SUNSCOUT: AN OFFLINE MOBILE APPLICATION FOR SOLAR PANEL FAULT DETECTION USING ON-DEVICE DEEP LEARNING

Zonglin Li [1] , Austin Amakye Ansah [2]

[1] Anglo-Chinese School, 121 Dover Rd, Singapore 139650
[2] The University of Texas at Arlington, 701 S Nedderman Dr, Arlington, TX 76019

ABSTRACT

Manual inspection of photovoltaic systems is expensive, hazardous, and prone to inconsistency. This paper presents SunScout, a mobile application for offline solar-panel image management and on-device fault classification. The current mobile release organizes drone, gallery, and camera captures into reusable datasets and analyzes each stored asset with a fine-tuned EfficientNetB0 classifier deployed through ONNX Runtime. On the cleaned 1,575-image dataset, a frozen MobileNetV2 baseline reached 90.79% validation accuracy, while the proposed EfficientNetB0 model achieved 94.29%, a 3.50 percentage point improvement. Together, these results show that accurate solar fault analysis can be delivered on a consumer smartphone without requiring a persistent network connection.

KEYWORDS

Solar panel inspection, Fault classification, EfficientNet, Computer vision, Mobile deployment, On-device inference

1. INTRODUCTION

Solar panels operate continuously and require periodic maintenance to sustain peak efficiency. Visual inspection costs can range from $150 to $350 per visit [1]. Despite these recurring maintenance expenses, traditional inspection practices still rely heavily on manual rooftop evaluations performed by qualified technicians. These inspections introduce significant safety hazards, operational inefficiencies, and diagnostic inconsistencies that limit the effectiveness of long-term photovoltaic (PV) system maintenance. Manual roof inspections expose personnel to serious physical risks. Technicians must operate on elevated and often steep residential rooftops while working near energized electrical equipment, creating constant exposure to fall hazards, electrical shock, and laceration injuries from mounting hardware and panel frames [2]. Even with modern safety procedures such as lock-out/tag-out systems and personal protective equipment, rooftop access remains one of the most dangerous aspects of photovoltaic maintenance. Beyond safety concerns, manual inspection results are inherently inconsistent. Human visual assessment depends on technician experience, viewing angle, lighting conditions, and subjective judgment. As a result, subtle defects such as micro-cracks, early electrical degradation, or localized shading are frequently missed [3]. Traditional handheld diagnostic tools further limit accuracy because they measure only isolated points rather than providing full spatial coverage across an entire array. Inspection duration also presents a major operational challenge. Manual evaluation of solar

David C. Wyld et al. (Eds): WiMNeT, NLDM, EDUPT, SIP, AISO – 2026
pp. 43-55, 2026. CS & IT - CSCP 2026 DOI: 10.5121/csit.2026.1601005

installations requires technicians to physically traverse rooftops and inspect panels individually. Large systems may require hours or days to evaluate completely, encouraging partial sampling rather than full coverage [4]. This creates diagnostic blind spots, allowing undetected faults to compound over time and reduce system efficiency and financial return.

To address the limitations of manual inspection, this work proposes SunScout, a mobile application that transforms a standard smartphone into an intelligent solar inspection and dataset-analysis tool. Rather than treating each image as an isolated scan, the system lets users organize drone captures, gallery imports, and direct camera images into persistent datasets, review and annotate them, and then run a trained classifier across the entire collection. The deployed model categorizes each image into one of six operational classes: Bird-drop, Clean, Dusty, Electrical-damage, Physical-Damage, and Snow-Covered. The application stores predictions, confidences, user labels, and notes locally so that repeated analysis runs can be compared over time.

By eliminating the need for prolonged rooftop analysis and reducing dependence on specialized inspection equipment, SunScout provides a lightweight and accessible alternative to traditional inspection workflows. The system improves safety by shifting more review work onto captured imagery, enhances diagnostic consistency through automated classification, and supports rapid field documentation directly from a mobile device.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. Classification Accuracy on Visually Similar Classes

Although the baseline performed reasonably well on the cleaned dataset, visually similar categories such as Bird-Drop, Clean, and Dusty remained the main source of error. Frozen ImageNet features from the MobileNetV2 backbone still lacked some of the fine-grained texture sensitivity needed to separate subtle surface contamination patterns, with Bird-Drop remaining the weakest class at 78.6% accuracy.

