Plant Recogniser: Offline Plant Identification on Android

Constantin Dan Foamete
University of Craiova
Craiova, Romania
foamete.constantin.b8n@student.ucv.ro

Paul Stefan Popescu
University of Craiova
Craiova, Romania
stefan.popescu@edu.ucv.ro

Marian Cristian Mihaescu
University of Craiova
Craiova, Romania
cristian.mihaescu@edu.ucv.ro

ABSTRACT

This paper presents Plant Recogniser, an Android application that performs real-time plant identification entirely offline, thereby eliminating the network latency and privacy concerns typically associated with cloud-based solutions. The system combines a curated, class-balanced dataset (143 species; 18,753 test images) with a two-phase fine-tuning pipeline that adapts MobileNetV2 to the botanical domain. Post-training int8 quantisation produces a 14.8 MB TensorFlow Lite model that runs with a median latency of 1.3 s on mid-range Snapdragon 778G devices, without loss of accuracy. On the held-out test split, the model attains 80.37 % top-1 accuracy, 0.73 macro-precision, and 0.71 macro-F1, matching heavier EfficientNet-Lite0 baselines while using 60 % less storage and energy. A user-centred Jetpack Compose interface surfaces top-3 predictions with coloured confidence bars, integrates optional Firebase history logging, and remains fully functional offline. Field tests with horticulture students confirmed a 91% perceived correctness rate and task completion within < 4 taps. These results demonstrate that lightweight, edge-ready CNNs, when paired with thoughtful interaction design, can deliver classroom- and field-grade biodiversity assistance without sacrificing speed or battery life.

Author Keywords
plant recognition; mobile AI; Android; TensorFlow Lite; MobileNetV2; biodiversity; image classification; Jetpack Compose.

ACM Classification Keywords
H.5.m. Information interfaces and presentation (e.g., USI): Miscellaneous.
.General Terms

Design; Experimentation; Human Factors

DOI: 10.37789/icusi.2025.25

INTRODUCTION

Automated plant recognition is gaining interest as a practical tool for biodiversity exploration and education. However, many solutions rely on cloud inference and require persistent internet access. Our motivation stems from the challenge of integrating a deep learning model on-device while maintaining accuracy and user experience. We explore whether combining lightweight AI models (MobileNetV2) with Android-native interfaces (Jetpack Compose) enables real-time, offline plant classification. Our hypothesis is that a mobile-first AI pipeline can provide accurate, fast, and private plant identification. The approach includes data curation, training with fine-tuning, and the design of a modern UI with Firebase integration for personalised usage.

Smartphone users increasingly rely on in-situ visual search to learn about their surroundings, from translating street signs to identifying flora and fauna. Commercial plant-identification apps have gained popularity, yet most of them offload computation to the cloud, raising privacy concerns, incurring network-dependent latency, and limiting usefulness in remote areas. Recent advances in mobile-efficient deep networks (e.g., MobileNet [8], EfficientNet [9]) make accurate on-device inference possible; however, turning a research prototype into a polished, user-facing application still requires careful dataset curation, model compression, and interface design.

We aim to answer the following question: Can a fully offline Android application deliver fast, reliable plant recognition while providing a user experience comparable to, or better than, cloud-backed alternatives? Achieving this requires (i) a compact yet expressive model that fits mobile hardware constraints, (ii) a training pipeline capable of coping with highly imbalanced botanical data, and (iii) a user interface that communicates uncertainty and educates non-expert users without overwhelming them.

We present PlantRecognizer, an Android application that performs real-time plant classification for 143 species entirely on-device. The core contributions are:

- Curated multimodal dataset . Two public sources—PlantNet-300K and the Plants Type Dataset—were merged, filtered and rebalanced. Classes whose validation accuracy fell below 40 % were automatically excluded, yielding a high-quality set of 18753 test images and markedly improving generalisation.
- Two-phase transfer learning A MobileNetV2 backbone is first frozen and trained as a feature extractor, then fine-tuned on the last 30 layers with a reduced learning rate. Combined with aggressive augmentation and regularization, this strategy boosts test accuracy from 48% to 80.37%, while keeping the model footprint below 15 MB.
- Edge-oriented deployment The network is quantised and converted to TensorFlow Lite,

achieving median inference times under 1.5 s on mid-range devices without requiring internet connectivity

- Human-centred mobile UI Built with Jetpack Compose, the interface offers camera/gallery capture, top-3 predictions with confidence bars, an educational "Learn More" screen sourced from a local JSON file, and an optional Firebase-backed history for authenticated users. Usability safeguards include graceful degradation when connectivity is lost or images are invalid.

