shows that earlier batik classification studies reported substantially higher accuracy than the present work, but under markedly different data conditions and without quantifying the cost of deployment. The number of classes and the amount of data available per class differ considerably: Sastypratiwi et al. [14] distributed 1,520 images

This Journal is licensed under a Creative Commons Attribution 4.0 International License

over only four classes, Meranggi et al. [13] used approximately 600 images over five classes, and Syafi'i and Khomsah [17] expanded 572 original images into 5,148 through augmentation across 11 classes, whereas the present study distributed 644 valid images over 14 classes, that is, roughly 30–50 images per class. The influence of data volume is visible within the earlier work itself: Meranggi et al. [13] obtained 66.14% on the original dataset and 88.88% after correcting and improving the data without changing the architecture. The test accuracy of 43.08% reported here should therefore be interpreted primarily as a consequence of limited and visually similar training data rather than of the MobileNetV2 architecture. The comparison also identifies the dimension in which this study adds new evidence. The work of Sastypratiwi et al. [14] is the closest to the present study, because it likewise converted a MobileNet model into TensorFlow Lite format and deployed it in an Android application, yet it reported only classification accuracy and training time, without measuring the resulting model size or the on-device inference latency. None of the compared studies reported these two quantities. The present study measures both, showing a 77.47% reduction in model size and an 86.08% reduction in inference latency for the Float16 variant without any change in its predictions on the test subset. Two cautions attach to this comparison. First, the accuracy figures of the earlier studies are not directly comparable, because a four-class or five-class problem is substantially easier than the fourteen-class problem addressed here; the 100% accuracy reported by Sastypratiwi et al. [14] in particular is better read as an indication of limited test-set difficulty than as evidence of generalization. Second, the contribution claimed here is the measurement of deployment cost, not an improvement in classification accuracy, which remains the principal limitation of this work.

3.10 General Discussion

The findings indicate that MobileNetV2 can identify several batik motifs; however, the 43.08% accuracy reflects limited performance on previously unseen images. This limitation is mainly associated with the restricted dataset size and the strong visual resemblance among several batik motifs. Although accuracy is not yet high, optimization using TensorFlow Lite provides substantial gains in efficiency. These gains are variant-specific and should not be combined into a single claim: the Dynamic Range variant achieves the largest size reduction at 87.34%, the Int8 variant the largest speed-up at 13.82×, and the recommended Float16 variant a 77.47% size reduction together with a 7.19× speed-up.

Of all the variants, TFLite Float16 is recommended because it reproduced the predictions of the TensorFlow reference at a markedly lower size and latency. TFLite Int8 was faster but produced three fewer correct predictions, a difference that the present test subset is too small to confirm as real. Therefore, the optimizations in this study focused more on deployment efficiency than accuracy improvement. The model is promising for mobile devices, but still requires additional data, dataset quality improvements, and further architectural testing before use in a production environment.

4. CONCLUSION

This study applied MobileNetV2 transfer learning with partial fine-tuning to the classification of batik motifs and then optimized the trained model with TensorFlow Lite for deployment on resource-constrained devices. Evaluated on 644 valid images from 14 batik motif classes, the model reached a test accuracy of 43.08%, corresponding to 28 of the 65 test images, a macro F1-score of 41.65%, and a weighted F1-score of 40.39%, with distinctive motifs such as Megamendung and Kawung recognized far more reliably than visually similar motifs such as Bali and Tambal. In the deployment benchmark, whose internal reference accuracy of 46.15% for the unquantized model is explained in Section 3.8, the Float16 variant offered the best trade-off by reproducing the reference predictions while reducing model size from 19.04 MB to 4.29 MB and inference latency from 205.52 ms to 28.60 ms, whereas the Int8 variant achieved the fastest inference at 14.87 ms with an accuracy of 41.54%. Because the test subset contains only 65 images, the accuracy differences among the four model formats amount to at most three images and are not statistically significant; the variants are separated by their size and latency rather than by their predictive performance. These results confirm that the principal benefit of TensorFlow Lite optimization lies in reducing model size and inference latency rather than in improving classification accuracy. MobileNetV2 combined with TensorFlow Lite therefore provides a viable deployment pathway for on-device batik recognition, although the accuracy obtained here is not yet sufficient for production use. The comparison with previous studies also indicates a concrete target for further work: the dataset used here provides only about 46 images per class, whereas comparable studies that reached considerably higher accuracy worked with roughly 120 to 380 images per class, so the collection should be enlarged to at least 300 images per class through additional documentation rather than through augmentation alone, while retaining the 14 motif classes. Beyond dataset expansion, further work should analyse frequently confused motif pairs and evaluate alternative lightweight architectures in order to raise accuracy to a level suitable for production use.