2.2. Class Imbalance Across Defect Categories

The dataset exhibited significant class imbalance, with Snow-Covered panels representing only 123 images while Clean and Dusty classes contained 330 each. Without rebalancing, mini batches during training would be dominated by the majority classes, preventing the model from learning reliable representations for rare defect types. A weighted random sampler was required to ensure balanced class exposure across all training phases.

2.3. On-Device Deployment Constraints

Deploying a capable deep learning model to a consumer smartphone requires balancing classification accuracy against memory footprint, inference latency, and cross-platform compatibility. In the current mobile app, this constraint led to an ONNX-based classifier deployment that analyzes full dataset assets on device using `flutter_onnxruntime` [14]. This choice reduced implementation complexity preserved fully offline operation, and kept the mobile workflow aligned with the dataset-centric interface described in this paper.

3. SOLUTION

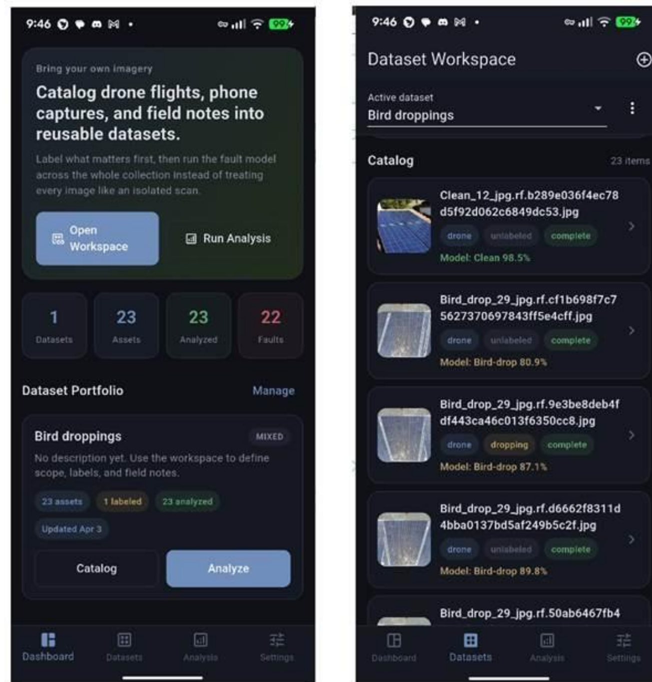

Figure 1. Dashboard and dataset workspace screens in the current SunScout mobile application

Sun Scout is implemented as a Flutter application that launches through `main.dart`, loads environment variables from `.env`, wraps the app in a Riverpod `ProviderScope`, and routes through `go_router` into a four-tab shell consisting of Dashboard, Datasets, Analysis, and Settings. The current mobile workflow is dataset-centric rather than inspection-centric. Users first create a dataset that represents a flight, site visit, or mixed capture session, then import imagery from drone exports, gallery selection, or direct camera capture. The dashboard aggregates portfolio-level statistics such as dataset count, asset count, analyzed assets, and detected faults, while the dataset workspace exposes per-dataset catalog management, manual labeling, and note entry for each stored image.

This architecture is primarily offline-first. Image ingestion, model execution, SQLite persistence, and local file storage all run on devices without requiring a network connection. The only optional cloud-assisted component is the natural-language insight generator used after batch analysis; when an API key is unavailable, the application falls back to deterministic locally generated summary text [8]. This split keeps the operational path usable in the field while still allowing richer narrative summaries when connectivity is available.

The deployed mobile application uses a single-stage image classification pipeline built around the exported EfficientNetB0 ONNX model. Each imported dataset asset is treated as a single analysis unit and classified directly into one of six categories: Bird-drop, Clean, Dusty, Electrical-damage, Physical-Damage, and Snow-Covered.