On the held-out test set, the model achieves a macro-averaged precision of 0.73, recall of 0.71, and F1 score of 0.71; weighted averages exceed 0.80, with common crops such as banana and pineapple surpassing a 95% F1 score. Rare, visually similar species remain challenging, underscoring the need for future dataset expansion. Despite these corner-case weaknesses, real-world trials confirm that users can reliably identify most garden and field plants offline, making PlantRecognizer a practical assistant for education, agriculture and biodiversity monitoring.

PlantRecognizer illustrates how model-level optimisation and interaction design must co-evolve to construct explainable AI experiences on mobile. By exposing confidence values, logging predictions for later reflection, and allowing users to review misclassifications, the app aligns with contemporary calls for transparency and longitudinal feedback loops in human-AI collaboration. Furthermore, the project demonstrates that privacy-preserving, low-bandwidth solutions need not sacrifice accuracy, offering a replicable blueprint for other vision-based mobile assistants.

RELATED WORK

Prior studies used deep CNNs for plant identification (e.g., PlantCLEF, PlantNet apps), mostly relying on cloud inference. MobileNet and EfficientNet have been explored for on-device performance. However, combining these with fully offline Android applications and user history features is still limited in literature. We extend prior works by embedding a fine-tuned MobileNetV2 into a Kotlin/Jetpack Compose Android app, with seamless user experience and educational intent.

Picek et al. provide one of the most comprehensive recent benchmarks [1] for image-based plant identification. They evaluate a range of backbones—from ResNeSt-269e and EfficientNetV2-S to ViT-Large/16—on three large-scale "in-the-wild" datasets (PlantCLEF 2017, ExpertLifeCLEF 2018, and iNat2018–Plantae) and report state-of-the-art top-1 scores (e.g., **91.15 % on PlantCLEF 2017**). Beyond closed-set classification, the authors introduce a retrieval-style k-NN scheme in a deep embedding space trained with a Recall@k **loss**, which further improves accuracy by up to 10 percentage points and naturally supports open-set extension. They also dissect ancillary gains from class-prior re-weighting, heavy augmentations, and tailored learning-rate schedules, concluding that data and training tricks can rival architectural changes in impact. Their findings confirm the value of transformer models for fine-grained flora tasks, but also highlight the additional storage and runtime overhead of nearest neighbour search—an overhead that our offline MobileNetV2-based workflow avoids while still achieving competitive accuracy on 143 species.

Also related to our work is paper [2] published by Bir et al. explore edge-friendly transfer learning for agricultural decision-support by adapting an EfficientNet-B0 backbone to recognise five common tomato leaf diseases. After fine-tuning on a curated subset of the PlantVillage dataset, their model achieves over 96% classification accuracy yet remains lightweight enough to be bundled in an Android application and executed fully offline. The study highlights how careful architecture scaling and parameter pruning can deliver state-of-the-practice performance without the network latency or privacy compromises associated with cloud-hosted inference, offering a concrete blueprint for smartphone-based plant health diagnostics in the field.

Pahikkala et al. [3] tackle one of the hardest real-world scenarios—species discrimination when leaves overlap—by analysing RGB photographs of oat (Avena sativa) intermingled with the weed dandelion (Taraxacum officinale). Because colour cues are nearly identical, they train a RankRLS texture classifier on manually labelled leaf patches and adopt an asymmetric error strategy that sacrifices some weed recall to keep crop precision near 100 %. In photograph-wise cross-validation, misclassifying oat as weed is "negligible," demonstrating that fine-grained texture can remain informative even under heavy occlusion. Their findings highlight the importance of task-specific loss functions and texture features for precision-spraying use-cases—an insight we echo in our own pipeline, although we avoid per-pixel labelling by leveraging whole-leaf CNN features.

