Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

A Mobile-Centric Comparative Evaluation of Deep Learning Models for Real-Time Flower Recognition

Fatemeh Sabourinia

fatemehsabourinia@gmail.com

Independent Researcher https://orcid.org/0009-0009-7681-5170

Alireza Zaeri

Independent Researcher https://orcid.org/0009-0006-4836-4358

Vahid Abolghasemi

University of Essex https://orcid.org/0000-0002-2151-5180

Research Article

Keywords: Flower recognition, Deep learning, Mobile computing, Image classification, Real-time inference, Android deployment

Posted Date: November 11th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11681786/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

A Mobile-Centric Comparative Evaluation of Deep Learning Models for Real-Time Flower Recognition

Fatemeh Sabourinia[1*], Alireza Zaeri[1] and Vahid Abolghasemi[2]

[1]Colchester, United Kingdom.
[2]School of Computer Science and Electronic Engineering, University of Essex, Colchester, United Kingdom.

*Corresponding author(s). E-mail(s): fatemehsabourinia@gmail.com;
Contributing authors: alireza.zaeri1995@gmail.com;
v.abolghasemi@essex.ac.uk;

Abstract

Flower recognition has important applications in education, agriculture, and intelligent mobile systems. However, previous studies have primarily focused on classification accuracy, with less attention to the joint requirements of real-time mobile deployment. This study presents a mobile-centric flower recognition framework whose technical contribution lies in the dataset-specific adaptation and deployment-oriented evaluation of established learning architectures. Rather than applying an identical transfer-learning configuration to all convolutional models, architecture-specific classification heads and training strategies were developed for VGG16, NASNet-Mobile, and MobileNetV2 according to their learning behaviour on the Oxford 102 Flowers dataset. In addition, a hybrid MobileNetV2–XGBoost pipeline was constructed by combining deep features extracted from a pretrained MobileNetV2 backbone with an independently optimised XGBoost classifier. The four resulting pipelines were evaluated on a common data split using classification performance, inference time, training time, model size, architectural complexity, and deployment feasibility. On the 2,539-image test set, the adapted MobileNetV2 achieved approximately 96% accuracy and a weighted F1-score of 0.96, providing the most suitable overall trade-off for mobile use. It was subsequently converted to TensorFlow Lite, reducing its size from 27.74 MB to 2.53 MB, and integrated into an offline Android application. Repeated measurements on physical Android devices produced mean preprocessing-plus-inference latencies ranging from 55.57 to 68.11 ms. These results demonstrate that dataset-specific model adaptation and deployment-oriented selection can enable accurate and responsive flower recognition without cloud-based processing.

Keywords: Flower recognition, Deep learning, Mobile computing, Image classification, Real-time inference, Android deployment

1 Introduction

Flower recognition is an image classification task in which visual characteristics of flower images are analysed to identify their corresponding species. Automated flower recognition has practical applications in education, biodiversity monitoring, agriculture, horticulture, and citizen science. Students may use mobile applications to learn about flower species through direct observation, while researchers and conservationists can perform preliminary species identification during field surveys. Such applications are particularly useful in farms, forests, botanical gardens, and remote locations where reliable internet connectivity may not be available. Consequently, performing recognition directly on a mobile device can improve accessibility, reduce communication latency, and eliminate dependence on cloud-based services.

Recent advances in convolutional neural networks (CNNs) and transfer learning have considerably improved image-classification performance. However, deploying these models on mobile devices remains challenging because smartphones have limited processing capability, memory, storage capacity, and energy resources. Lightweight architectures such as MobileNetV2 were developed to balance recognition performance and computational efficiency for mobile and embedded vision applications [1]. Similarly, NASNet-Mobile employs convolutional structures identified through neural architecture search to support efficient recognition on resource-constrained devices [2]. Nevertheless, predictive accuracy alone does not establish whether a model is suitable for practical mobile deployment.

Recent flower-recognition studies have begun to consider deployment-related factors in addition to classification performance. Rajkomar and Pudaruth compared InceptionV3, MobileNetV2, and ResNet50V2 using classification metrics, prediction time, parameter count, and model size; they subsequently converted the selected MobileNetV2 model to TensorFlow Lite and integrated it into a mobile application [3]. However, their study did not report training time or controlled end-to-end preprocessing-plus-inference latency measurements on specified physical devices, nor did it investigate separately designed classification heads and training strategies for each architecture.

To address this gap, the present study develops and evaluates four dataset-adapted flower-recognition pipelines using the Oxford 102 Flowers dataset. Rather than applying an identical transfer-learning configuration to all CNN architectures, architecture-specific classification heads and training strategies were developed for VGG16, NASNet-Mobile, and MobileNetV2 according to their architectural characteristics and observed validation behaviour. In addition, a hybrid MobileNetV2–XGBoost pipeline was constructed by combining fixed-length deep features extracted using a pretrained MobileNetV2 backbone with a separately optimised XGBoost classifier [4]. The four pipelines were evaluated using a common dataset split, preprocessing procedure, evaluation protocol, and hardware environment. Model selection considered

weighted F1-score, training time, inference time, model size, architectural complexity, and deployment feasibility rather than classification accuracy alone.

The model providing the most suitable overall trade-off was converted to TensorFlow Lite and integrated into an offline Android application. Its practical deployment performance was subsequently evaluated through repeated preprocessing-and-inference measurements on physical Android devices. The main contributions of this work are summarised as follows:

- Dataset-specific and architecture-specific adaptation of VGG16, NASNet-Mobile, and MobileNetV2 through separately selected classification heads, regularisation configurations, and fine-tuning strategies.
- Development of a hybrid MobileNetV2–XGBoost pipeline that combines deep visual features from a pretrained lightweight CNN backbone with an independently optimised gradient-boosted classifier.
- A consistent comparison of the four adapted pipelines using predictive performance, training time, inference time, model size, architectural complexity, and mobile deployment feasibility.
- An end-to-end offline deployment workflow comprising model selection, TensorFlow Lite conversion, Android application integration, and physical-device benchmarking using 300 repeated measurements per device.

The remainder of this paper is organized as follows. Section 2 reviews related work in flower recognition and mobile deep learning applications. Section 3 presents the proposed methodology, including dataset, data preprocessing, and model development procedures. Section 4 discusses the experimental setup, comparative evaluation results, and mobile deployment. Finally, Section 5 concludes the paper and discusses future work.

2 Related Work

Flower recognition and plant classification have increasingly benefited from advances in deep learning and convolutional neural networks (CNNs), particularly in fine-grained image recognition tasks [5]. Early studies mainly relied on manual feature extraction methods combined with traditional machine learning classifiers; however, CNN-based approaches significantly improved automatic visual feature learning for flower classification applications. Building upon these advances, several deep learning architectures have been widely adopted for image classification tasks. Simonyan and Zisserman [6] demonstrated that deeper CNN architectures such as VGG16 can achieve strong visual representation learning through stacked convolutional layers. However, the large number of parameters and computational requirements of heavyweight CNNs can limit their suitability for mobile deployment. To address these limitations, Sandler *et al.* [1] proposed MobileNetV2 using inverted residual blocks and depthwise separable convolutions for computationally efficient mobile inference. Similarly, Zoph *et al.* [2] introduced NASNet-Mobile, which applied neural architecture search to automatically discover efficient architectures for resource-constrained

devices. These studies highlighted the growing importance of lightweight deep learning models for mobile image recognition systems.