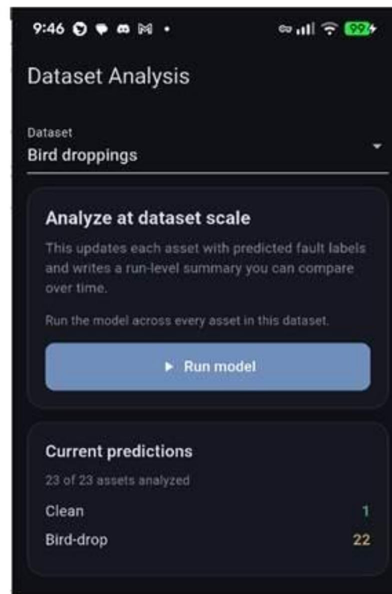

Figure 2. Dataset analysis screen showing batch inference, prediction breakdown, and run history

Model inference is implemented in `TFLiteClassifierService`, which now wraps `flutter_onnxruntime` rather than TensorFlow Lite despite the legacy service name. On startup, `loadModel()` opens `assets/models/best_model.onnx`, and each image is preprocessed in Dart using bilinear resize to 224 x 224 pixels, channel-first tensor layout, and ImageNet normalization constants. The raw logits returned by ONNX Runtime are converted to probabilities with a Dart softmax routine, after which the highest-probability class is emitted together with its confidence score [9].

Batch execution is orchestrated by `AnalysisPage`. For each asset in the selected dataset, the application reads the stored image bytes from disk, calls `classifyPanel`, and immediately writes the predicted label, confidence, and `analysis_status = complete` back into SQLite [10]. The page then aggregates counts for healthy versus defective assets, computes a category breakdown, and presents both a live prediction summary and a historical record of previous analysis runs. In other words, inference output is not transient UI state; it becomes part of the persistent dataset record and can be compared across repeated analysis passes.

Persistence is handled by a singleton `DatabaseService` backed by SQLite (`solarcheck.db`, schema version 4). The current schema is centered on three active tables: `datasets`, which store collection-level metadata; `dataset_assets`, which store imported image paths, optional user labels, notes, prediction results, and analysis status; and `dataset_runs`, which store run-level summaries after batch inference. Riverpod providers expose these tables to the UI as typed asynchronous models, allowing the dashboard, dataset workspace, and analysis screens to refresh automatically after import, relabeling, deletion, or inference.

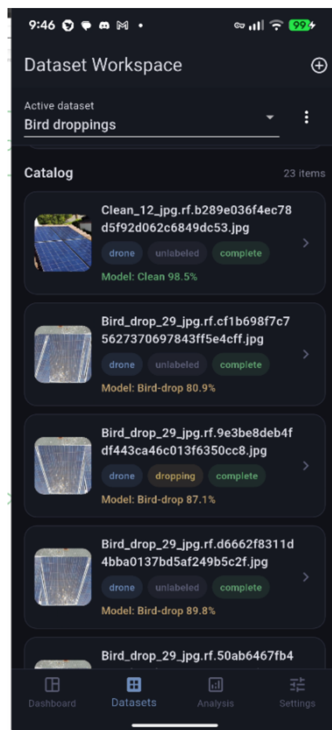

Figure 3. Dataset workspace screen showing persistent asset cataloging, labels, and model predictions

The persistence layer also coordinates filesystem state through `ImageStorageService`. Imported images are copied into the application documents directory under an `inspections/` folder and referenced from `dataset_assets.file_path`, which allows the app to work with stable local copies regardless of the original source. Dataset-level metrics such as `asset_count`, `labeled_count`, and `analyzed_count` are recomputed by `_refreshDatasetMetrics()` after every insert, update, prediction reset, or delete operation. When a dataset or asset is removed, the corresponding local image files are deleted before the database rows are cleared. Legacy inspection, detection, report, and inspection-image APIs still exist in the codebase for compatibility, but they are no longer part of the routed user workflow and are explicitly kept inert at the provider layer.

After a batch analysis pass completes, SunScout generates a run-level summary rather than a standalone PDF-style report. The application records how many assets were analyzed, how many were classified as Clean, how many were flagged as faults, and what categories were observed. These aggregate statistics are then transformed into user-facing narrative output that can be reviewed later from the Analysis screen alongside previous runs for the same dataset.