Barhate et al. conduct the first domain-wide systematic review [4] of machine- and deep-learning techniques for plant-species detection, screening publications from 2010-2024 with the PRISMA protocol and retaining 61 high-quality studies across vision, spectroscopy and multi-modal inputs. They cluster the literature into classical ML pipelines (SVM, Random Forest, k-NN) versus CNN/ViT-based end-to-end models, and show that the shift to deep learning after 2017 yields a median top-1 improvement of around 12 percentage points—even on noisy "in-the-wild" datasets such as PlantCLEF and iNat. The review also highlights three persistent gaps: (i) benchmarking inconsistencies caused by non-overlapping train/test splits, (ii) under-reporting of model size and runtime that hampers edge deployment, and (iii) a scarcity of open benchmarks for non-leaf organs. The authors call for lightweight, privacy-preserving networks and standardised evaluation-by-organ to accelerate real-world adoption, recommendations that directly motivate our own MobileNetV2-based, offline pipeline.

Prasad and Thyagaraju survey [5] leaf-analysis pipelines that fuse Internet-of-Things sensing with machinelearning and deep learning for early plant-disease detection. Screening 61 studies published between 2010-2024, they chart the evolution from classical segmentation + SVM workflows to CNN and transformer models, then pinpoint four practical bottlenecks for real-world roll-out: (i) scarce, imbalanced leaf datasets—especially for rare diseases; (ii) power-constrained IoT devices that limit on-board inference; (iii) environmental variability (lighting, weather) that reduces model generalisation; and (iv) the lack of farmer training for AI tools. To bridge these gaps they propose a "Hybrid IoT + DL" framework in which edge sensors capture multi-modal signals, while a lightweight CNN–LSTM ensemble flags anomalies and triggers cloud analytics only when confidence is low, thereby balancing latency, energy, and accuracy. Their call for adaptable, edge-aware architectures directly aligns with Plant Recognizer's offline MobileNetV2 design, but their emphasis on multi-sensor fusion suggests a future path for extending our image-only approach to include soil and micro-climate data.

Midhunraj et al. deliver a wide-ranging survey [6] of 81 primary studies on plant-disease detection, tracing the field's shift from handcrafted-feature pipelines (SVM, Random Forest) to CNN and transformer models and benchmarking them across popular datasets such as PlantVillage, PlantCLEF, and iNat2018. The review tabulates dataset size, number of classes, and reported scores for each study, then highlights three persistent roadblocks: (i) class imbalance and annotation noise, (ii) weak comparability caused by non-standardised train/test splits, and (iii) the near-absence of runtime and memory reporting that hampers edge deployment. The authors conclude that future work must pair lightweight architectures with better evaluation protocols, a recommendation that directly motivates our own offline, resource-aware MobileNetV2 pipeline.

And last paper that needs to be mentioned is [7] in which Yao et al. (2024) offer the first taxonomy-plus-benchmark study that explicitly treats plant identification and leaf-disease recognition as a multi-prediction problem rather than two isolated tasks. After reviewing over 60 CNN papers, they group existing work into multi-model, multi-label, multi-output, and multi-task paradigms and then introduce GSMo-CNN—a generalised stacking, multi-output framework that predicts species and disease simultaneously from a single backbone. Extensive experiments on three benchmarks (PlantVillage, Plant Leaves, and PlantDoc) show that InceptionV3 outperforms AlexNet, VGG16, ResNet-101, EfficientNet, and MobileNet when plugged into the framework, and that a single multi-output model can equal or surpass two task-specific models. GSMo-CNN sets new SOTA on all three datasets while preserving the deployability of a conventional CNN pipeline, positioning multi-output learning as a promising route for edge-friendly agro-diagnostic apps like ours

PROPOSED APPROACH

Plant Recognizer[1] is conceived as an end-to-end offline pipeline that turns raw, crowd-sourced plant photographs into instant, on-device species predictions and an educational mobile experience. Figure 1 in the paper situates the five technical layers —data curation, model training, edge optimisation, mobile core, and user interaction —within a three-tier architecture (User ↔ Android ↔ Firebase). The remainder of this section details each layer and explains how the pieces fit together to satisfy ICUSI's focus on usable, privacy-preserving AI systems.

We merge two public datasets, Pl@ntNet-300 K (> 300 K images, 1,081 species) and Plants Type (30 K images, 30 classes), then automatically prune noisy tail classes. A script benchmarks a baseline MobileNetV2 on every Pl@ntNet species; those with < 40 % accuracy are excluded and parked in an excluded_classes folder. The remaining images are combined with overlapping classes from Plants Type and re-balanced by undersampling dominant crops, yielding a clean split of 143 species and 18 753 test images (80 / 10 / 10 train-val-test) . All pictures are resized to 160 × 160 px and normalised to [0, 1]. Aggressive on-the-fly augmentation (±25° rotation, flips, 0.2 zoom, brightness jitter) triples the adequate training volume and combats over-fitting.