REFERENCES

[1] M. Aprianti, D. A. Dewi, and Y. F. Furnamasari, "Kebudayaan Indonesia di Era Globalisasi Terhadap Identitas Nasional Indonesia," *Edumaspul: Jurnal Pendidikan*, vol. 6, no. 1, pp. 996–998, Mar. 2022, doi: 10.33487/edumaspul.v6i1.2294.

This Journal is licensed under a Creative Commons Attribution 4.0 International License

[2] A. D. W. Sumari, N. L. Aziza, and M. Hani'ah, "Preserving Indigenous Indonesian Batik Motif Using Machine Learning and Information Fusion," *JOIV : International Journal on Informatics Visualization*, vol. 9, no. 5, p. 2018, Sep. 2025, doi: 10.62527/joiv.9.5.3714.

[3] A. Dhillon and G. K. Verma, "Convolutional neural network: a review of models, methodologies and applications to object detection," *Progress in Artificial Intelligence*, vol. 9, no. 2, pp. 85–112, Jun. 2020, doi: 10.1007/s13748-019-00203-0.

[4] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, "Deep Learning for Computer Vision: A Brief Review," *Comput. Intell. Neurosci.*, vol. 2018, pp. 1–13, 2018, doi: 10.1155/2018/7068349.

[5] N. Sharma, V. Jain, and A. Mishra, "An Analysis Of Convolutional Neural Networks For Image Classification," *Procedia Comput. Sci.*, vol. 132, pp. 377–384, 2018, doi: 10.1016/j.procs.2018.05.198.

[6] S. Subramanian *et al.*, "Mobile Object Detection using TensorFlow Lite and Transfer Learning," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[7] F. Windhager *et al.*, "Visualization of Cultural Heritage Collection Data: State of the Art and Future Challenges," *IEEE Trans. Vis. Comput. Graph.*, vol. 25, no. 6, pp. 2311–2330, Jun. 2019, doi: 10.1109/TVCG.2018.2830759.

[8] D. Chen and P. Cheng, "A method to extract batik fabric pattern and elements," *Journal of the Textile Institute*, vol. 112, no. 7, 2021, doi: 10.1080/00405000.2020.1802885.

[9] Z.-P. Jiang, Y.-Y. Liu, Z.-E. Shao, and K.-W. Huang, "An Improved VGG16 Model for Pneumonia Image Classification," *Applied Sciences*, vol. 11, no. 23, p. 11185, Nov. 2021, doi: 10.3390/app112311185.

[10] Mehrunnisa, M. Leszczuk, D. Juszka, and Y. Zhang, "Improved Binary Classification of Underwater Images Using a Modified ResNet-18 Model," *Electronics (Basel).*, vol. 14, no. 15, p. 2954, Jul. 2025, doi: 10.3390/electronics14152954.

[11] K. Zhao *et al.*, "Efficient memory reuse methodology for CNN-based real-time image processing in mobile-embedded systems," *J. Real. Time. Image Process.*, vol. 20, no. 6, p. 118, Dec. 2023, doi: 10.1007/s11554-023-01375-8.

[12] O. Jouini, K. Sethom, A. Namoun, N. Aljohani, M. H. Alanazi, and M. N. Alanazi, "A Survey of Machine Learning in Edge Computing: Techniques, Frameworks, Applications, Issues, and Research Directions," *Technologies (Basel).*, vol. 12, no. 6, p. 81, Jun. 2024, doi: 10.3390/technologies12060081.