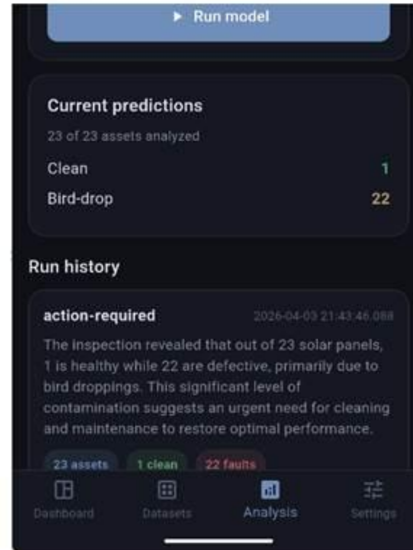

Figure 4. Run-history view showing stored batch summaries and recommendations after dataset analysis

Narrative generation is handled by `OpenAIService.generateInspectionInsights`. If `OPENAI_API_KEY` is configured, the app sends the dataset totals, defect counts, healthy counts, and category breakdown to the OpenAI API and requests structured JSON containing a summary, severity level, recommendations, and an efficiency note [11]. If the API is unavailable, the service falls back to a deterministic template that still produces actionable recommendations and an estimated array-health statement. The resulting text is stored in the `dataset_runs` table together with `status`, `asset_count`, `defect_count`, `healthy_count`, and a timestamp. The Analysis page then renders these stored runs as reusable historical records, which makes the reporting function part of the dataset lifecycle rather than a separate export-only feature.

4. EXPERIMENT

All classifier experiments were conducted on a workstation equipped with an NVIDIA GeForce RTX 3060 (12 GB VRAM). Input images were resized to 224×224 pixels and normalized using ImageNet statistics (mean = [0.485, 0.456, 0.406], std = [0.229, 0.224, 0.225]). An 80/20 train-validation split was applied with a fixed random seed (42) for reproducibility across all classifier experiments.

The classifier dataset was assembled from two sources. The primary source consisted of 870 manually curated solar panel images spanning six defect categories. To address class imbalance, particularly for Electrical-Damage and Physical-Damage—690 additional images were drawn from the Roboflow solar-panel.v1i.yolo26 dataset (CC BY 4.0), covering train, valid, and test splits. Images were matched to classifier categories via filename prefix. Images with generic filenames (prefixed IMG*) were excluded as their class label could not be reliably inferred; 31 such images were discarded. The final dataset comprised 1,575 images distributed as shown in Table 1.

Category	Image Count
Bird-Drop	281
Clean	330
Dusty	330
Electrical-Damage	206
Physical-Damage	305
Snow-Covered	123
Total	**1,575**

Table 1. Dataset class distribution

Class imbalance was handled during training via a weighted random sampler, with per-class weights computed as $w_c = N / (C \times n_c)$, ensuring balanced class exposure across mini-batches regardless of absolute class frequencies.

The current SunScout application deploys only the image classifier. Imported images are stored as dataset assets and analyzed individually by the ONNX EfficientNetB0 model described in Section 3. No object-detection stage is invoked in the routed Flutter workflow, and all classifier results reported below correspond to image-level six-class prediction rather than panel-level localization.

4.1. Experiment 1

The updated baseline uses a MobileNetV2 backbone (2.2M parameters) pretrained on ImageNet with all convolutional layers frozen. Only a single linear classification head (1280 -> 6) was trained on the current 1,260-image training split using Adam (lr = 1e-3), categorical cross-entropy loss, no class weighting, and no data augmentation beyond resizing.

On the cleaned 1,575-image dataset, this baseline achieved 90.79% overall validation accuracy on the 315-image held-out split after early stopping (best epoch 13, total training time 897.8 s). Performance remained weakest on Bird-Drop (78.6%), whose errors largely overlap with Dusty and Physical-Damage, but all other classes exceeded 89% [12]. This result indicates that a frozen ImageNet backbone is already competitive on the cleaned dataset, while still leaving a measurable gap to the proposed EfficientNetB0 pipeline.

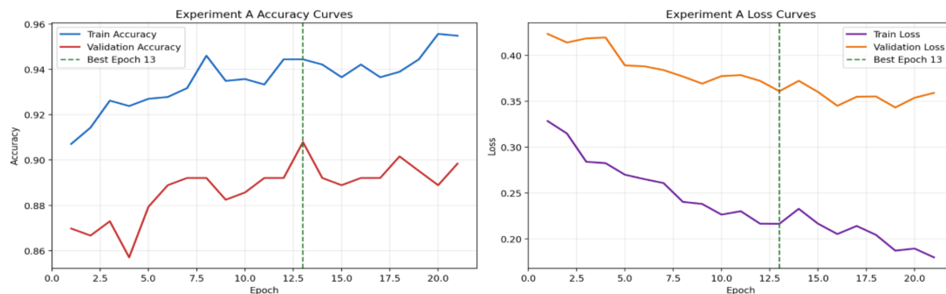

Figure 5. Training and validation curves for the rerun MobileNetV2 baseline (Experiment A)