As lightweight architectures became increasingly important for mobile applications, recent studies began exploring mobile deployment and on-device inference for deep learning systems. Guo [7] investigated mobile deep learning inference systems and demonstrated that cloud-based approaches can effectively perform computationally intensive image classification tasks. Although cloud-based systems can achieve high classification accuracy, they often depend on internet connectivity and external computational resources, which may increase latency and reduce usability in offline environments. To address these limitations, Lane *et al.* [8] and Ignatov *et al.* [9] examined the challenges of executing deep neural networks on resource-constrained mobile devices. Ignatov *et al.* also reviewed mobile deep-learning frameworks and discussed how TensorFlow Lite supports efficient on-device inference through optimized kernels, a compact runtime, and Android hardware acceleration [9]. Several recent studies have also investigated mobile-based plant and flower recognition systems using lightweight deep learning architectures. Reda *et al.* [10] developed a TensorFlow Lite-based mobile application for plant species and disease classification, demonstrating the feasibility of lightweight deployment on smartphones. Malik *et al.* [11] proposed a real-time medicinal plant identification system using deep learning techniques for mobile-assisted applications. Gavai *et al.* [12] applied MobileNet architectures for flower classification and reported reduced computational complexity compared with conventional CNN models. Burhan *et al.* [13] developed an IoT-enabled mobile flower recognition system that combined deep learning with smartphone-based image classification to identify flower species and monitor flower status, demonstrating the practical applicability of deep learning for intelligent mobile flower recognition. Although these studies demonstrated strong classification capabilities and mobile deployment feasibility, limited attention has been given to the trade-off between classification accuracy, inference speed, model size, and deployment efficiency under practical mobile constraints. Consequently, comprehensive comparative evaluation of heavyweight and lightweight models for real-time on-device flower recognition remains insufficiently explored.

To address these limitations, this study presents a mobile-centric comparative framework for flower recognition using the Oxford 102 Flowers dataset. Unlike studies that apply uniform transfer-learning configurations or select a model primarily according to classification accuracy, the present work adapts each candidate pipeline to the target dataset. Architecture-specific classification heads and training strategies are developed for VGG16, NASNet-Mobile, and MobileNetV2, while a hybrid MobileNetV2–XGBoost pipeline combines deep features extracted by a lightweight pretrained backbone with an independently optimised gradient-boosted classifier. The resulting pipelines are evaluated under a common experimental protocol using predictive performance, training time, inference time, model size, architectural complexity, and deployment feasibility. Based on this joint evaluation, the model providing the most suitable overall trade-off is selected, converted to TensorFlow Lite, integrated into an offline Android application, and benchmarked repeatedly on physical mobile

devices. Therefore, the contribution of this study lies in the dataset-specific adaptation, consistent deployment-oriented comparison, and physical-device validation of the candidate pipelines rather than in proposing a new backbone architecture.

3 Proposed Methodology

This section presents the proposed methodology used to develop the flower recognition framework for mobile image classification. The methodology includes dataset, image preprocessing, and model development using both deep learning and machine learning approaches. In addition, the section describes the training strategies and optimization procedures applied throughout the study to support reliable and reproducible model training under mobile-oriented computational constraints. Fig. 1 provides an overview of how these stages lead to model evaluation, final model selection, and mobile deployment.

3.1 Dataset

The experiments in this study were conducted using the Oxford 102 Flowers dataset, which contains 8,189 JPEG images distributed across 102 flower categories commonly found in the United Kingdom, with each category containing between 40 and 258 images [14]. The original images have varying spatial resolutions and aspect ratios and exhibit significant variations in color, shape, scale, illumination, and background complexity, making the dataset suitable for fine-grained flower classification tasks. Furthermore, the visual similarity among several flower categories presents a challenging classification problem for both traditional machine learning and deep learning

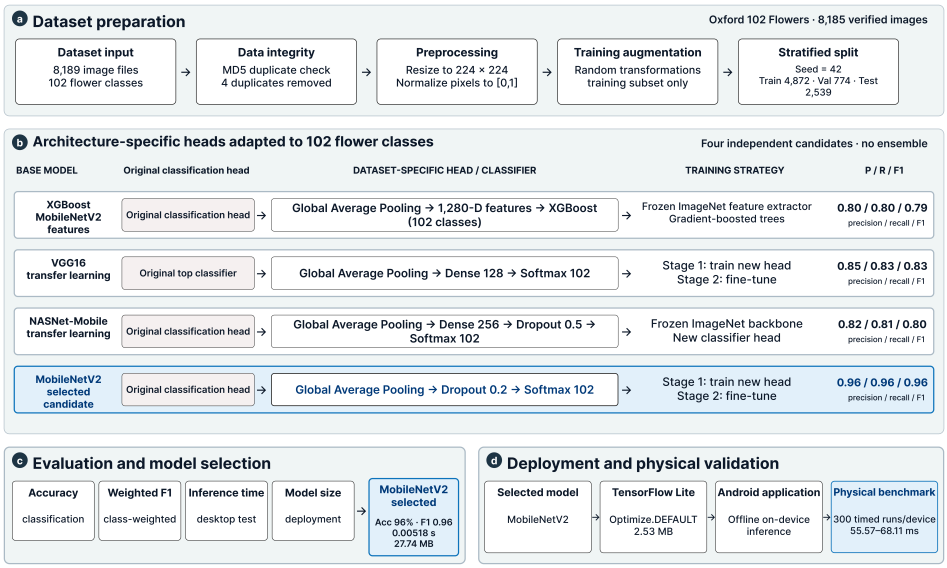


Fig. 1 Flower recognition workflow from data preparation to mobile deployment

models. To ensure balanced data distribution across all classes, a stratified split was applied to divide the dataset into training, validation, and testing subsets with proportions of 60%, 10%, and 30%, respectively.

3.2 Data Preprocessing

Before model training, dataset integrity was verified by checking all 8,189 images for missing or corrupted files. No missing or unreadable images were detected. Exact duplicate detection using MD5 hashes identified four duplicate pairs. One image from each pair was excluded, leaving 8,185 unique images for model development and evaluation.

The class distribution of the dataset was analyzed before splitting the images into training, validation, and testing subsets. The number of images in each flower category was calculated and visualized using a bar chart to identify potential distribution inconsistencies across classes. The class distribution before dataset splitting is shown in Fig. 2.

The analysis showed that the dataset was not perfectly balanced, as some classes contained significantly more images than others. Certain flower categories included more than 200 images, while several classes contained fewer than 100 samples. After dataset verification and imbalance analysis, the images were divided using a stratified splitting strategy with ratios of 60% for training, 10% for validation, and 30% for testing. The splitting process preserved the original class distribution across all subsets, and a fixed random seed was used to ensure reproducibility. The final dataset contained 4,872 training images, 774 validation images, and 2,539 testing images. To validate the effectiveness of the stratified splitting process, the image distributions across all subsets were analyzed after splitting. The results confirmed that all flower categories were represented proportionally according to the selected split ratios, ensuring consistent class representation for reliable model evaluation.

All images were resized to 224×224 pixels before training to maintain a consistent input size across all models. Pixel values were normalized using a rescaling factor of $1/255$, converting the original image intensities to the range of $[0, 1]$. This normalization step helped stabilize gradient updates and improve convergence during training. To improve model robustness and reduce overfitting, data augmentation was applied

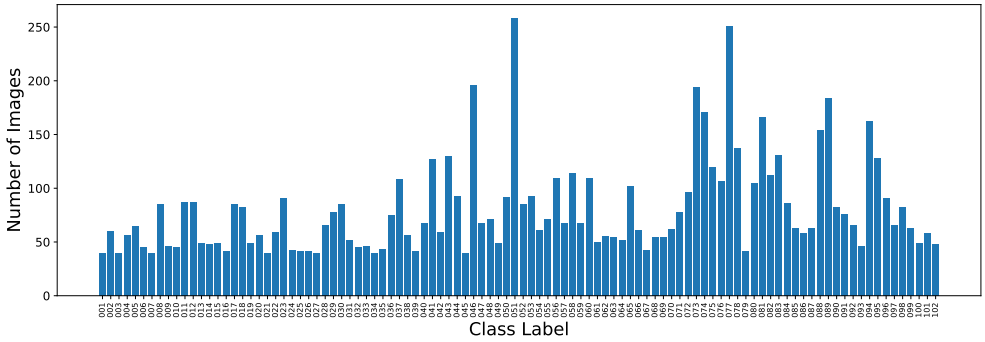


Fig. 2 Oxford 102 Flowers class distribution before splitting



Fig. 3 Augmentation effects on images from the Oxford 102 Flowers training subset

exclusively to the training subset using TensorFlow ImageDataGenerator utilities [15]. The augmentation configuration included random rotations of up to 15°, width and height shifts of 0.1, shear transformations of 0.1, zoom augmentation of 0.2, horizontal flipping, and brightness adjustments within the range of $[0.8, 1.2]$. The effects of these operations on images selected from the Oxford 102 Flowers training subset are illustrated in Fig. 3.