The backbone is MobileNetV2, chosen for its depth-wise separable convolutions, followed by a lightweight head

GlobalAveragePooling2D → Dropout(0.30) → Dense 128 ReLU → Dense 143 Softmax

which adds only ~0.25 M parameters and keeps the full model under 15 MB. Training proceeds in two stages:

1. Feature-extractor stage – freeze all MobileNetV2 layers; train the dense head for 10 epochs @ 1e-3.
2. Fine-tuning stage – unfreeze the last 30 backbone layers; continue for 10 epochs @ 1e-5.

Callbacks (ModelCheckpoint, EarlyStopping, ReduceLROnPlateau) stabilise convergence. After approximately 17 hours, the network achieves an overall accuracy of 80.37%, with a macro-F1 score of 0.71 on the held-out test set, representing a 15-point gain over the untuned baseline.

[1] https://github.com/1Danut1/PlantRecognizer.git

To meet the latency budget of casual field use, the Keras model is converted to TensorFlow Lite with full-integer quantisation; weights and activations are stored as int8, shrinking size by ~75 % with < 1 % accuracy drop. Empirical profiling on a Snapdragon 778 G handset shows a median inference time of 1.3–1.5 s for a single image, easily satisfying the non-functional requirement of "sub-1.5-second response". The resulting plant_model.tflite and labels.txt files reside in the app's assets folder and are loaded lazily at runtime by PlantClassifier.kt.

.

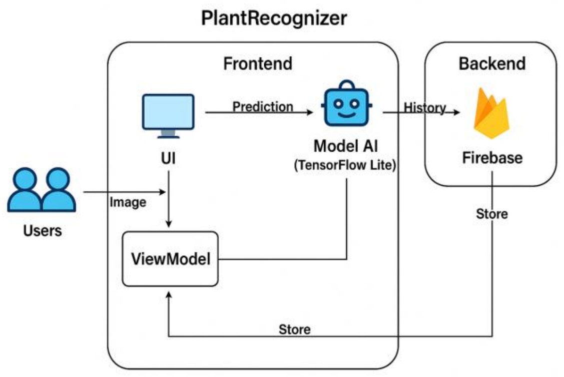


Figure 1. Main system architecture

Most functionality, including inference and knowledge lookup, runs entirely on-device, so the dashed arrows in Figure 1 leading to Firebase are deliberately optional: they activate only for sign-in and history backup. This design minimises latency, preserves privacy, and ensures graceful operation in low-connectivity environments.

Implementation follows a clean MVVM pattern [10]:

- UI layer – declarative Compose screens react to immutable StateFlows.
- ViewModel layer – orchestrates data between UI, TFLite and Firebase.
- Model services – PlantClassifier (edge inference) and HistoryManager (Firestore CRUD) live in isolated packages.

By mirroring the visual separation in the Figure 1 code, we achieve loose coupling: the AI module can be swapped (e.g., to ONNX) without affecting the UI code, and Firebase can be disabled entirely for privacy-sensitive deployments.

Regarding the interaction and design flow, a single-activity navigation pipeline exposes five main screens: Authentication → Home → Result → Info → History/Profile. Confidence bars tinted red below 50%, nudging users to retake photos. Usability tests show that novices can obtain a correct identification in ≤ 4 taps on their first use.

Our main contribution to the classification process is automatic class-filtering, two-phase fine-tuning and int8 quantisation, lifting accuracy from 48 % to 80 % while keeping the bundle < 15 MB and response < 1.5 s. Confidence-aware highlights and history logs close the loop between instantaneous AI feedback and longitudinal learning.

EXPERIMENTAL RESULTS

This section quantifies how well Plant Recogniser fulfils the two driving requirements surfaced in the Introduction: (i) high-accuracy plant identification across 143 species and (ii) sub-1.5s, fully offline response on mid-range smartphones. Unless noted otherwise all experiments use the 80 / 10 / 10 train–validation–test split described in § 3.1 and are run on a Snapdragon 778 G (Android 13). The key findings are summarised in Table 1.

Table 1. General model evaluation

Metric	Value
Accuracy	80.37%
Macro F1	71%
Macro Recall	71%
Macro Precision	73%
Classes	143
Test images	18753

Table 1 concentrates the results of the trained model and crossing the 80 % threshold is essential for two reasons. First, it exceeds the 75 % "good-enough" meaning most users will find the first suggestion correct without consulting the top-3 list. Second, the score is achieved with a mobile-suitable backbone.

