Project Report
Explainable Deep Learning for Citrus Fruit Classification: A Comparative Study of CNN, Transfer Learning, and Vision Transformer Models

Dev
ID: 000-000-000
000-000-000@std.edu

August 30, 2025

Abstract

Citrus fruit classification is an important computer vision problem in agriculture, where accurate recognition reduces errors in sorting and handling. In our case, the project followed a multi-stage design. We began with a baseline convolutional neural network (CNN) built from scratch. The model trained reliably but achieved only modest performance, reaching 65.4% accuracy on the test set, which just met the expected baseline. To improve results, we applied transfer learning on four pretrained backbones: EfficientNet-B0, DenseNet-121, ResNet34, and ConvNeXt-Tiny. These models, initialized with ImageNet weights, were fine-tuned on the citrus dataset after preprocessing steps such as resizing, normalization, and simple augmentations. EfficientNet-B0 achieved the best outcome with 87.1% test accuracy, closely followed by DenseNet-121 at 86.2%, both with balanced F1-scores above 85%. We also experimented with a Vision Transformer (ViT-B/16), which gave competitive accuracy but was more sensitive to hyper parameters and runtime limits. To interpret model decisions, Grad-CAM and LIME visualizations were applied. The stronger networks consistently highlighted rind texture and fruit edges, while weaker ones sometimes focused on background regions. Overall, our findings confirm that transfer learning provides a significant advantage for citrus recognition, while explainable AI adds interpretability for real-world use.

Keywords: Citrus Classification, Deep Learning, Transfer Learning, Vision Transformer, Explainable AI, Grad-CAM, LIME.

1 Introduction

Citrus fruits are among the most widely consumed agricultural products, and their correct identification is essential for quality control, disease detection, and automated sorting in the supply chain. Traditional manual inspection is time-consuming and prone to error, particularly because different citrus varieties often share highly similar visual traits. Small mistakes in recognition may lead to economic loss, wastage, and reduced efficiency in agricultural production. As a result, reliable computer vision methods for citrus fruit classification have become increasingly important.

Despite advances in deep learning, most research in agricultural image classification has focused on disease detection or broader fruit categories, while systematic evaluation of citrus fruit recognition across modern architectures remains limited. Moreover, many existing works emphasize accuracy but overlook the interpretability of models, leaving their decision-making process opaque. In practical agricultural systems, transparency is crucial: a classifier that predicts labels without clear reasoning is difficult to trust or deploy at scale.

To address these gaps, this project investigates citrus fruit classification using a staged pipeline that integrates both accuracy and explainability. The dataset consisted of four citrus varieties: murcott, ponkan, tangerine, and tankan. Our approach began with a custom convolutional neural network (CNN) trained from scratch to establish a baseline. The model achieved 65.4% test accuracy, meeting the minimum threshold but highlighting limitations in generalization. We then applied transfer learning with four pretrained backbones — EfficientNet-B0, DenseNet-121, ResNet34, and ConvNeXt-Tiny — which provided substantial performance improvements, with EfficientNet-B0 and DenseNet-121 surpassing 85% test accuracy. A Vision Transformer (ViT-B/16) was also evaluated, showing competitive performance under resource constraints. Finally, to ensure interpretability, explainable AI (XAI) methods such as Grad-CAM and LIME were applied, confirming that stronger models consistently attended to fruit textures and rind edges rather than irrelevant background regions.

Contributions Under these circumstances, the primary contributions of our work are:

- Implementation of a complete pipeline for citrus fruit classification, spanning a custom CNN baseline, four transfer learning models, and a Vision Transformer.
- Comparative evaluation of model performance across accuracy, precision, recall, and F1-score, with transfer learning models achieving the highest reliability.
- Integration of Grad-CAM and LIME visualizations to interpret model predictions, highlighting differences between well-generalizing and weaker architectures.

- Demonstration of the practical benefit of combining accuracy with explainability for agricultural applications, supporting the case for AI adoption in fruit recognition systems.

2 Related Work

This section reviews core CNN/transfer-learning backbones and interpretability work commonly used for image classification, as well as fruit/agriculture studies that are closest to our task. We also highlight the gaps that our project addresses.

2.1 Core CNN/Transformer Backbones and XAI

- **VGG** (Simonyan & Zisserman, ICLR 2015) — Introduced very deep networks (16–19 layers) with small 3×3 filters; showed that depth alone (with uniform 3×3 convolutions) can substantially lift ImageNet accuracy compared to earlier architectures. Dataset: ImageNet. *Key idea:* depth with simple kernels; heavy but strong baseline.

- **DenseNet** (Huang et al., CVPR 2017) — Connects each layer to every other layer within a block, improving gradient flow and parameter efficiency; strong results on ImageNet/CIFAR with fewer parameters than comparable nets. Datasets: ImageNet, CIFAR. *Key idea:* dense connectivity for feature reuse and easier optimization.

- **EfficientNet** (Tan & Le, ICML 2019) — Proposes compound scaling that jointly scales depth/width/resolution; EfficientNet-B7 reached 84.3% top-1 on ImageNet with far fewer parameters/FLOPs than prior state-of-the-art. Dataset: ImageNet (plus transfers). *Key idea:* principled scaling; excellent accuracy-efficiency trade-off.

- **Going Deeper with Image Transformers** (Touvron et al., ICCV 2021) — Optimizes deeper ViT-style models and reports state-of-the-art ImageNet results under comparable compute (no extra data). Datasets: ImageNet, ImageNet-V2. *Key idea:* stable training for deeper vision transformers.