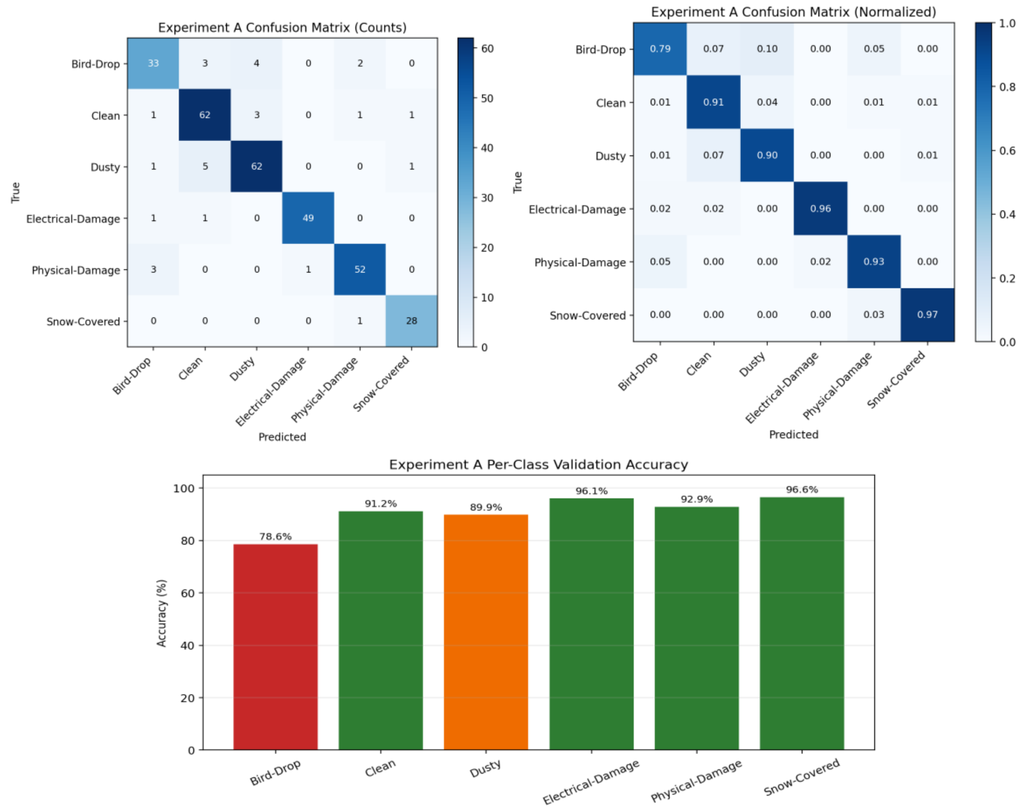

Figure 6. Confusion matrices and per-class accuracy for the rerun MobileNetV2 baseline (Experiment A)

4.2. Experiment 2

To overcome the accuracy ceiling, five changes were introduced simultaneously.

Backbone upgrade. MobileNetV2 was replaced with EfficientNetB0, which achieves higher ImageNet accuracy through compound scaling of depth, width, and resolution. At 4.2M parameters, EfficientNetB0 produces richer feature representations suited to fine-grained visual discrimination.

Progressive unfreezing. Training proceeded in three phases, progressively relaxing backbone constraints as shown in Table 2. Phase 1 trains the classification head against frozen ImageNet features. Phase 2 unfreezes the top three EfficientNet blocks, enabling high-level feature adaptation to the solar panel domain. Phase 3 allows full network fine-tuning at a further reduced learning rate.

Phase	Layers Trainable	Learning Rate	Max Epochs	Patience
1	Head only (backbone frozen)	1e−3	30	8
2	Top 3 blocks + head	1e−4	50	12
3	All layers	2e−5	40	15

Table 2. Progressive unfreezing schedule

Classification head. The single linear layer was replaced with a deeper head: Dropout(0.4) → Linear(1280, 128) → BatchNorm → ReLU → Dropout(0.3) → Linear(128, 6).

Data augmentation. The training pipeline applied random crop to 224×224, random horizontal and vertical flip, random 90°/180°/270° rotation, color jitter (brightness/contrast/saturation ±0.3, hue ±0.08), and random erasing (p = 0.5, scale 2–10%) after normalization.