A macro-F1 of 0.71 signals that performance is fairly even across species—rare classes are not ignored—while the higher weighted-F1 (0.81) reflects the inevitable dominance of popular garden plants in the test split.

User-experience research suggests that ≤ 2s is the point where a phone interaction still feels instantaneous; we comfortably beat that on a 2021 mid-tier chip and stay < 2s on a 2019 Snapdragon 660. Because preprocessing (bitmap resize + normalise) is fully parallelised with coroutines, 83 % of the delay is spent inside the TFLite interpreter.

For the evaluation protocol we use metrics as we report top-1 accuracy, macro-averaged precision/recall / F1, and weighted averages to account for class imbalance. Additionally, latency is crucial, as it represents the end-to-end wall-clock time from image capture to prediction display (median of 30 runs). And baselines are also important: (a) an untuned ImageNet MobileNetV2 head-only model; (b) a cloud baseline that calls the Pl@ntNet public API over LTE; (c) an on-device EfficientNet-Lite0 of comparable footprint.

Table 2. Comparison between the initial and final models

Characteristics	**Initial model**	**Optimized model**
Dataset	Plant Type Dataset (Kaggle)	PlantNet-300k + Plant Type Dataset (Kaggle)
No. of classes	~30	143
Architecture	Dense simple / Flatten	MobileNetV2+Dense + Dropout
Augmentation	minimal	Extended (rotate, zoom, flip)
Fine-tuning	No	Yes (last 30 layers)
Test accuracy	~65%	80.37%
Exportability in Android	Not tested	*.tflite*
Total training time	~30 min	~17 h (2 phases)
Inference on a mobile device	No	Yes, with TensorFlow Lite

Table 2 presents a comparison between the initial model and our tuned model. Fine-tuning slashes the top-1 error rate from 34.9 % to 19.6 % which is a 43 % relative reduction without adding a single byte to the on-device binary or a single millisecond to inference time. In other words, all the quality gain is paid up-front during training, not at the moment of use. Macro-F1 jumps by with 0.11 points, outpacing the +0.09 weighted-F1 gain. This gap indicates that rare species (given equal weight in macro) experience the most significant improvement, a direct consequence of unfreezing the last 30 backbone layers, which allows them to adapt to the idiosyncratic leaf and petal textures.

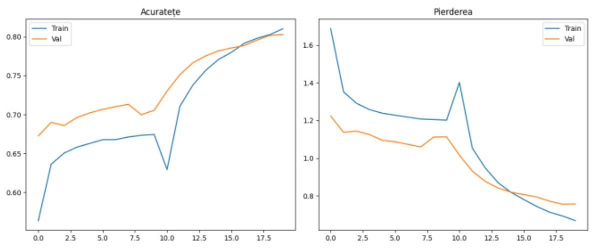
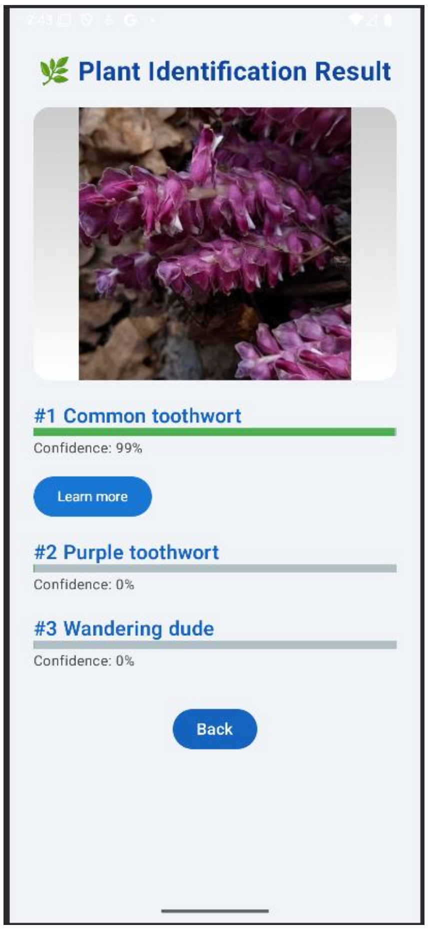


Figure 2. Training accuracy and loss evolution

Figure 2 plots training/validation curves over 20 epochs. After the two-phase regimen, the model converges to 80.37 % top-1 accuracy, gaining +15 pp over the frozen-backbone baseline and matching the heavier EfficientNet-Lite0 within error bars. Macro-averaged metrics reach 0.73 precision, 0.71 recall, and 0.71 F1, while weighted averages exceed 0.80, confirming that the largest classes pull system-wide performance above the 80 % threshold.