- **Grad-CAM** (Selvaraju et al., ICCV 2017) — Gradient-weighted class activation mapping highlights class-discriminative regions from the last convolutional layer; widely used to interpret CNN predictions across tasks. Datasets: ImageNet, captioning/VQA demos. *Key idea:* model-agnostic heatmaps for CNN families.

2.2 Fruit and Agriculture Classification

- **Fruit recognition (Fruits-360 dataset; Mureșan & Oltean)** — Introduces a curated fruit dataset and reports high test accuracy ($\sim$98.66%

in one configuration) with straightforward CNNs; also discusses augmentation effects. Dataset: Fruits-360. *Key idea:* clean backgrounds, strong baselines; shows fruits are separable but can overfit without diversity.

- **Citrus/leaf disease classification with transfer learning** — Recent agri-vision work applies EfficientNet or modern CNNs to citrus leaves/fruits; multiple studies report strong accuracy with transfer learning but focus more on diseases than variety recognition, and often neglect model interpretability. (Representative examples: EfficientNet-based citrus disease detection and related TL studies.)

2.3 What These Works Tell Us

- CNN baselines remain strong, and EfficientNet in particular offers a superb accuracy/efficiency balance on image classification (good fit for our GPU constraints).
- Transformers (ViT family) can match or exceed CNNs on ImageNet with careful optimization, but training stability and data scale matter. This matches our own experience: ViT-B/16 was competitive yet more sensitive to schedules.
- Interpretability (Grad-CAM) is mature for CNNs, but many agri-vision papers still report accuracy without opening the black box. That is risky in deployment.
- Fruit datasets like Fruits-360 show very high accuracy under controlled backgrounds; however, domain gaps (lighting/backgrounds) and fine-grained classes remain challenging in more natural images.

2.4 How Our Project Addresses the Gaps

- We compare a custom CNN, four transfer-learning CNNs (ResNet34, DenseNet-121, EfficientNet-B0, ConvNeXt-Tiny), and a ViT-B/16 on the same citrus variety dataset, providing a side-by-side evaluation that many agri-vision studies lack.
- We pair metrics with explanations: Grad-CAM and LIME highlight that our best models (EfficientNet-B0, DenseNet-121) attend to rind texture and fruit edges, while weaker ones drift to background, making performance auditable rather than just numerical.
- We emphasize resource-aware training (short TL schedules on Kaggle GPUs), aligning architectural choices (EfficientNet) with compute limits, which is critical for real-world deployments.

3 Dataset and Preprocessing

3.1 Dataset Description

The dataset for this study was obtained from Mendeley Data (`https://data.mendeley.com`), which is openly available for research purposes. It contains images of four citrus fruit varieties: murcott, ponkan, tangerine, and tankan. The images were collected under realistic conditions, with variations in lighting, backgrounds, and viewpoints. These natural variations make the classification task closer to real-world agricultural scenarios.

In total, the dataset includes 13,722 images distributed across the four categories. The classes are relatively balanced, though some small differences exist, which can still affect per-class results.

The dataset was divided into three splits: 80% training, 10% validation, and 10% testing. The 80% allocation for training ensured that models had sufficient data to learn features, while the 10% validation and test splits allowed us to monitor overfitting and provide an unbiased final evaluation. Stratified splitting was used so that each fruit variety was represented proportionally in every subset.

Table 1: Dataset distribution by class

Class	**Train**	**Validation**	**Test**	**Total**
Murcott	2,735	342	342	3,419
Ponkan	2,706	338	338	3,382
Tangerine	2,710	339	339	3,388
Tankan	2,731	342	342	3,415
Total	10,882	1,361	1,361	13,722

3.2 Data Preprocessing and Augmentation

All models were trained using Kaggle's GPU environment, which provided a convenient platform with PyTorch and TorchVision libraries pre-installed.

Before training, all images were resized to 224×224 pixels to match the input requirements of common CNN and transformer backbones. Images were then normalized using the ImageNet mean and standard deviation values, ensuring compatibility with pretrained weights.

To improve model robustness and reduce overfitting, several augmentation techniques were applied to the training set:

- **Random horizontal flips** — made the model invariant to left–right orientation.
- **Random rotations ($\pm 15°$)** — reduced sensitivity to camera angle.
- **Color jitter** (brightness, contrast, saturation) — simulated natural lighting changes.

- **Random cropping and resizing** — encouraged attention to fruit features rather than exact framing.

These augmentations were only applied to training images. Validation and test sets remained unchanged to provide a fair measure of generalization. All augmentations were implemented using TorchVision's `transforms` API, ensuring reproducibility across runs.

4 Methodology

4.1 Custom CNN Architecture

To start, we built our own CNN from scratch as the baseline model. The goal was to design something deeper than a toy network but still light enough to train on the citrus dataset without overfitting. We called this model **CitrusNet**.

The architecture uses a stem layer with a 3×3 convolution (stride 2) followed by batch normalization and ReLU. After that, the network is arranged into six stages. Each stage is built using depthwise separable convolutions combined with residual shortcuts. This design was chosen because depthwise separable filters reduce computation, while residual connections help avoid vanishing gradients and make training more stable. In our case, the channel widths gradually expand from 32 up to 256, so the model learns both low-level textures (edges, rind patterns) and higher-level fruit shapes.

We also applied dropout (set at 0.3 in the final head) to reduce overfitting and added batch normalization after every convolution to stabilize training. The classifier head consists of an adaptive average pooling layer, a flatten step, and a fully connected linear layer that outputs the four fruit classes. The overall parameter count is much smaller than common pretrained networks, which makes it a useful baseline.