Learning rate schedule. Each phase used cosine annealing with a 3-epoch linear warmup, decaying smoothly to near-zero by phase end:

$$lr(t) = lr_0 \times 0.5 \times (1 + \cos(\pi \times (t - tc) / (T - tc))).$$

All models were evaluated on the same held-out validation split (315 images, 20% of the 1,575-image dataset, fixed seed 42). Early stopping on validation accuracy prevented overfitting. Training curves confirmed val loss divergence in Phase 3 was minimal.

Metric	Exp. A (Baseline)	Exp. B (Proposed)
Backbone	MobileNetV2	EfficientNetB0
Parameters	2.2M	4.2M
Backbone fine-tuning	None (frozen)	Progressive 3-phase
Class weighting	None	Weighted sampler
Data augmentation	None	6 transforms
Training framework	PyTorch (RTX 3060)	PyTorch (RTX 3060)
Training time	15.0 min	65.6 min
Peak VRAM	1,028 MB	2,957 MB
Validation accuracy	90.79%	94.29%
Improvement over A	—	+3.50 pp
Export format	.pt checkpoint (8.8 MB)	ONNX (16 MB)

Table 3. Model comparison (Experiment A vs. Experiment B)

Category	Exp. A	Exp. B	Change
Bird-Drop	78.6%	85.7%	+7.1 pp
Clean	91.2%	97.1%	+5.9 pp
Dusty	89.9%	92.8%	+2.9 pp
Electrical-Damage	96.1%	96.1%	+0.0 pp
Physical-Damage	92.9%	94.6%	+1.7 pp
Snow-Covered	96.6%	100.0%	+3.4 pp
Overall	90.79%	94.29%	+3.50 pp