In contrast, the validation and testing subsets were processed using normalization only to avoid data leakage during evaluation. A unified preprocessing pipeline was implemented for all experiments using reusable data generator functions. The generators were configured with a batch size of 32 and categorical class encoding for multi-class classification. Training samples were shuffled during loading, whereas validation and testing samples were processed without shuffling to ensure deterministic evaluation. In addition, a fixed random seed value of 42 was applied during preprocessing and dataset generation to maintain reproducibility across different experimental runs. To reduce the influence of class imbalance, balanced class weights were computed from the training subset using the Scikit-learn `compute_class_weight` function [16]. The resulting weights were incorporated during model training so that minority classes contributed more strongly to the optimization process, reducing bias toward categories with larger numbers of samples.

3.3 Model Development

Four candidate model pipelines with different architectural and computational characteristics were developed and evaluated: a hybrid MobileNetV2–XGBoost pipeline, VGG16, NASNet-Mobile, and MobileNetV2. The hybrid pipeline was included to investigate whether compact deep features extracted by a lightweight pretrained backbone could support effective classification using a non-neural gradient-boosted

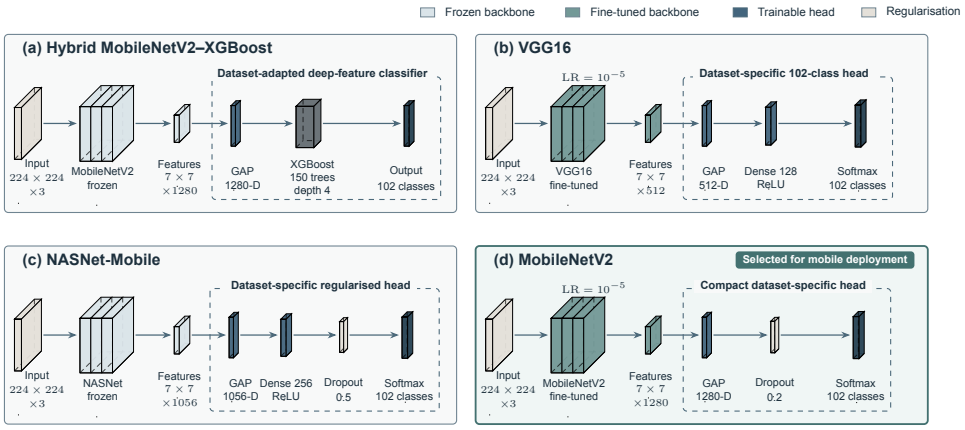


Fig. 4 Architectures of the four dataset-adapted flower recognition pipelines

classifier [17]. VGG16 represented a conventional deep convolutional architecture with strong hierarchical feature-extraction capacity [18]. NASNet-Mobile represented a compact architecture generated through neural architecture search [19], whereas MobileNetV2 represented a lightweight architecture based on inverted residual blocks and depthwise separable convolutions [20]. Together, these pipelines provided different combinations of feature extraction, classification strategy, model complexity, and mobile deployment suitability.

The candidate pipelines were evaluated using the same dataset partitions, preprocessing procedure, evaluation metrics, and hardware environment. However, an identical classifier head and training configuration were not imposed on all architectures. The hybrid MobileNetV2–XGBoost pipeline used fixed-length features extracted by a frozen MobileNetV2 backbone together with a separately optimised XGBoost classifier. For the CNN pipelines, the original ImageNet classification layers were removed and replaced with task-specific heads adapted to the 102 flower categories. Classification-head depth, dense-layer size, dropout regularisation, and backbone freezing or fine-tuning were selected separately according to each model's architectural characteristics and observed validation behaviour. The resulting dataset- and architecture-adapted pipelines, including their backbone configurations and task-specific classification stages, are illustrated in Fig. 4.

3.3.1 Hybrid MobileNetV2–XGBoost

The first candidate was implemented as a hybrid MobileNetV2–XGBoost pipeline rather than as a standalone XGBoost classifier. Since gradient-boosted trees cannot learn spatial representations directly from raw images, a pretrained MobileNetV2 backbone was used as a fixed deep-feature extractor [21]. The backbone was initialised using ImageNet weights with its original classification layers removed. Each $224 \times 224 \times 3$ input image produced a $7 \times 7 \times 1280$ convolutional feature tensor. Global average pooling was then applied to obtain a compact 1,280-dimensional feature vector for

classification. Feature matrices were generated separately for the training, validation, and testing subsets to preserve the established data partitions.

The connection between the frozen feature extractor and the gradient-boosted classifier can be summarised as

$$\begin{aligned} \mathbf{g}(x) &= \mathrm{GAP}(F_\theta(x)) \in \mathbb{R}^{1280}, \\ \hat{y} &= \underset{c\in\{1,\ldots,102\}}{\arg\max}\; H_{\psi,c}(\mathbf{g}(x)), \end{aligned} \tag{1}$$

where $F_\theta(x)$ denotes the convolutional feature tensor produced by the fixed MobileNetV2 backbone, $\mathbf{g}(x)$ is its global-average-pooled representation, and $H_{\psi,c}$ denotes the probability assigned to class c by the XGBoost classifier. This configuration combines transferable visual representations from a lightweight CNN with the non-neural decision boundaries learned by gradient-boosted trees.

The XGBoost component was configured for 102-class classification using the `multi:softprob` objective and the `mlogloss` evaluation metric. Hyperparameter selection was performed on the extracted training features using `RandomizedSearchCV` with two-fold cross-validation and accuracy-based scoring [22]. The search considered 100 or 150 estimators, maximum depths of 4 or 6, and learning rates of 0.05 or 0.1. The selected configuration used 150 estimators, a maximum depth of 4, and a learning rate of 0.1. A fixed random seed of 42 was maintained throughout feature extraction and classifier optimisation to support reproducibility.

3.3.2 VGG16

The VGG16 backbone was initialised using ImageNet pretrained weights [23], while its original fully connected classification layers were removed using `include_top=False`. Input images were provided at a resolution of $224 \times 224 \times 3$. During the initial training stage, all pretrained convolutional layers remained frozen, allowing the newly introduced classification layers to adapt to the flower dataset without modifying the transferred ImageNet representations [24].

Rather than retaining the original VGG16 classifier or applying a shared classification head across all CNN candidates, a task-specific head was developed for the Oxford 102 Flowers dataset. The final head comprised a `GlobalAveragePooling2D` layer, a fully connected layer with 128 neurons and ReLU activation, and a 102-class Softmax output. Global average pooling converted the $7 \times 7 \times 512$ backbone output into a compact 512-dimensional representation while avoiding the large parameter increase associated with flattening [25]. The resulting dataset-specific classification stage can be expressed as

$$\begin{aligned} \mathbf{g}(x) &= \mathrm{GAP}(F_\theta(x)) \in \mathbb{R}^{512}, \\ \mathbf{h}(x) &= \mathrm{ReLU}(\mathbf{W}_1\mathbf{g}(x) + \mathbf{b}_1) \in \mathbb{R}^{128}, \\ \hat{\mathbf{y}}(x) &= \mathrm{softmax}(\mathbf{W}_2\mathbf{h}(x) + \mathbf{b}_2) \in \mathbb{R}^{102}, \end{aligned} \tag{2}$$

where $F_\theta(x)$ denotes the feature tensor produced by the VGG16 backbone, $\mathbf{g}(x)$ is the global-average-pooled representation, $\mathbf{h}(x)$ is the task-specific hidden representation, and $\hat{\mathbf{y}}(x)$ contains the predicted probabilities for the 102 flower categories.

This formulation directly represents the classification head introduced for the target dataset rather than the standard VGG16 output structure.