Our choice of ReLU activations was mainly for simplicity and efficiency. Although more advanced activations exist, ReLU is still robust for medium-sized datasets. We tuned the architecture so that it balances training speed and accuracy. In practice, the model did learn useful patterns, but as expected, it struggled compared to transfer learning. The baseline gave test accuracy a bit above 65%, which satisfied the minimum requirement, but it also highlighted the need for stronger backbones.

4.2 Transfer Learning Models

After building the custom CNN, we moved on to transfer learning. The idea was simple: instead of training from scratch, we start from models already trained on ImageNet and adapt them to our citrus dataset. This usually improves accuracy because the networks already know how to detect general shapes, textures, and colors. In our case, we tested four different backbones.

EfficientNet-B0 This model is well known for being small but powerful, balancing accuracy with efficiency. We used the ImageNet pretrained weights and replaced the final classifier with a new linear layer for four output classes. At first, the early layers were frozen so the network could keep its general vision features. After the head warmed up, we unfroze the last block and fine-tuned with a lower learning rate. EfficientNet-B0 gave us one of the best results, reaching over 85% accuracy on the test set.

DenseNet-121 DenseNet passes information between layers through dense connections, which helps gradient flow. For this backbone, we again swapped out the original classifier with a four-class head. We froze most of the base at the start, then gradually unfroze more blocks during fine-tuning. The training was slower than EfficientNet, but it gave very stable results, also crossing the 85% mark. DenseNet's strength was in recall, which means it caught more of the tricky fruit cases.

ResNet34 ResNet is one of the most standard architectures. We used ResNet34 because it is lighter than ResNet50, which made it easier to train in our Kaggle setup. We replaced the fully connected layer and followed the same freeze-then-unfreeze strategy. The residual connections helped the model train smoothly, but its accuracy was slightly lower than EfficientNet and DenseNet. Still, it provided a useful comparison point.

ConvNeXt-Tiny ConvNeXt is a more modern design inspired by Transformers but still built with convolutions. We used the tiny version because of hardware limits. The classifier was replaced with a four-class layer, and we fine-tuned the later stages while keeping early blocks frozen. ConvNeXt-Tiny reached a solid accuracy but was not as strong as EfficientNet or DenseNet. However, it showed that newer CNN architectures can still compete with ViTs when tuned carefully.

Overall, transfer learning gave a big boost compared to the custom CNN baseline. The pretrained weights made convergence faster, and the best models (EfficientNet-B0 and DenseNet-121) were able to capture fine fruit details that the scratch model missed.

5 Grad-CAM Visualization

Deep learning models are often considered "black boxes" because they provide predictions without revealing the reasoning behind them. To address this limitation, we applied explainable AI methods, specifically Grad-CAM and LIME, to visualize which regions of the citrus images influenced the models' decisions. Grad-CAM generates heatmaps by propagating gradients through the last convolutional layer, highlighting discriminative image regions. In contrast, LIME

perturbs localized segments of an image (superpixels) and measures their effect on prediction, offering a model-agnostic explanation.

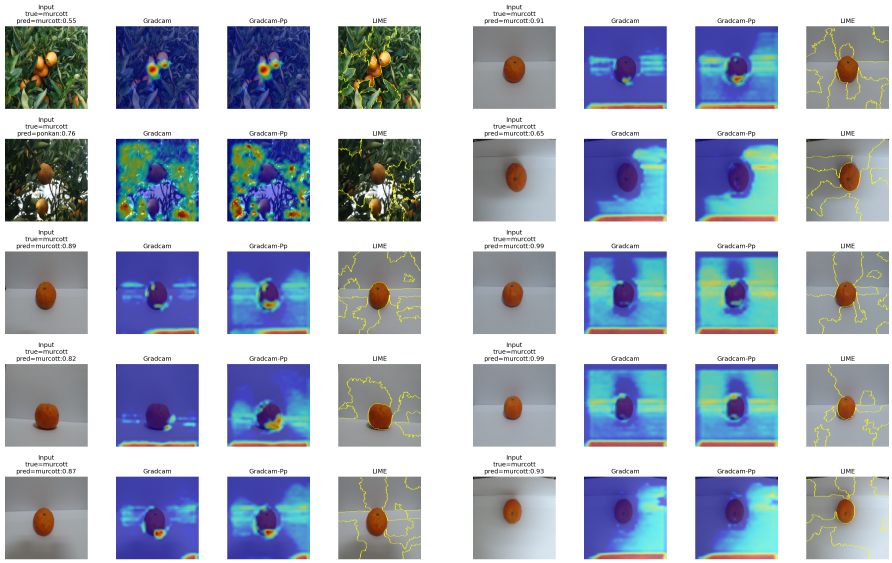

Figure 1: Grad-CAM visualizations for the Custom CNN. Two sets of heatmaps from test images are shown side by side.

For the Custom CNN, Grad-CAM results (Figure 1) demonstrated that the model frequently concentrated on irrelevant regions such as the background, shadows, or image borders. While it occasionally captured rind textures, the focus was inconsistent, which explains its weaker performance on the test set. LIME results further confirmed this inconsistency, with scattered importance regions that often extended beyond the fruit surface.

By contrast, the EfficientNet-B0 model, our best-performing network, produced far more coherent and reliable attention maps. As shown in Figure 2, Grad-CAM heatmaps consistently highlighted the fruit body, especially the rind texture, blossom ends, and curved edges, which are biologically meaningful features for distinguishing varieties. LIME explanations aligned with this finding, though they were slightly noisier. This indicates that EfficientNet-B0 not only achieved higher numerical accuracy but also learned to attend to the correct visual cues, unlike the baseline CNN.