Table 4. Per-class validation accuracy

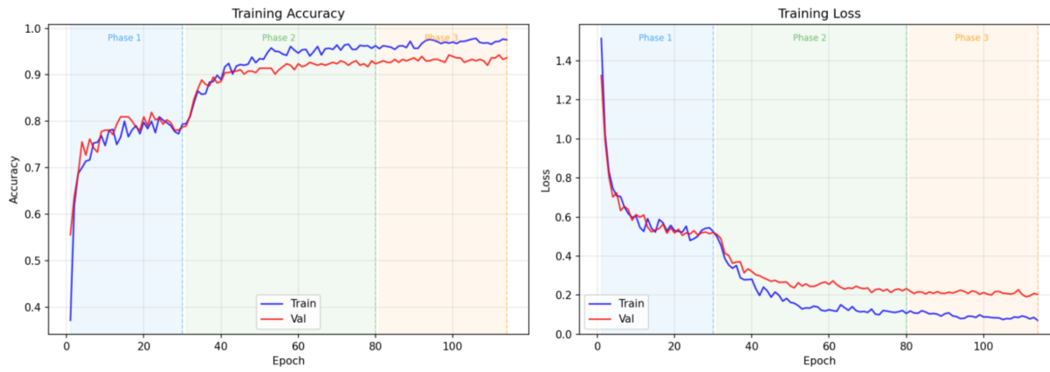

Figure 7. Training and validation accuracy/loss curves for Experiment B

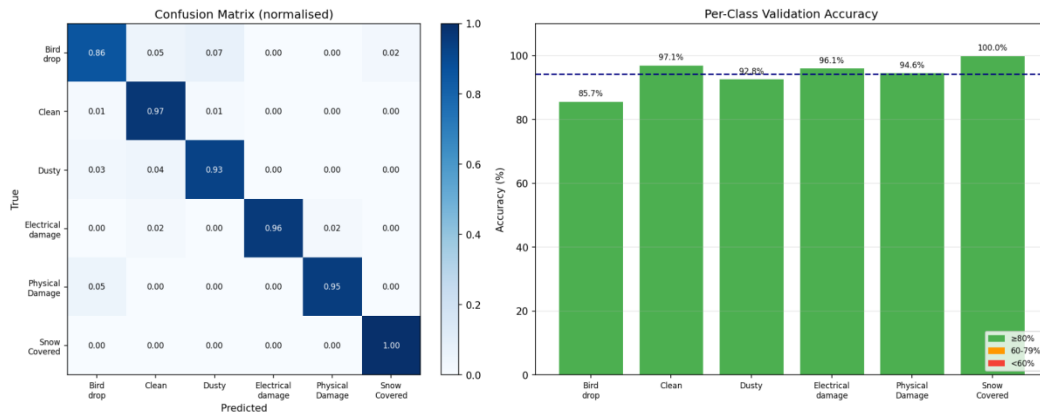

Figure 8. Per-class and overall evaluation metrics for Experiment B

Experiment B achieved 94.29% overall validation accuracy, a 3.50 percentage point improvement over the rerun MobileNetV2 baseline. The largest gains were observed on Bird-Drop (+7.1 pp) and Clean (+5.9 pp), while Electrical-Damage remained unchanged at 96.1%. All six classes in Experiment B exceed 85%, with five exceeding 92%. Snow-Covered reached 100% on the 29-image validation subset.

Training curves show val accuracy stabilizing around 94% by mid-Phase 2, with Phase 3 yielding marginal gains while val loss begins a slight upward trend—consistent with saturation rather than overfitting. The primary remaining weakness is Bird-Drop (85.7%), whose errors are distributed across Dusty and Physical-Damage, categories that share similar surface texture characteristics.

The 3.50 pp improvement over the rerun MobileNetV2 baseline arises from multiple simultaneous changes. Because these changes were introduced together, the following contribution estimates should be interpreted as approximate rather than causal measurements from a full ablation study.

1. Progressive unfreezing (estimated +1.0 to +1.5 pp). Allowing the upper EfficientNet layers to adapt to solar-panel imagery improved discrimination on visually ambiguous categories, particularly Bird-Drop and Clean, where frozen ImageNet features remained the main limitation of the baseline.

2. Data augmentation (estimated +0.8 to +1.2 pp). Random crop, flips, rotation, color jitter, and random erasing increased effective dataset diversity and improved robustness to viewpoint, lighting, and partial-surface visibility changes that are common in field imagery.

3. Class-weighted sampling (estimated +0.4 to +0.7 pp). Weighted mini-batch sampling prevented the model from overfitting to frequent classes such as Clean and Dusty and ensured continued exposure to rarer categories such as Bird-Drop and Snow-Covered throughout training.

4. Architecture and head design (estimated +0.5 to +0.8 pp). EfficientNetB0 provides stronger feature representations than MobileNetV2, and the deeper classification head with batch normalization and dropout adds useful capacity and regularization for the six-class defect task.

5. Longer converged training on GPU (estimated +0.3 to +0.5 pp). Running the PyTorch training pipeline on the RTX 3060 enabled the full three-phase schedule to reach early-stopping convergence in practical wall-clock time, which reduced undertraining risk relative to earlier shorter baseline attempts.

The trained EfficientNetB0 classifier was exported to ONNX format (opset 17, 16 MB) and deployed in the current SunScout mobile application via the `flutter_onnxruntime` package. In the shipped app, users import images into datasets and run classifier inference directly on each stored asset, with predictions and confidence scores written back to SQLite for later review [13]. This keeps the production mobile workflow simple, fully offline, and tightly aligned with the dataset-centric interface shown in Section 3.

The mobile release therefore reflects the classifier experiments reported in this paper rather than a detection-plus-classification stack. This distinction is important for interpretation: the reported deployment characteristics, screenshots, and stored run summaries all correspond to image-level classifier inference on device.

5. RELATED WORK

Recent research has explored automated photovoltaic inspection using aerial imaging and machine learning techniques. Tsanakas et al. [5] proposed a drone-based thermal imaging system capable of detecting deteriorated photovoltaic modules through automated image acquisition and analysis. The authors demonstrated that aerial thermography significantly improves fault

localization compared to manual inspection, particularly for identifying hotspots caused by electrical failures.

Rahman et al. [6] investigated deep learning approaches for monitoring solar panel defects using RGB imagery. The study applied convolutional neural networks to classify panel conditions such as soiling and structural damage, showing that computer vision systems can achieve high accuracy in identifying surface-level defects without human supervision. This research highlights the feasibility of visual-only inspection pipelines similar to mobile-based solutions.

Almutairi et al. [7] focused on artificial-intelligence-driven photovoltaic fault diagnosis using combined image processing and machine learning classification. Results showed that automated detection systems outperform traditional inspection approaches by providing consistent and repeatable analysis across large datasets while significantly reducing inspection time. These findings reinforce the importance of integrating AI-based detection methods into practical maintenance workflows.