[13] D. G. T. Meranggi, N. Yudistira, and Y. A. Sari, "Batik Classification Using Convolutional Neural Network with Data Improvements," *International Journal on Informatics Visualization*, vol. 6, no. 1, 2022, doi: 10.30630/joiv.6.1.716.

[14] H. Sastypratiwi, H. Muhardi, and Y. Yulianti, "Batik Recognition and Classification Using Transfer Learning and MobileNet Approach," *JOIV : International Journal on Informatics Visualization*, vol. 8, no. 4, p. 2400, Dec. 2024, doi: 10.62527/joiv.8.4.2407.

[15] C. Öztürk, M. Taşyürek, and M. U. Türkdamar, "Transfer learning and fine-tuned transfer learning methods' effectiveness analyse in the CNN-based deep learning models," *Concurr. Comput.*, vol. 35, no. 4, Feb. 2023, doi: 10.1002/cpe.7542.

[16] N. C. A. Sallang, M. T. Islam, M. S. Islam, and H. Arshad, "A CNN-Based Smart Waste Management System Using TensorFlow Lite and LoRa-GPS Shield in Internet of Things Environment," *IEEE Access*, vol. 9, pp. 153560–153574, 2021, doi: 10.1109/ACCESS.2021.3128314.

[17] R. Syafi'i and S. Khomsah, "Classification of Indonesian Batik: A Comparative Study of CNN and VGG16 with Canny Edge Detection," in *2024 IEEE International Conference on Communication, Networks and Satellite (COMNETSAT)*, IEEE, Nov. 2024, pp. 355–363. doi: 10.1109/COMNETSAT63286.2024.10862216.

[18] A. Oktarino, Y. Desnita-Tasri, and A. Efendi, "Identification of Batik Motif Based Deep Learning-Convolutional Neural Network Approach," *The Journal of Ocean, Mechanical and Aerospace -science and engineering- (JOMAse)*, vol. 68, no. 3, pp. 139–147, Nov. 2024, doi: 10.36842/jomase.v68i3.381.

[19] C.-H. Tu, J.-H. Lee, Y.-M. Chan, and C.-S. Chen, "Pruning Depthwise Separable Convolutions for MobileNet Compression," in *2020 International Joint Conference on Neural Networks (IJCNN)*, IEEE, Jul. 2020, pp. 1–8. doi: 10.1109/IJCNN48605.2020.9207259.

[20] H.-I. Liu *et al.*, "Lightweight Deep Learning for Resource-Constrained Environments: A Survey," *ACM Comput. Surv.*, vol. 56, no. 10, pp. 1–42, Oct. 2024, doi: 10.1145/3657282.

[21] A. M. Youssef, B. Pradhan, A. Dikshit, and A. M. Mahdi, "Comparative study of convolutional neural network (CNN) and support vector machine (SVM) for flood susceptibility mapping: a case study at Ras Gharib, Red Sea, Egypt," *Geocarto Int.*, vol. 37, no. 26, pp. 11088–11115, Dec. 2022, doi: 10.1080/10106049.2022.2046866.

[22] R. B. (author). Anubhav Singh (author), *Mobile Deep Learning Projects 8 Project Guides to Help You Work Through End-to-End Neural Network Projects on Cross-Platform Apps. Anubhav Singh (author), Rimjhim Bhadani (author)*. Packt Publishing, 2020.

[23] B. Jacob *et al.*, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2018. doi: 10.1109/CVPR.2018.00286.

[24] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," in *7th International Conference on Learning Representations, ICLR 2019*, 2019.

[25] S. Richoz, A. Perez-Uribe, P. Birch, and D. Roggen, "Benchmarking deep classifiers on mobile devices for vision-based transportation recognition," in *Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers*, New York, NY, USA: ACM, Sep. 2019, pp. 803–807. doi: 10.1145/3341162.3344849.

[26] D. A. Anggoro, A. A. T. Marzuki, and W. Supriyanti, "Classification of Solo Batik patterns using deep learning convolutional neural networks algorithm," *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, vol. 22, no. 1, p. 232, Aug. 2023, doi: 10.12928/telkomnika.v22i1.24598.