The model was optimised using Adam and categorical cross-entropy loss [26]. Initial head training was configured for a maximum of 35 epochs. Early stopping monitored validation loss with a patience of five epochs and restored the best-performing weights, while `ReduceLROnPlateau` reduced the learning rate by a factor of 0.5 after three epochs without validation-loss improvement [27]. Following the initial stage, the complete VGG16 backbone was unfrozen and fine-tuned using a reduced learning rate of 1×10^{-5} for a maximum of 200 epochs. Class weighting and the same validation-based callbacks were retained during fine-tuning. This two-stage strategy was selected because updating the pretrained backbone improved its adaptation to the dataset-specific colour, texture, and shape characteristics of the 102 flower categories.

3.3.3 NASNet-Mobile

NASNet-Mobile was initialised using ImageNet pretrained weights with its original classification layers removed through `include_top=False`. Input images were provided at a resolution of $224 \times 224 \times 3$. During the initial training stage, the pretrained backbone remained frozen so that its transferred visual representations were preserved while the newly introduced classification layers were adapted to the Oxford 102 Flowers dataset.

Rather than applying the same classifier used for the other CNN candidates, a dataset-specific regularised head was developed for NASNet-Mobile. The $7 \times 7 \times 1056$ feature tensor produced by the backbone was transformed using global average pooling, followed by a dense layer with 256 neurons and ReLU activation. A dropout layer with a rate of 0.5 was then applied to reduce co-adaptation and improve generalisation [28]. The classification stage concluded with a 102-class Softmax output. The adapted head can be expressed as

$$\begin{aligned} \mathbf{g}(x) &= \mathrm{GAP}(F_\theta(x)) \in \mathbb{R}^{1056}, \\ \mathbf{h}(x) &= \mathcal{D}_{0.5}(\mathrm{ReLU}(\mathbf{W}_1\mathbf{g}(x) + \mathbf{b}_1)) \in \mathbb{R}^{256}, \\ \hat{\mathbf{y}}(x) &= \mathrm{softmax}(\mathbf{W}_2\mathbf{h}(x) + \mathbf{b}_2) \in \mathbb{R}^{102}, \end{aligned} \tag{3}$$

where $F_\theta(x)$ denotes the feature tensor produced by the NASNet-Mobile backbone, $\mathbf{g}(x)$ is the pooled 1,056-dimensional representation, $\mathcal{D}_{0.5}$ denotes dropout with a rate of 0.5, $\mathbf{h}(x)$ is the resulting task-specific representation, and $\hat{\mathbf{y}}(x)$ contains the predicted probabilities for the 102 flower categories.

The model was optimised using Adam and categorical cross-entropy loss. Training was configured for a maximum of 200 epochs. Early stopping monitored validation loss with a patience of five epochs and restored the best-performing weights. In addition, `ReduceLROnPlateau` reduced the learning rate by a factor of 0.5 after three consecutive epochs without validation-loss improvement.

Following the initial training stage, multiple fine-tuning configurations were investigated by selectively unfreezing deeper backbone layers and, separately, the complete NASNet-Mobile backbone. These configurations did not produce consistent validation improvements and occasionally reduced generalisation performance. Consequently,

the final NASNet-Mobile pipeline retained the frozen pretrained backbone together with the dataset-specific regularised head. This model-specific decision avoided unnecessary backbone updates while preserving the configuration that provided the most stable validation behaviour for the target dataset.

3.3.4 MobileNetV2

The final MobileNetV2 pipeline used a compact dataset-specific classification head designed to preserve the lightweight characteristics of the backbone. Unlike the deeper heads used for VGG16 and NASNet-Mobile, no intermediate dense layer was introduced. The $7 \times 7 \times 1280$ feature tensor was transformed using global average pooling, followed by dropout with a rate of 0.2 and a 102-class Softmax output. This configuration limited the number of additional trainable parameters while providing a direct mapping from the extracted visual features to the target flower categories. The compact head can be expressed as

$$\begin{aligned} \mathbf{g}(x) &= \mathrm{GAP}(F_\theta(x)) \in \mathbb{R}^{1280}, \\ \tilde{\mathbf{g}}(x) &= \mathcal{D}_{0.2}(\mathbf{g}(x)), \\ \hat{\mathbf{y}}(x) &= \mathrm{softmax}(\mathbf{W}\tilde{\mathbf{g}}(x) + \mathbf{b}) \in \mathbb{R}^{102}, \end{aligned} \tag{4}$$

where $F_\theta(x)$ denotes the feature tensor produced by the MobileNetV2 backbone, $\mathbf{g}(x)$ is the global-average-pooled representation, $\mathcal{D}_{0.2}$ denotes dropout with a rate of 0.2, and $\hat{\mathbf{y}}(x)$ contains the predicted probabilities for the 102 flower categories.

The classification head was initially trained for a maximum of 20 epochs while the pretrained backbone remained frozen. The model was optimised using Adam and categorical cross-entropy loss, with class weighting and early stopping applied during training. The complete backbone was subsequently unfrozen and fine-tuned end to end using a reduced learning rate of 1×10^{-5} for a maximum of 200 epochs. Early stopping retained the best validation checkpoint, while `ReduceLROnPlateau` reduced the learning rate by a factor of 0.5 after three consecutive epochs without validation-loss improvement. This model-specific two-stage strategy improved adaptation to the target dataset while retaining a compact classification structure suitable for mobile deployment.

4 Experimental Results

4.1 Experimental Setup and Performance Metrics

A comprehensive experimental study was conducted to evaluate and compare the flower classification models under a common experimental protocol. To ensure a fair comparison, all models used identical dataset splits, preprocessing procedures, evaluation metrics, and hardware conditions. However, the classification heads and training strategies were adapted separately according to the behaviour of each model. The experiments were carried out on a system equipped with an Apple M4 processor and 16 GB RAM using CPU-based computation. Model development and training were implemented in Python using TensorFlow/Keras and Scikit-learn libraries within a Jupyter Notebook environment, where Scikit-learn was additionally employed for the

XGBoost classifier within the hybrid MobileNetV2–XGBoost workflow and evaluation procedures. TensorFlow version 2.19.0 and Scikit-learn version 1.5.1 were utilized during model development and training. A separate TensorFlow 2.14.0 environment was additionally used during the TensorFlow Lite conversion stage to support stable TensorFlow Lite model generation for mobile deployment. Both classification and computational performance metrics were utilized to assess model effectiveness throughout the experimental evaluation. The evaluation metrics employed in this study are widely used in machine learning and image classification research to provide a comprehensive assessment of model performance, including accuracy, precision, recall, and F1-score. In the following equations, TP, TN, FP, and FN denote true positives, true negatives, false positives, and false negatives, respectively [29].

Classification accuracy was used to measure the proportion of correctly classified samples among all test instances and is defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

Precision was employed to evaluate classification reliability across flower categories by measuring the proportion of correctly predicted positive samples among all predicted positive samples:

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

Recall was additionally used to measure model sensitivity by evaluating the proportion of correctly predicted positive samples among all actual positive samples:

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

To provide a balanced evaluation under class imbalance conditions, the F1-score was also computed as the harmonic mean of precision and recall:

$$F_1\text{-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{8}$$

Since the Oxford 102 Flowers dataset contains class imbalance, the weighted F1-score was additionally employed to ensure that classes with larger numbers of samples contributed proportionally to the final evaluation metric:

$$\text{Weighted } F_1\text{-score} = \frac{\sum_{i=1}^{N} w_i \times F_{1,i}}{\sum_{i=1}^{N} w_i} \tag{9}$$

where $F_{1,i}$ represents the F1-score of class i, w_i denotes the number of samples belonging to class i, and N is the total number of classes.

Inference time was measured as the average time required by a trained model to generate a prediction for one test image:

$$T_{\text{inf}} = \frac{T_{\text{end}} - T_{\text{start}}}{N_{\text{test}}} \tag{10}$$

where T_{inf} denotes the inference time per image, T_{start} and T_{end} represent the prediction start and end times, and N_{test} is the total number of test images.