The present work extends this body of research by pairing mobile-deployable classification with a dataset-centric field workflow. This combination emphasizes practical deployment: imagery can be cataloged, analyzed, summarized, and reviewed on devices without specialized hardware, persistent connectivity, or expert operators.

6. CONCLUSIONS

The current system has several limitations. The classifier dataset of 1,575 images, while sufficient to demonstrate the approach, remains small relative to production-scale requirements. Bird-Drop classification at 85.7% accuracy is still the weakest category and would benefit most from additional training data. The application also currently operates at image level, so future work is needed if region-level panel localization is required inside the mobile workflow.

Future work should include: (1) expanding the dataset with more diverse field imagery, particularly for Bird-Drop and Snow-Covered; (2) exploring lighter classifier backbones for lower-end devices; (3) adding richer dataset-management and annotation tools; and (4) collecting real-world field validation data to measure performance outside the controlled benchmark split.

SunScout demonstrates that a carefully tuned, fully on-device solar-fault analysis workflow can achieve 94.29% classification accuracy on a consumer smartphone while organizing imagery into reusable datasets for repeated review. Relative to the frozen-backbone MobileNetV2 baseline, the proposed EfficientNetB0 system improves validation accuracy by 3.50 percentage points and delivers stronger performance on the most visually ambiguous categories. The resulting mobile workflow provides a practical, accessible, and scalable alternative to traditional photovoltaic inspection and documentation routines [15].

REFERENCES

[1] Meribout, Mahmoud, et al. "Solar panel inspection techniques and prospects." Measurement 209 (2023): 112466.
[2] Walker, H. A. Best practices for operation and maintenance of photovoltaic and energy storage systems. No. NREL/TP-7A40-73822. National Renewable Energy Laboratory (NREL), Golden, CO (United States), 2018.
[3] Abubakar, Rabiu Ahmad. "Design and Construction of Solar Panel Cleaner." Engineering Science & Technology (2025): 286-300.

[4] Pierdicca, Roberto, et al. "Automatic faults detection of photovoltaic farms: solAIr, a deep learning-based system for thermal images." Energies 13.24 (2020): 6496.
[5] Henry, Chris, et al. "Automatic detection system of deteriorated PV modules using drone with thermal camera." Applied Sciences 10.11 (2020): 3802.
[6] Hamdi, Ahmed, Hassan N. Noura, and Joseph Azar. "Deep Learning-Based Approach to Automated Monitoring of Defects and Soiling on Solar Panels." Future Internet 17.10 (2025): 433.
[7] Thakfan, Ali, and Yasser Bin Salamah. "Artificial-intelligence-based detection of defects and faults in photovoltaic systems: A survey." Energies 17.19 (2024): 4807.
[8] Lu, Hongqian Karen. "Keeping your API keys in a safe." 2014 IEEE 7th International Conference on Cloud Computing. IEEE, 2014.
[9] Kim, Seok Young, et al. "Extending the onnx runtime framework for the processing-in-memory execution." 2022 International Conference on Electronics, Information, and Communication (ICEIC). IEEE, 2022.
[10] Gaffney, Kevin P., et al. "SQLite: past, present, and future." Proceedings of the VLDB Endowment 15.12 (2022).
[11] Auger, Tom, and Emma Saroyan. "Overview of the openaiapis." Generative AI for Web Development: Building Web Applications Powered by OpenAI APIs and Next. js. Berkeley, CA: Apress, 2024. 87-116.
[12] Goodhead, Dudley T. "The initial physical damage produced by ionizing radiations." International journal of radiation biology 56.5 (1989): 623-634.
[13] Bhosale, Satish Tanaji, Tejaswini Patil, and Pooja Patil. "Sqlite: Light database system." Int. J. Comput. Sci. Mob. Comput 44.4 (2015): 882-885.
[14] Gong, Youdi, et al. "A survey on dataset quality in machine learning." Information and Software Technology 162 (2023): 107268.
[15] Giner, Pau, et al. "Developing Mobile Workflow Support in the Internet of Things." IEEE Pervasive Comput. 9.2 (2010): 18-26.

©2026 By AIRCC Publishing Corporation. This article is published under the Creative Commons Attribution (CC BY) license.