This Journal is licensed under a Creative Commons Attribution 4.0 International License

[27] S. Deepa, J. L. Zeema, and S. Gokila, "Exploratory Architectures Analysis of Various Pre-trained Image Classification Models for Deep Learning," *Journal of Advances in Information Technology*, vol. 15, no. 1, 2024, doi: 10.12720/jait.15.1.66-78.

[28] M. Khatri, M. Sahoo, S. Sayyad, and J. Sayyad, "Adaptive and User-Friendly Framework for Image Classification with Transfer Learning Models," *Future Internet*, vol. 17, no. 8, p. 370, Aug. 2025, doi: 10.3390/fi17080370.

[29] V. Shah and N. Sajnani, "Multi-Class Image Classification using CNN and Tflite," *International Journal of Research in Engineering, Science and Management*, vol. 3, no. 11, 2020, doi: 10.47607/ijresm.2020.375.

[30] A. Ajit, K. Acharya, and A. Samanta, "A Review of Convolutional Neural Networks," in *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*, IEEE, Feb. 2020, pp. 1–5. doi: 10.1109/ic-ETITE47903.2020.049.

[31] F. A. Putra *et al.*, "Classification of Batik Authenticity Using Convolutional Neural Network Algorithm with Transfer Learning Method," in *2021 Sixth International Conference on Informatics and Computing (ICIC)*, IEEE, Nov. 2021, pp. 1–6. doi: 10.1109/ICIC54025.2021.9632937.

[32] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artif. Intell. Rev.*, vol. 53, no. 8, pp. 5455–5516, Dec. 2020, doi: 10.1007/s10462-020-09825-6.

[33] S. Cong and Y. Zhou, "A review of convolutional neural network architectures and their optimizations," *Artif. Intell. Rev.*, vol. 56, no. 3, pp. 1905–1969, Mar. 2023, doi: 10.1007/s10462-022-10213-5.

[34] J. Yang and G. Yang, "Modified Convolutional Neural Network Based on Dropout and the Stochastic Gradient Descent Optimizer," *Algorithms*, vol. 11, no. 3, p. 28, Mar. 2018, doi: 10.3390/a11030028.

[35] E. M. Dogo, O. J. Afolabi, N. I. Nwulu, B. Twala, and C. O. Aigbavboa, "A Comparative Analysis of Gradient Descent-Based Optimization Algorithms on Convolutional Neural Networks," in *2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS)*, IEEE, Dec. 2018, pp. 92–99. doi: 10.1109/CTEMS.2018.8769211.

[36] C. Shorten and T. M. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning," *J. Big Data*, vol. 6, no. 1, p. 60, Dec. 2019, doi: 10.1186/s40537-019-0197-0.

[37] D. Parra, D. Escobar Sanabria, and C. Camargo, "A Methodology and Open-Source Tools to Implement Convolutional Neural Networks Quantized with TensorFlow Lite on FPGAs," *Electronics (Basel).*, vol. 12, no. 20, p. 4367, Oct. 2023, doi: 10.3390/electronics12204367.

[38] I. L. Orasan, C. Seiculescu, and C. D. Caleanu, "Benchmarking TensorFlow Lite Quantization Algorithms for Deep Neural Networks," in *2022 IEEE 16th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, IEEE, May 2022, pp. 000221–000226. doi: 10.1109/SACI55618.2022.9919465.

[39] P. Kluska and M. Zięba, "Post-training Quantization Methods for Deep Learning Models," 2020, pp. 467–479. doi: 10.1007/978-3-030-41964-6_40.

[40] S. E. Adi and A. J. Casson, "Design and optimization of a TensorFlow Lite deep learning neural network for human activity recognition on a smartphone," in *2021 43rd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, IEEE, Nov. 2021, pp. 7028–7031. doi: 10.1109/EMBC46164.2021.9629549.

[41] P. D. Rosero-Montalvo, P. Tözün, and W. Hernandez, "Optimized CNN Architectures Benchmarking in Hardware-Constrained Edge Devices in IoT Environments," *IEEE Internet Things J.*, vol. 11, no. 11, pp. 20357–20366, Jun. 2024, doi: 10.1109/JIOT.2024.3369607.

This Journal is licensed under a Creative Commons Attribution 4.0 International License