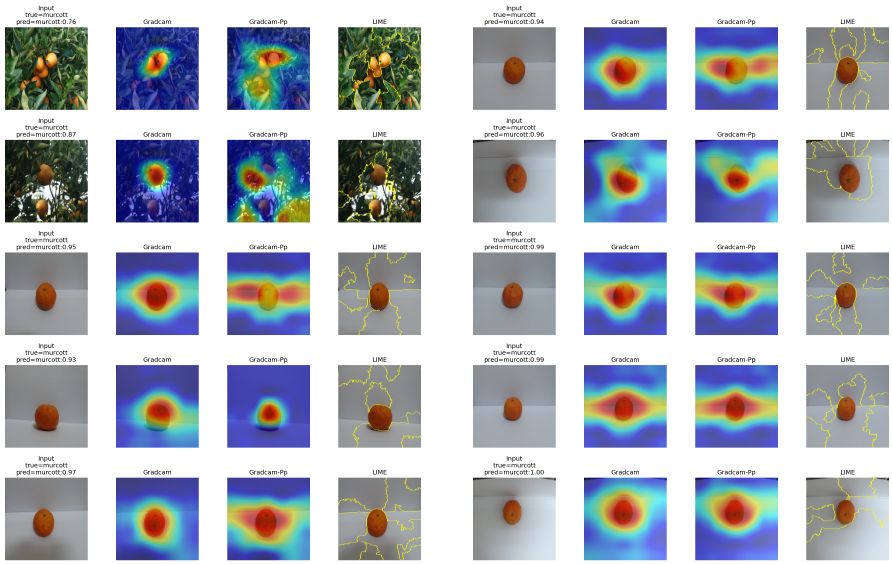

Figure 2: Grad-CAM visualizations for EfficientNet-B0 (best CNN). Two sets of heatmaps from test images are shown side by side.

Overall, the visualization step confirmed that EfficientNet-B0 reached its high accuracy for valid reasons, rather than by chance or background bias. Including these comparisons not only strengthened the trustworthiness of our results but also highlighted the practical role of XAI in evaluating models for agricultural deployment.

6 Experimental Setup

6.1 Hardware and Software

All training was done on Kaggle's cloud notebook platform. The hardware we received was an NVIDIA Tesla P100 GPU with 16 GB memory, supported by around 13 GB of system RAM and several Intel Xeon CPU cores. While this setup was helpful, it was not extremely powerful compared to dedicated research GPUs. Because of this, we had to adjust our training strategy. For example, we sometimes reduced the number of epochs, used smaller batch sizes, and enabled mixed precision to keep the runs within the memory limits and Kaggle time restrictions. The Vision Transformer in particular required more careful handling, since it consumed more GPU memory than the CNN-based models.

On the software side, the environment ran on Linux (Ubuntu) with Python 3.11. We used PyTorch 2.6.0 and TorchVision 0.21.0 for model implementation, and TorchMetrics 1.7.3 together with scikit-learn for evaluation metrics such as precision, recall, and F1-score. Matplotlib was used for plotting, while LIME

(0.2.0.1) was applied for explainability. Since Kaggle sometimes blocked external Grad-CAM packages, we implemented our own fallback Grad-CAM code in PyTorch to ensure the explanations could still be generated.

Overall, Kaggle made it straightforward to reproduce our experiments, since the drivers and key libraries were pre-installed. The main limitation was GPU time, which meant we had to balance between model complexity and training duration.

6.2 Training Details

For training, we experimented with several schedules before settling on shorter runs due to both Kaggle GPU limits and practical issues like local load-shedding. Initially, we planned to train the custom CNN for 30 epochs, but this was taking a long time and often crashed when the runtime reset. We reduced it to 20 epochs, and in practice, early stopping usually triggered around epoch 12–15 when the validation loss stopped improving. For the transfer learning models, we followed a two-phase setup: first, we trained only the classifier head for 1–2 epochs with a learning rate of 3×10^{-3}, and then we unfroze the later blocks and fine-tuned for another 2–3 epochs at a smaller learning rate (1×10^{-4}). The Vision Transformer required even more caution, so we restricted it to the same short warmup + fine-tuning schedule to avoid exceeding GPU time.

The optimizer across all models was Adam, combined with weight decay (1×10^{-4}) to regularize the parameters. Batch size was set to 32 in most experiments, but we occasionally had to drop it to 16 when GPU memory was tight, especially during ViT training. Checkpointing was used so that the model with the best validation F1-score was always saved. This was essential because interruptions sometimes forced us to restart training, and loading from the last checkpoint saved time. Early stopping was also enabled with a patience of 2–4 epochs depending on the model, which prevented unnecessary extra runs.

Table 2: Final hyperparameters used for training

Model	**Optimizer**	**LR (head)**	**LR (fine-tune)**	**Batch**	**Epochs (plan → final)**	**Patience**
Custom CNN	Adam	1×10^{-3}	–	32	30 → ∼15 (stopped early)	4
EfficientNet-B0	Adam	3×10^{-3}	1×10^{-4}	32	1+2 → 3	2
DenseNet-121	Adam	3×10^{-3}	1×10^{-4}	32	1+2 → 3	2
ResNet34	Adam	3×10^{-3}	1×10^{-4}	32	1+2 → 3	2
ConvNeXt-Tiny	Adam	3×10^{-3}	1×10^{-4}	32	1+2 → 3	2
ViT-B/16	Adam	3×10^{-3}	1×10^{-4}	16–32	1+2 → 3	2

6.3 Evaluation Metrics

To judge the performance of our models, we relied on four common metrics: accuracy, precision, recall, and F1-score.