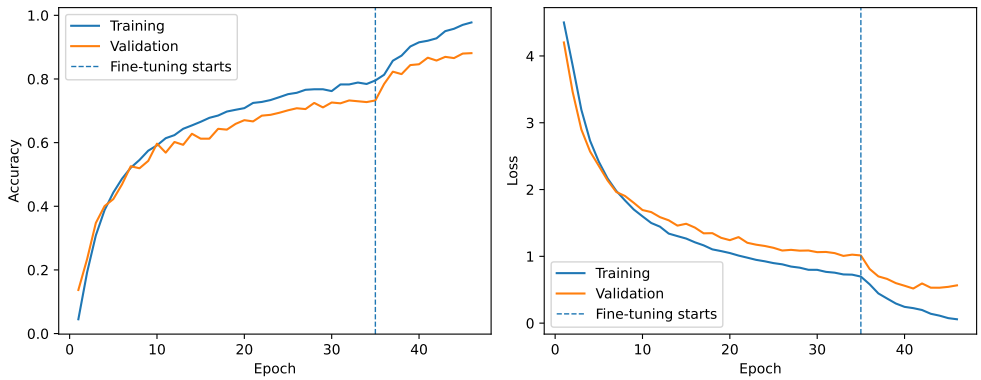


Fig. 5 VGG16 learning curves during base training and fine-tuning

4.2 Model Evaluation and Comparative Results

For models where fine-tuning improved performance, both the initial training and fine-tuning stages are presented. All deep learning models used the same dataset partitions and evaluation protocol, whereas their classification heads, training schedules, and fine-tuning strategies were configured separately for each architecture. Early stopping and learning-rate scheduling were applied to reduce overfitting and support stable convergence. In addition to the initial training phase, fine-tuning experiments were performed to further improve model performance. While fine-tuning resulted in noticeable performance gains for some models, it provided limited benefits or reduced generalization performance for others. Consequently, the best-performing version of each model, obtained either after the initial training phase or after fine-tuning, was retained for the final comparative evaluation.

4.2.1 Training Behaviour Analysis

The hybrid MobileNetV2–XGBoost pipeline does not follow the epoch-based neural-network training process used by the CNN classifiers. Its MobileNetV2 backbone was employed as a fixed feature extractor, while the XGBoost component was trained as a gradient-boosted tree classifier. Therefore, conventional epoch-wise accuracy and loss curves were not applicable to this pipeline. Training behaviour analysis was consequently performed only for VGG16, NASNet-Mobile, and MobileNetV2 using their corresponding accuracy and loss curves.

As illustrated in Fig. 5, the VGG16 model demonstrated a steady increase in both training and validation accuracy throughout the training process, accompanied by a consistent reduction in loss values.

The initial training phase converged after 35 epochs, while the subsequent fine-tuning stage terminated after 11 epochs due to the EarlyStopping criterion. Following fine-tuning, additional performance improvements were observed, indicating that updating the pretrained convolutional layers enabled the model to better adapt to the flower classification task. Although a slight gap between training and validation

performance emerged during later epochs, the validation curves remained relatively stable throughout training, indicating good generalization capability.

The learning curves of NASNet-Mobile shown in Fig. 6 exhibited stable convergence behaviour, with both training and validation accuracy increasing consistently while loss values decreased progressively throughout training.

The relatively small gap between the training and validation curves suggests reasonable generalization to unseen data. Since multiple fine-tuning strategies failed to provide consistent performance improvements during preliminary experiments, the model was evaluated using the best-performing version obtained from the initial training stage. Training was automatically terminated after 60 epochs by the EarlyStopping mechanism, indicating that further optimization no longer improved validation performance.

As shown in Fig. 7, MobileNetV2 demonstrated rapid convergence during the initial training stage and achieved further improvements after fine-tuning.

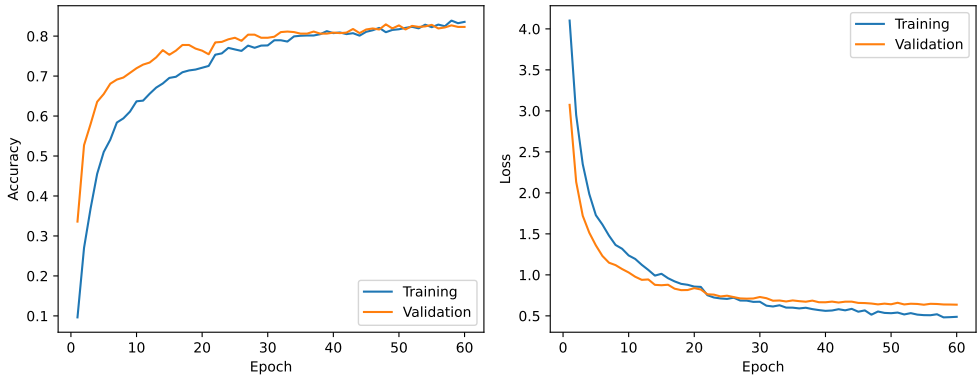


Fig. 6 NASNet-Mobile training and validation curves

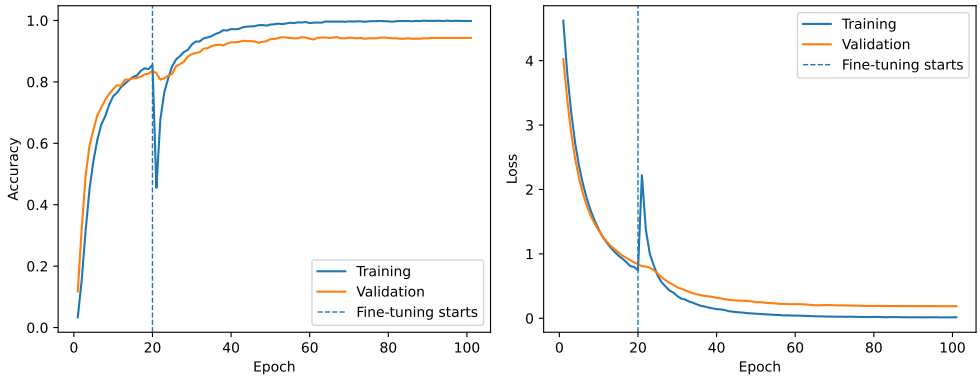


Fig. 7 MobileNetV2 learning curves during base training and fine-tuning

The initial training phase converged after 20 epochs, whereas the fine-tuning stage continued until epoch 81 before being terminated by EarlyStopping. A brief fluctuation was observed immediately after the transition to fine-tuning, which is expected when previously frozen layers become trainable and their weights begin to update. Subsequently, both the accuracy and loss curves stabilized, while validation performance remained consistently high. The validation curves remained stable throughout training, while both accuracy and loss continued to improve after fine-tuning, indicating strong generalization performance and effective feature adaptation.

4.2.2 Classification Performance

The classification performance of the evaluated models was assessed using weighted precision, recall, and F1-score metrics computed on the test set. Weighted averaging was selected to account for the class imbalance present in the Oxford 102 Flowers dataset by weighting each class according to its support. Fig. 8 summarizes the weighted classification metrics achieved by all evaluated models.

As shown in Fig. 8, MobileNetV2 achieved precision, recall, and F1-score values of 0.96 across all three metrics. The strong agreement among these values indicates balanced classification behaviour and a high degree of consistency in predicting flower categories throughout the evaluation dataset. The results further suggest that the learned feature representations remained effective across a wide range of flower classes. Similarly, VGG16 demonstrated stable classification performance, achieving a precision of 0.85 together with recall and F1-score values of 0.83. The close correspondence between the three metrics indicates reliable predictive behaviour and consistent performance on previously unseen flower images. NASNet-Mobile also demonstrated consistent classification behaviour, producing precision, recall, and F1-score values of 0.82, 0.81, and 0.80, respectively. The relatively small variation among these metrics suggests stable prediction quality across the evaluation dataset. Using the same weighted evaluation metrics, the hybrid MobileNetV2–XGBoost pipeline achieved precision and recall values of 0.80 together with an F1-score of 0.79. These results indicate