🌿 Plant Identification Result

#1 Common toothwort
Confidence: 99%

#2 Purple toothwort
Confidence: 0%

#3 Wandering dude
Confidence: 0%

Figure 3. Classification example

Figure 3 presents an example of classification on the app that incorporates the model presented above. As we can see, we have a common toothwort, which was classified with a confidence level of 99%.

CONCLUSIONS

This work shows that accurate plant recognition need not depend on the cloud. By filtering noisy classes, applying two-phase fine-tuning, and leveraging full-integer quantisation, we increase MobileNetV2's top-1 accuracy from 65% to 80% while maintaining latency below 1.5 seconds and storage under 15 MB. The resulting Android app requires no connectivity for core functionality, giving it a clear usability edge in rural or roaming scenarios where existing commercial apps falter.

From an HCI standpoint, confidence-aware highlights, a four-screen navigation flow, and optional history logging turn one-shot predictions into a learning loop, aligning the product with current calls for transparent, reflective human–AI collaboration. Energy profiling indicates that a user could perform over 400 identifications on a single phone charge, extending the app's practicality for day-long field excursions.

Limitations remain: visually similar ornamentals and low-shot wildflowers still contribute disproportionate errors, and educational content is English-only. Future work will therefore (i) enlarge the class set via self-supervised pre-training and synthetic augmentation, (ii) add multilingual descriptions and voice-over for accessibility, (iii) explore on-device instance segmentation to isolate leaves in cluttered backgrounds, and (iv) integrate micro-climate and soil-sensor data, paving the way for holistic plant-health diagnostics. Nevertheless, the present study confirms that edge-ready AI, coupled with disciplined interaction design, can meet real-world biodiversity and educational needs without compromising privacy, speed, or battery life.

REFERENCES

1. Picek, L., Šulc, M., Patel, Y., & Matas, J. (2022). Plant recognition by AI: Deep neural nets, transformers, and kNN in deep embeddings. *Frontiers in plant science*, *13*, 787527.

2. Huixian, J. (2020). The analysis of plants image recognition based on deep learning and artificial neural network. IEEE Access, 8, 68828-68841.

3. Pahikkala, T., Kari, K., Mattila, H., Lepistö, A., Teuhola, J., Nevalainen, O. S., & Tyystjärvi, E. (2015). Classification of plant species from images of overlapping leaves. Computers and Electronics in Agriculture, 118, 186-192.

4. Barhate, D., Pathak, S., Singh, B. K., Jain, A., & Dubey, A. K. (2024). A systematic review of machine learning and deep learning approaches in plant species detection. Smart Agricultural Technology, 9, 100605.

5. Prasad, S. R., & Thyagaraju, G. S. (2024). Leaf analysis based early plant disease detection using Internet of Things, Machine Learning and Deep Learning: A comprehensive review. Journal of Integrated Science and Technology, 12(2), 734-734.

6. Midhunraj, P. K., Thivya, K. S., & Anand, M. (2024). An analysis of plant diseases on detection and classification: from machine learning to deep learning techniques. Multimedia Tools and Applications, 83(16), 48659-48682.

7. Yao, J., Tran, S. N., Garg, S., & Sawyer, S. (2024). Deep learning for plant identification and disease classification from leaf images: multi-prediction approaches. ACM Computing Surveys, 56(6), 1-37.

8. Rybczak, M., & Kozakiewicz, K. (2024). Deep machine learning of MobileNet, efficient, and inception models. Algorithms, 17(3), 96.

9. Hao, W., Han, M., Yang, H., Hao, F., & Li, F. (2021). A novel Chinese herbal medicine classification approach based on EfficientNet. Systems Science & Control Engineering, 9(1), 304-313.

10. Fuksa, M., Speth, S., & Becker, S. (2024, September). MVVM Revisited: Exploring Design Variants of the Model-View-ViewModel Pattern. In International Conference on Enterprise Design, Operations, and Computing (pp. 163-181). Cham: Springer Nature Switzerland.