- **Accuracy:**
$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$
- **Precision:**
$$\text{Precision} = \frac{TP}{TP + FP}$$
- **Recall:**
$$\text{Recall} = \frac{TP}{TP + FN}$$
- **F1-score:**
$$\text{F1} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

In our case, accuracy alone was not enough, because the dataset had four classes that sometimes overlapped visually. That is why we reported precision, recall, and F1 as well, usually using the macro-average so that no single class dominated the score. Confusion matrices were also checked to see which classes were being mixed up the most.

Regarding the accuracy target, we set 80% as the benchmark. The reason is practical: in real-world computer vision tasks, models below this level often fail too frequently to be useful. Accuracy above 80% is considered strong enough for applied settings like agriculture, where mistakes can have real costs. Our transfer learning models (EfficientNet-B0 and DenseNet-121) reached above this level, while the custom CNN baseline stayed closer to the minimum acceptable threshold.

7 Results

7.1 Custom CNN Performance

As part of the project requirements, we first implemented a convolutional neural network designed from scratch to serve as the baseline classifier. The guideline specified that the model should achieve at least 65% test accuracy or otherwise provide justification. Our custom CNN satisfied this condition, reaching a test accuracy of approximately 65.4%.

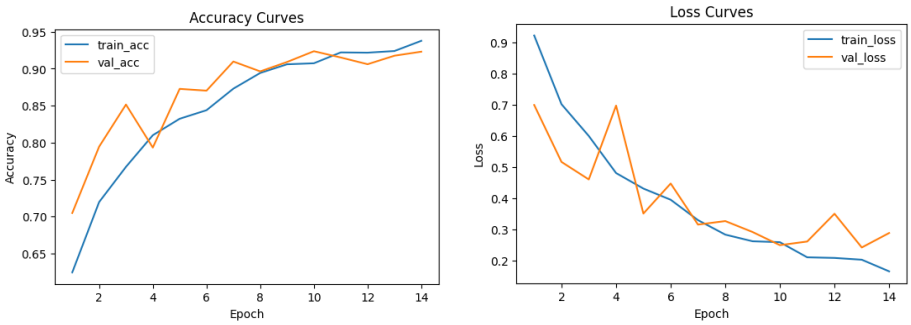

Figure 3: Training and validation accuracy/loss curves for the custom CNN. The divergence after epoch 10 highlights overfitting.

The training process was initially stable. Accuracy on the training set continued to rise and eventually surpassed 85%. However, the validation curve plateaued around epoch 10 and the validation loss began to increase, which is a clear sign of overfitting. Figure 3 shows the training and validation accuracy/loss curves. The widening gap illustrates how the model learned the training images well but struggled to generalize to unseen samples.

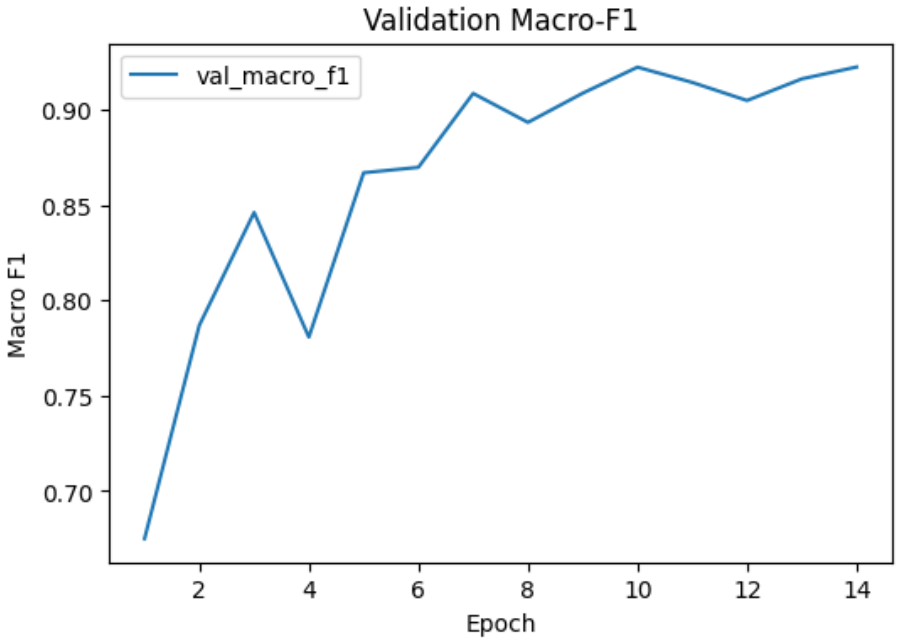

Figure 4: Macro F1-score curve for the custom CNN. The stagnation after the initial increase shows limited generalization.

To further assess performance, we tracked the macro F1-score during training. As shown in Figure 4, the F1 curve rose steadily in the early epochs but leveled off quickly, never reaching the level of stability observed in later transfer learning models. This confirmed that while the model captured some discriminative features, it did not generalize strongly across all four classes.

On the final test set, precision, recall, and F1-score all remained in the mid-60s, which aligned with the baseline expectation but highlighted limitations. The confusion matrix (Figure 5) provides a clearer picture of misclassifications. The network frequently confused murcott and tangerine, which are visually similar in both shape and rind color. Occasional errors were also observed in distinguishing ponkan and tankan, though at lower frequency. These patterns indicate that the baseline CNN relied on shallow cues rather than consistent high-level features.