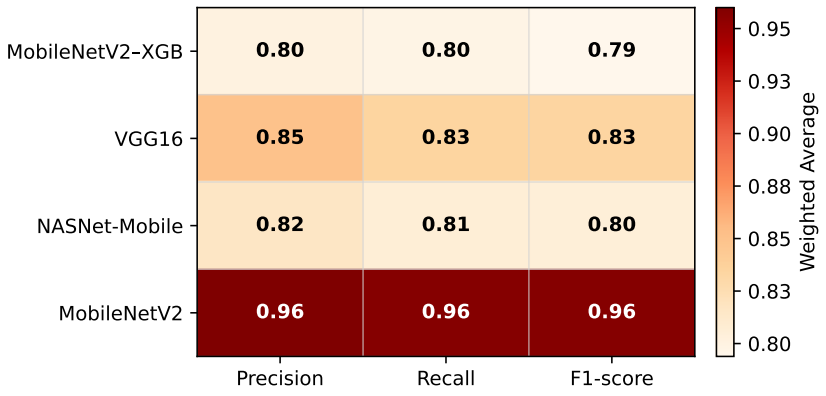


Fig. 8 Weighted classification metrics across the evaluated models

that the deep feature representations extracted by MobileNetV2 provided useful discriminative information for classification. Overall, MobileNetV2 achieved the strongest classification performance, with weighted precision, recall, and F1-score values of 0.96. Together with its stable learning behaviour and the improvements observed after fine-tuning, these results identified MobileNetV2 as the leading candidate based on predictive performance. However, no final model was selected at this stage, as computational efficiency, model size, training cost, and architectural complexity were also considered in the subsequent analysis.

4.2.3 Computational and Architectural Analysis

In addition to classification performance, computational and architectural characteristics were examined to provide a more comprehensive evaluation of the investigated models. While classification performance remains an important evaluation criterion in flower recognition studies, practical deployment scenarios also require consideration of inference latency, storage requirements, training cost, and architectural complexity. The comparative results for the evaluated models are presented in Table 1.

The results reveal clear differences in the relationship between recognition performance and computational cost across the evaluated approaches. In particular, the models exhibited markedly different computational characteristics despite being evaluated on the same flower recognition task. Higher recognition performance was not always associated with larger architectures or greater computational requirements, suggesting that architectural design plays an important role in determining both effectiveness and efficiency.

A comparison of the computational metrics reveals substantial differences among the evaluated models. VGG16 had the largest model size, highest inference latency, and greatest training cost, whereas NASNet-Mobile and MobileNetV2 required considerably less storage and provided faster inference. MobileNetV2 achieved the highest weighted F1-score while maintaining a practical inference time and model size. The XGBoost classifier within the hybrid MobileNetV2–XGBoost pipeline had the shortest reported classifier inference and training times; however, these measurements exclude the preceding MobileNetV2 feature-extraction stage and therefore do not represent the computational cost of the complete recognition pipeline. The architectural results in Table 2 help explain these differences.

VGG16 uses a conventional stack of convolutional layers and had the highest parameter count. MobileNetV2 employs inverted residual blocks and depthwise

Table 1 Model performance and computational cost

Model	Weighted F1-score	Inference time (s)	Model size (MB)	Training time (min)
MobileNetV2–XGBoost	0.79	0.00002	13.52	8.05
VGG16	0.83	0.05570	169.40	360.70
NASNet-Mobile	0.80	0.00855	22.30	47.02
MobileNetV2	0.96	0.00518	27.74	197.36

Table 2 Model complexity comparison

Model	Params (M)	Layers
XGBoost	N/A	N/A
VGG16	14.79	22
NASNet-Mobile	4.57	773
MobileNetV2	2.39	157

separable convolutions, resulting in the lowest parameter count among the evaluated CNNs. NASNet-Mobile uses a cell-based architecture obtained through neural architecture search and had the highest reported layer count despite containing substantially fewer parameters than VGG16. The hybrid MobileNetV2–XGBoost pipeline differs from the CNN pipelines because its XGBoost component was trained on 1,280-dimensional features extracted by a pretrained MobileNetV2 rather than directly on raw images.

Although no new backbone architecture was proposed, each pretrained CNN was adapted specifically to the target task. During model development, multiple classification-head configurations with different numbers, types, and sizes of layers were experimentally evaluated. Increasing head complexity did not consistently improve training and validation performance, whereas overly simplified configurations provided insufficient capacity for learning task-specific representations. Therefore, a model-specific classification head and training strategy were selected for each CNN based on its observed training and validation behaviour. The final VGG16 head consisted of Global Average Pooling, a 128-unit dense layer, and a 102-class softmax output. NASNet-Mobile used Global Average Pooling, a 256-unit dense layer, 0.5 dropout, and a 102-class softmax output, whereas MobileNetV2 used Global Average Pooling, 0.2 dropout, and a 102-class softmax output. Fine-tuning further improved VGG16 and MobileNetV2, while the frozen-backbone NASNet-Mobile configuration was retained because fine-tuning produced no consistent improvement. Thus, the contribution lies in the systematic, model-specific adaptation of established pretrained architectures to the target dataset and their comparative evaluation under a common, mobile-oriented experimental framework.

4.2.4 Final Model Selection

The four candidate approaches were developed, trained, and evaluated independently and were not combined into an ensemble. For the CNN models, the established ImageNet-pretrained backbones were retained. At the same time, the classification heads and training strategies were adapted separately based on each model's behaviour on the Oxford 102 Flowers dataset. The hybrid MobileNetV2–XGBoost pipeline combined features extracted by a pretrained MobileNetV2 with a separately trained XGBoost classifier. Final model selection was performed offline after completing the experimental comparison; the mobile application does not run the four models or select between them automatically. The results showed clear trade-offs between recognition performance and computational requirements. MobileNetV2 achieved the highest

weighted F1-score of 0.96, compared with 0.83 for VGG16, 0.80 for NASNet-Mobile, and 0.79 for the hybrid MobileNetV2–XGBoost pipeline. VGG16 required the greatest training time, storage capacity, and inference time. NASNet-Mobile was more computationally efficient than VGG16 but achieved lower recognition performance than MobileNetV2. The XGBoost classifier within the hybrid MobileNetV2–XGBoost pipeline had the shortest reported classifier inference and training times; however, these measurements excluded the preceding MobileNetV2 feature-extraction stage. Therefore, its reported computational advantages did not represent the cost of the complete recognition pipeline.

Among the model-specific configurations evaluated, the adapted MobileNetV2 provided the most effective balance between recognition performance, inference time, model size, and suitability for mobile deployment. Its final classification head consisted of Global Average Pooling, a dropout layer with a rate of 0.2, and a 102-class softmax output. After training the classification head, the MobileNetV2 backbone was fine-tuned using a lower learning rate, which improved its performance on the target dataset. Based on these findings, this adapted MobileNetV2 was selected as the only model for TensorFlow Lite conversion and integration into the proposed mobile flower recognition application.

4.3 Mobile Deployment

Following the final model selection process, MobileNetV2 was deployed within a mobile flower recognition application designed for on-device inference. To facilitate efficient execution on resource-constrained mobile devices, the selected MobileNetV2 model was converted to TensorFlow Lite (TFLite) format and optimised using TensorFlow Lite's default optimisation framework. TensorFlow Lite is a lightweight machine learning framework specifically designed for deploying deep learning models on mobile and embedded devices, providing efficient inference with reduced computational and memory requirements [30]. The resulting deployment model,

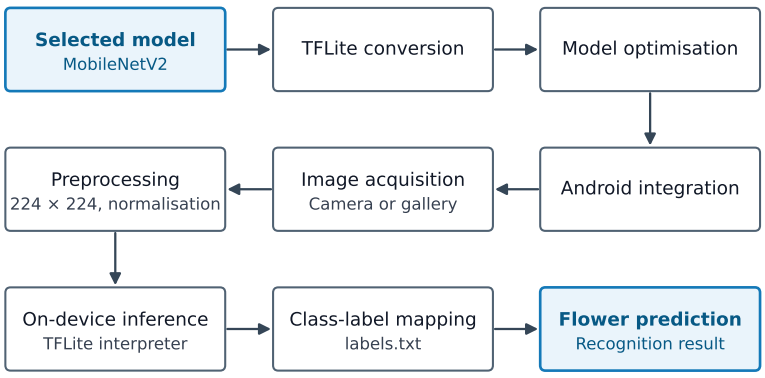
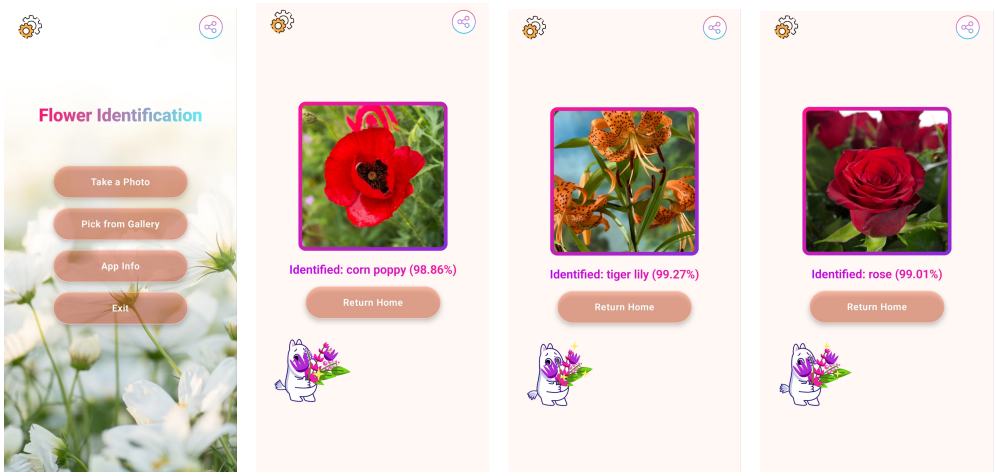


Fig. 9 Mobile deployment pipeline

`mobilenetv2_optimized.tflite`, was stored together with a `labels.txt` file containing the flower class labels required for prediction. The overall mobile deployment workflow is illustrated in Fig. 9, beginning with the selected MobileNetV2 model, followed by TFLite conversion and optimisation, and concluding with on-device flower prediction within the Android application.

To ensure that model optimisation did not compromise classification performance, the converted TFLite model was evaluated using the independent test set consisting of 2,539 images across 102 flower categories. The results demonstrated that the deployed model preserved classification effectiveness comparable to the original MobileNetV2 model, achieving weighted precision, recall, and F1-score values of 0.95. In addition, the optimisation process reduced the model size from 27.74 MB to 2.53 MB, improving its suitability for mobile deployment by reducing storage requirements and memory consumption. The optimised model was subsequently integrated into an Android-based flower recognition application using the TensorFlow Lite Interpreter. To improve deployment efficiency, the TFLite model was loaded using memory mapping, allowing the model file to be accessed directly from storage without fully copying its contents into application memory. This approach reduces loading overhead and memory consumption, which is particularly beneficial for mobile devices with limited computational resources. The deployment pipeline incorporated model loading, image preprocessing, on-device inference, and prediction decoding through the associated class label mappings. Combined with the reduced model size achieved through optimisation, memory-mapped model loading contributes to efficient resource utilisation and supports responsive on-device inference during application execution. During inference, images acquired either from the device camera or the local image gallery are resized to 224×224 pixels and normalised before being processed by the deployed model. All prediction operations are performed locally on the device, eliminating dependence on external servers or cloud-based computation. The application then returns the flower category associated with the highest prediction confidence. The developed Android application is publicly available on Google Play as FlowerApp [31]. Fig. 10 shows the application interface for selecting a flower image from the camera or gallery and displaying the on-device prediction result.

To further validate the practical deployment performance of the proposed application, a series of benchmarking experiments were conducted on two physical Android devices. The evaluation platform consisted of a Redmi 10C equipped with 4 GB RAM and a Qualcomm Snapdragon 680 octa-core processor operating at up to 2.40 GHz, together with a Redmi Note 13 equipped with 8 GB RAM and a Qualcomm Snapdragon 685 octa-core processor operating at up to 2.80 GHz. Evaluating the application on multiple devices provides a realistic assessment of its behaviour under different hardware configurations commonly found in Android smartphones. For each device, the application was executed using a benchmark dataset consisting of ten flower images selected from different categories of the Oxford 102 Flowers test set. Each image was processed ten times per benchmark cycle, resulting in 100 timed executions per cycle. The experiment was repeated three times with a two-minute cooling interval between consecutive cycles, producing a total of 300 end-to-end inference measurements per device. The benchmark results are summarised in Table 3.



(a) (b) (c) (d)

Fig. 10 FlowerApp: (a) home (b) corn poppy (c) tiger lily (d) rose

Table 3 Mobile benchmark results on physical Android devices

Device	Runs	Mean (ms)	p50 (ms)	p90 (ms)
Redmi 10C	300	68.11	64.03	75.10
Redmi Note 13	300	55.57	55.26	55.81

To characterise deployment performance, three latency statistics were considered: the mean latency, the 50th percentile (p50), and the 90th percentile (p90). The mean latency provides an overall measure of average execution time, whereas p50 represents the median response time observed during benchmarking. In contrast, p90 reflects the latency threshold below which 90% of all inference executions were completed, thereby providing insight into less frequent but higher-latency execution scenarios. Together, these metrics offer a more comprehensive assessment of runtime behaviour than average latency alone.

Beyond summary statistics, the latency distributions are illustrated in Fig. 11 using box plots.

This visualisation provides additional information regarding execution consistency, variability, and the presence of anomalous latency measurements. The relatively compact interquartile ranges indicate stable inference behaviour across repeated executions, while the limited number of outliers suggests that high-latency events occurred infrequently during benchmarking. Consequently, the recorded latency measurements demonstrate that the deployed application is capable of performing flower recognition within a short response time while maintaining consistent execution characteristics across different deployment conditions. Overall, the deployment

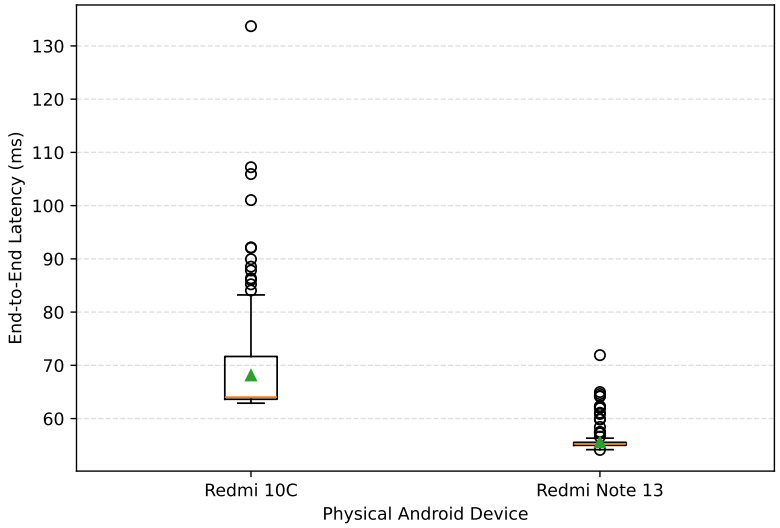


Fig. 11 End-to-end latency on physical Android devices

and benchmarking results demonstrate that the proposed system can deliver accurate and responsive on-device flower recognition while maintaining stable runtime characteristics, confirming its suitability for real-time mobile deployment.

5 Conclusion

This study presented a mobile-centric comparison of four independently trained and evaluated approaches for real-time flower recognition. Instead of applying the same transfer-learning configuration to all CNNs, different classification heads and training strategies were tested for each model on the Oxford 102 Flowers dataset. These experiments showed that each architecture required a model-specific configuration to achieve suitable learning and validation performance. Among the evaluated configurations, the adapted MobileNetV2 provided the best balance between recognition performance and mobile deployment requirements. Using a classification head and fine-tuning strategy selected specifically for the target dataset, MobileNetV2 achieved a weighted F1-score of 0.96 and was selected as the final model. The selected MobileNetV2 was converted to TensorFlow Lite for offline Android deployment, reducing the model size from 27.74 MB to 2.53 MB. Tests on physical Android devices recorded inference latencies of approximately 55.57–68.11 ms, confirming the feasibility of responsive flower recognition without cloud processing.