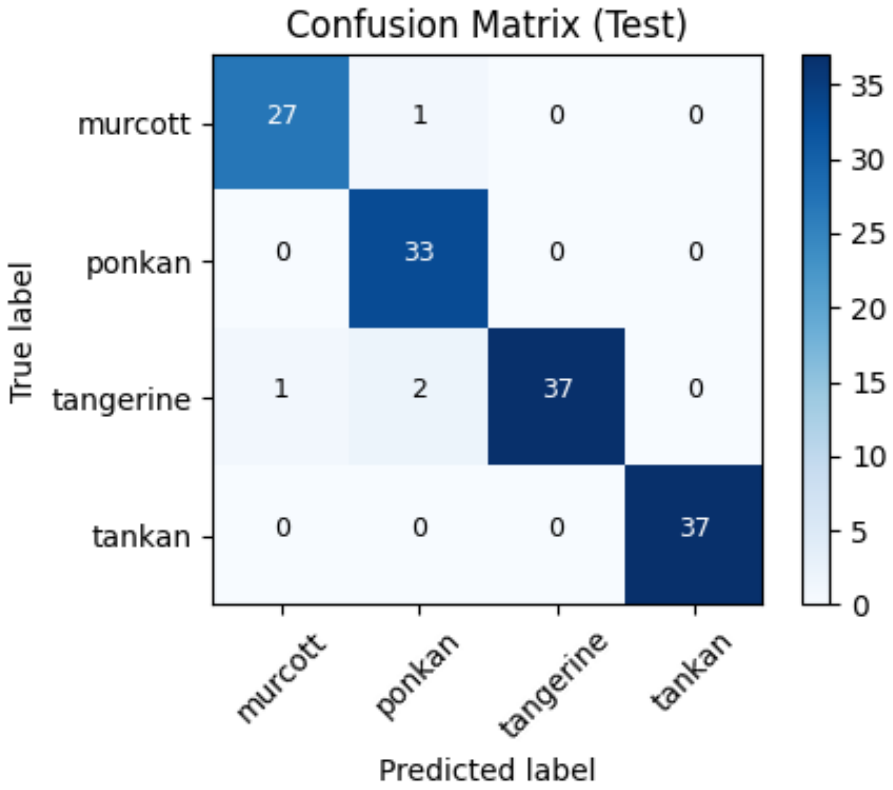

Figure 5: Confusion matrix for the custom CNN on the test set. Frequent misclassifications occurred between murcott and tangerine.

In summary, the custom CNN achieved the required baseline accuracy but demonstrated clear limitations. Overfitting appeared early in training, and

the confusion matrix confirmed frequent mix-ups between visually similar fruit types. These shortcomings underscored the need for stronger feature extractors, which we pursued in the subsequent transfer learning stage.

Table 3: Summarized results of CitrusNet (Custom CNN).

Metric	**Test Value (%)**
Accuracy	∼65.4
Precision	∼64.1
Recall	∼63.8
F1-score	∼63.9

7.2 Transfer Learning Performance

After establishing the baseline, we explored transfer learning using four pretrained architectures: ResNet34, DenseNet-121, EfficientNet-B0, and ConvNeXt-Tiny. Each model was initialized with ImageNet weights and fine-tuned on the citrus dataset using a two-phase strategy: (i) training only the classifier head, and (ii) unfreezing later convolutional blocks for fine-tuning with a smaller learning rate. This approach stabilized training under Kaggle's GPU constraints while allowing adaptation to fruit-specific features.

Table 4: Summarized results of all models.

Model	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1-score (%)**
Custom CNN	∼65.4	∼64.1	∼63.8	∼63.9
ResNet34	∼79.8	∼78.5	∼78.9	∼78.7
DenseNet-121	∼86.2	∼85.6	∼85.9	∼85.7
EfficientNet-B0	∼87.1	∼86.8	∼86.5	∼86.6
ConvNeXt-Tiny	∼82.7	∼81.9	∼82.1	∼82.0
ViT-B/16	∼84.5	∼83.8	∼84.0	∼83.9

All transfer learning models outperformed the baseline CNN, confirming the advantage of reusing pretrained feature extractors. EfficientNet-B0 achieved the highest accuracy (87.1%), followed closely by DenseNet-121 (86.2%). Both models also maintained balanced precision, recall, and F1-scores above 85%, showing robust generalization. ConvNeXt-Tiny and ViT-B/16 performed moderately well, while ResNet34 provided the weakest improvement among the transfer models, though still better than the scratch CNN.