These findings show that model-specific adaptation, predictive performance, inference latency, and model size should be considered together when selecting models for mobile deployment. Future work may evaluate other lightweight architectures, more diverse datasets, and more challenging real-world imaging conditions.

Statements and Declarations

Availability of Data and Material

The source code of our proposed neural network model is available at the following GitHub link: https://github.com/fatemehsabourinia/Flower-Classification. This repository contains all the necessary information, including the model architecture. The Oxford 102 Flowers dataset analysed in this study is publicly available from the Visual Geometry Group, University of Oxford [14]. Additional experimental results and the raw mobile-device benchmark measurements supporting Table 3 and Fig. 11 are available from the corresponding author upon reasonable request.

Competing Interests

The authors declare that they have no relevant financial or non-financial interests that could have influenced the work reported in this article.

Funding

No funds, grants, or other support were received from any organization for conducting this study or preparing this manuscript.

Author Contributions

Fatemeh Sabourinia and Alireza Zaeri contributed to the study conception and design, methodology development, implementation of the model pipelines and mobile application, experimental evaluation, data analysis, figure preparation, and writing of the original manuscript. Vahid Abolghasemi supervised the study, provided methodological guidance, and reviewed and edited the manuscript. All authors read and approved the final manuscript.

Acknowledgements

The authors have no acknowledgements to declare.

References

[1] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.-C.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4510–4520 (2018). https://doi.org/10.1109/CVPR.2018.00474

[2] Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 8697–8710 (2018). https://doi.org/10.1109/CVPR.2018.00907

[3] Rajkomar, G., Pudaruth, S.: A mobile app for the identification of flowers using deep learning. International Journal of Advanced Computer Science and Applications **14**(5), 76–102 (2023) https://doi.org/10.14569/IJACSA.2023.0140508

[4] Chen, T., Guestrin, C.: Xgboost: A scalable tree boosting system. In: Proceedings of the 22nd Acm Sigkdd International Conference on Knowledge Discovery and Data Mining, pp. 785–794 (2016). https://doi.org/10.1145/2939672.2939785

[5] Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, vol. 25, pp. 1097–1105 (2012)

[6] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: International Conference on Learning Representations (2015)

[7] Guo, T.: Cloud-based or on-device: An empirical study of mobile deep inference. In: 2018 IEEE International Conference on Cloud Engineering (IC2E), pp. 184–190 (2018). https://doi.org/10.1109/IC2E.2018.00042

[8] Lane, N.D., Bhattacharya, S., Georgiev, P., Forlivesi, C., Jiao, L., Qendro, L., Kawsar, F.: Deepx: A software accelerator for low-power deep learning inference on mobile devices. In: 2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), pp. 1–12 (2016). https://doi.org/10.1109/IPSN.2016.7460664 . IEEE

[9] Ignatov, A., Timofte, R., Chou, W., Wang, K., Wu, M., Hartley, T., Van Gool, L.: AI Benchmark: Running deep neural networks on Android smartphones. In: Computer Vision – ECCV 2018 Workshops. Lecture Notes in Computer Science, vol. 11133, pp. 288–314. Springer, Cham, Switzerland (2019). https://doi.org/10.1007/978-3-030-11021-5_19

[10] Reda, M., Suwwan, R., Alkafri, S., Rashed, Y., Shanableh, T.: AgroAId: A mobile app system for visual classification of plant species and diseases using deep learning and TensorFlow Lite. Informatics **9**(3), 55 (2022) https://doi.org/10.3390/informatics9030055

[11] Malik, O.A., Ismail, N., Hussein, B.R., Yahya, U.: Automated real-time identification of medicinal plants species in natural environment using deep learning models—a case study from borneo region. Plants **11**(15), 1952 (2022) https://doi.org/10.3390/plants11151952

[12] Gavai, N.R., Jakhade, Y.A., Tribhuvan, S.A., Bhattad, R.: Mobilenets for flower classification using tensorflow. In: 2017 International Conference on Big Data, IoT and Data Science (BID), pp. 154–158 (2017). https://doi.org/10.1109/BID.2017.8336590 . IEEE

[13] Burhan, I.M., Ali, Q.A., Hussein, I.S., Jaleel, R.A.: Mobile-computer vision model with deep learning for testing classification and status of flowers images by using iots devices. J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl. **14**(1), 82–94 (2023) https://doi.org/10.58346/JOWUA.2023.I1.007

[14] Nilsback, M.-E., Zisserman, A.: 102 Category Flower Dataset. Visual Geometry Group, University of Oxford. Accessed 16 May 2026 (2008). https://www.robots.ox.ac.uk/%7Evgg/data/flowers/102/

[15] Shorten, C., Khoshgoftaar, T.M.: A survey on image data augmentation for deep learning. Journal of big data **6**(1), 60 (2019) https://doi.org/10.1186/s40537-019-0197-0

[16] Buda, M., Maki, A., Mazurowski, M.A.: A systematic study of the class imbalance problem in convolutional neural networks. Neural networks **106**, 249–259 (2018) https://doi.org/10.1016/j.neunet.2018.07.011

[17] Shwartz-Ziv, R., Armon, A.: Tabular data: Deep learning is not all you need. Information fusion **81**, 84–90 (2022) https://doi.org/10.1016/j.inffus.2021.11.011

[18] Mascarenhas, S., Agarwal, M.: A comparison between vgg16, vgg19 and resnet50 architecture frameworks for image classification. In: 2021 International Conference on Disruptive Technologies for Multi-disciplinary Research and Applications (CENTCON), vol. 1, pp. 96–99 (2021). https://doi.org/10.1109/CENTCON52345.2021.9687944 . IEEE

[19] Bimorogo, S.D., Kusuma, G.P.: A comparative study of pretrained convolutional neural network model to identify plant diseases on Android mobile device. International Journal of Advanced Trends in Computer Science and Engineering **9**(3), 2824–2833 (2020) https://doi.org/10.30534/ijatcse/2020/53932020

[20] Dong, K., Zhou, C., Ruan, Y., Li, Y.: Mobilenetv2 model for image classification. In: 2020 2nd International Conference on Information Technology and Computer Application (ITCA), pp. 476–480 (2020). https://doi.org/10.1109/ITCA52113.2020.00106 . IEEE

[21] Thapa, B., Adhikari, S., Pokhrel, B., Gauli, A.: A hybrid approach for sugarcane leaf disease classification using CNN and gradient boosting algorithms. In: Proceedings of the 15th IOE Graduate Conference, vol. 15, pp. 81–87 (2024)

[22] Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. Journal of Machine Learning Research **13**(10), 281–305 (2012)

[23] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition, pp. 248–255 (2009). https://doi.org/10.1109/CVPR.2009.5206848 . Ieee

[24] Pan, S.J., Yang, Q.: A survey on transfer learning. IEEE Transactions on Knowledge and Data Engineering **22**(10), 1345–1359 (2010) https://doi.org/10.1109/TKDE.2009.191

[25] Lin, M., Chen, Q., Yan, S.: Network in network. In: International Conference on Learning Representations (2014). https://arxiv.org/abs/1312.4400

[26] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: International Conference on Learning Representations (2015). https://arxiv.org/abs/1412.6980

[27] Caruana, R., Lawrence, S., Giles, C.L.: Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In: Advances in Neural Information Processing Systems, vol. 13, pp. 402–408 (2000)

[28] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research **15**(56), 1929–1958 (2014)

[29] Sokolova, M., Lapalme, G.: A systematic analysis of performance measures for classification tasks. Information Processing & Management **45**(4), 427–437 (2009) https://doi.org/10.1016/j.ipm.2009.03.002

[30] Singh, A., Bhadani, R.: Mobile Deep Learning with TensorFlow Lite, ML Kit and Flutter: Build Scalable Real-world Projects to Implement End-to-end Neural Networks on Android and iOS. Packt Publishing Ltd, Birmingham, UK (2020)

[31] Zaeri, A., Sabourinia, F.: FlowerApp. Google Play. Accessed 5 August 2026 (2026). https://play.google.com/store/apps/details?id=com.zaeri.sabourinia.flowerapp