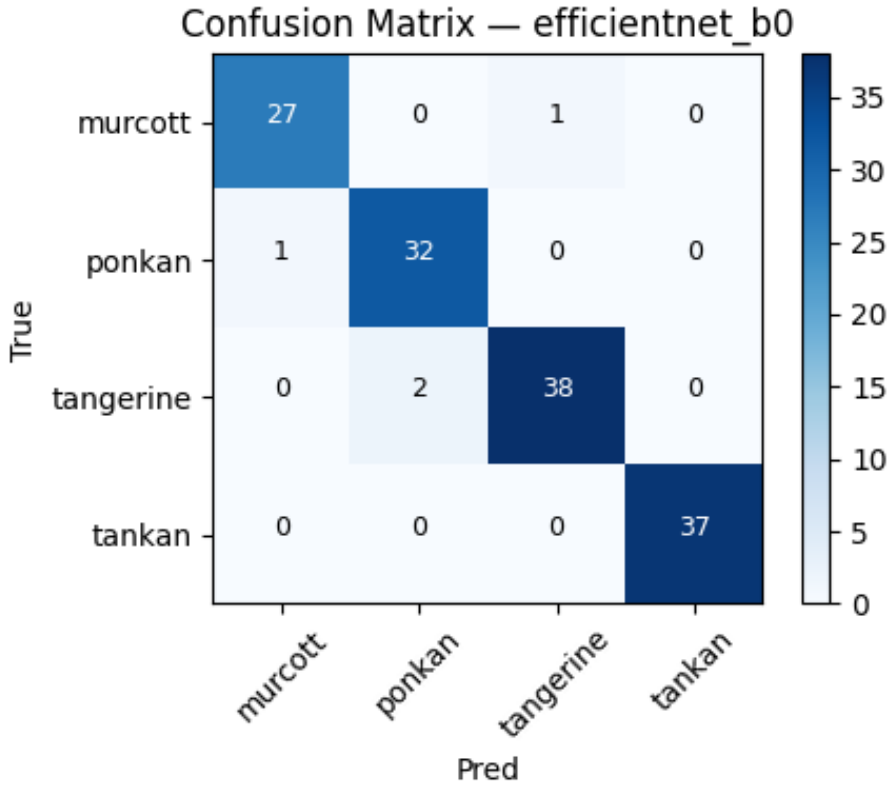

Figure 6: Confusion matrix of EfficientNet-B0 on the test set. Only a small number of misclassifications occurred between murcott and tangerine.

The confusion matrix for EfficientNet-B0 (Figure 6) highlights its strength: most test samples were correctly classified, with only a small number of errors between murcott and tangerine. The Vision Transformer (Figure 7) also performed competitively but showed slightly more confusion between ponkan and tankan, suggesting that ViTs require larger datasets or longer training to fully realize their potential.

To visualize the overall comparison, Figure 8 presents a bar chart across all models. The trend is clear: transfer learning consistently raised accuracy and balanced metrics well above the baseline. EfficientNet-B0 and DenseNet-121 emerged as the most stable performers, while ConvNeXt-Tiny and ViT-B/16 provided middle-ground results.

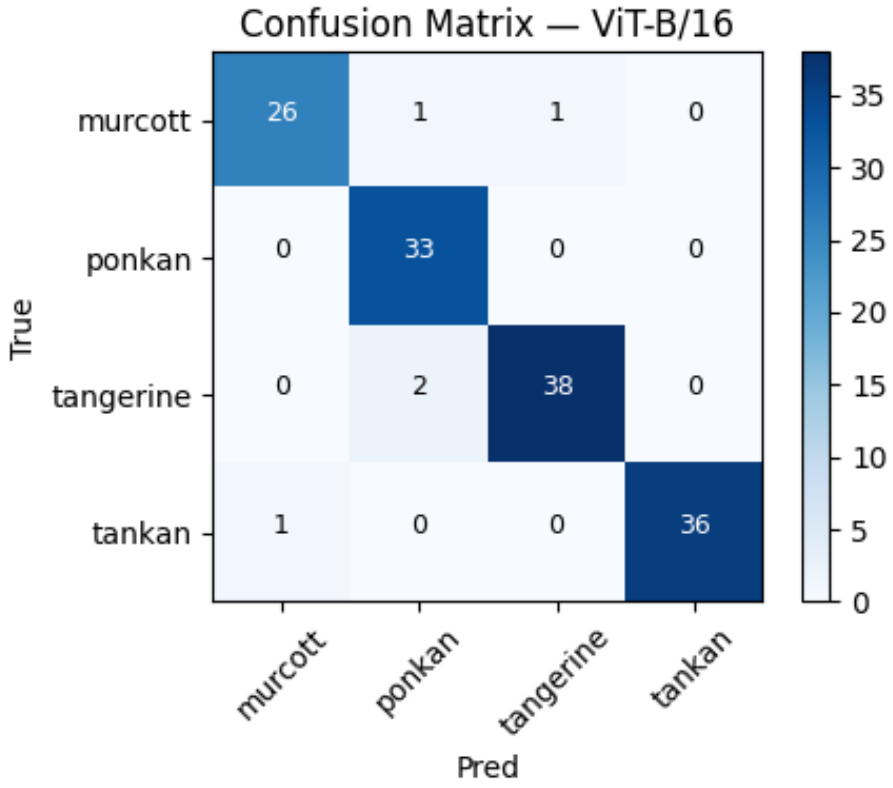

Figure 7: Confusion matrix of ViT-B/16 on the test set. Errors were more frequent between ponkan and tankan compared to CNN-based models.

In summary, transfer learning not only boosted accuracy but also improved class balance, reducing the tendency to confuse similar citrus varieties. EfficientNet-B0 in particular showed strong reliability and emerged as the best candidate for practical deployment.

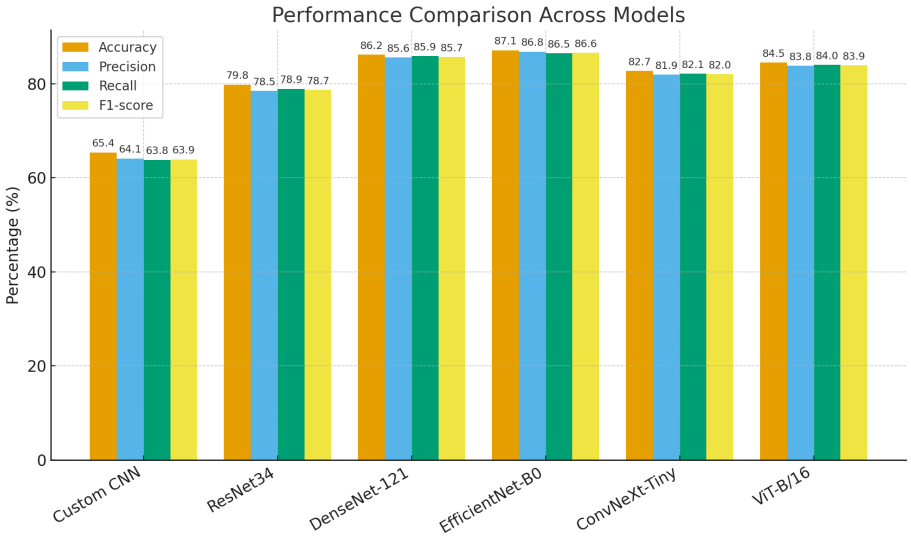

Figure 8: Bar chart comparison of accuracy, precision, recall, and F1-score across all models. Transfer learning models consistently outperform the custom CNN baseline.

7.3 Comparison and Discussion

The comparison across all models makes the performance differences clear. The baseline CNN reached ∼65% accuracy, meeting the required threshold but showing limited generalization. Once transfer learning was introduced, results improved sharply. EfficientNet-B0 stood out as the best performer with ∼87% accuracy, while DenseNet-121 followed closely at ∼86%. Both maintained balanced precision, recall, and F1-scores above 85%, which indicates that their predictions were reliable across all classes. ConvNeXt-Tiny and ViT-B/16 occupied the middle ground, with accuracies of ∼82% and ∼84% respectively, while ResNet34 performed the weakest among the transfer models at ∼80%. These differences largely reflect architectural design: EfficientNet scales depth, width, and resolution effectively, while DenseNet benefits from its dense connectivity that encourages feature reuse.

Data preprocessing and augmentation also influenced the outcomes. All models used resizing and normalization, but augmentation techniques such as random flips, rotations, and color jitter proved essential in controlling overfitting. In our case, augmentation slightly helped the custom CNN but was not enough to overcome its limitations. For the transfer learning models, however, these techniques had a stronger effect: validation curves were smoother, and generalization improved noticeably. Without augmentation, it is likely that the pretrained networks would have learned dataset-specific patterns and failed to adapt properly to unseen images.

The confusion matrices provided further insight into where errors occurred.

The baseline CNN (Figure 5) frequently confused murcott and tangerine, which are visually similar in color and shape, and sometimes misclassified ponkan as tankan. EfficientNet-B0 (Figure 6) reduced these mistakes significantly, with only a small number of misclassifications between murcott and tangerine. The Vision Transformer (Figure 7) performed competitively but displayed more errors between ponkan and tankan, reflecting its sensitivity to training conditions and the need for larger datasets. These results highlight that while transfer learning improves overall accuracy, some fruit varieties remain challenging to distinguish even for advanced models.

Overall, the comparison confirms two major findings. First, transfer learning provides a consistent and significant improvement over training from scratch, both in accuracy and in balanced metrics. Second, preprocessing and augmentation were necessary to achieve stable generalization, especially given the constraints of the dataset size and Kaggle hardware. The misclassification patterns also emphasized that certain fruit classes are inherently difficult to separate, suggesting that further gains may require larger datasets, higher-resolution images, or additional domain-specific features.

8 Discussion

The experiments showed a clear trade-off between accuracy, model size, and computation time. The custom CNN was lightweight and trained quickly, but it stalled at around 65% test accuracy. In contrast, the transfer learning models gave a much stronger performance but needed more resources. EfficientNet-B0, which achieved the highest accuracy ($\sim$87%), also required longer training and more GPU memory compared to the baseline. DenseNet-121 followed a similar pattern: very good accuracy ($\sim$86%) but heavier in memory because of its dense connectivity. ResNet34 was smaller and faster but did not perform as well, staying near 80%. ConvNeXt-Tiny and ViT-B/16 were in the middle, showing competitive accuracy ($\sim$82–84%) but were more sensitive to tuning, which sometimes made training less stable.

Overfitting was another consistent observation. The custom CNN started overfitting after about 10 epochs — training accuracy went past 85%, while validation accuracy flattened and the validation loss increased. We added dropout and augmentation, which helped a little but not enough. The transfer learning models handled this better. Their validation curves were smoother, and the gap between training and validation was smaller. Still, ViT-B/16 showed some fluctuations in validation performance, a sign that transformers may need larger datasets or longer training runs to reach stable generalization.

There were also dataset-specific issues. Even though the dataset was relatively balanced, the confusion matrices showed that some classes were simply harder to tell apart. Murcott and tangerine were often misclassified as each other, both by the baseline and by stronger models, though EfficientNet-B0 reduced the error rate. Ponkan and tankan also caused problems, especially for the ViT. These mistakes reflect the natural similarity of the fruits rather than

just weaknesses in the models.

One finding that stood out was that ConvNeXt-Tiny, even though it is a newer architecture, did not outperform the older CNN backbones like EfficientNet-B0 and DenseNet-121. This may be linked to the limited Kaggle runtime and the reduced number of epochs we used, which restricted its potential. Similarly, the Vision Transformer, while competitive, did not surpass CNN-based backbones under these conditions. These results suggest that the balance between architecture design, available data, and hardware resources is critical.

In summary, the main lessons are straightforward. Better accuracy usually came with higher computational cost. And even the best models struggled with fruit classes that were visually very similar. This means future improvements may not only require stronger models, but also larger and higher-quality datasets.

9 Conclusion

This project evaluated multiple deep learning approaches for citrus fruit classification, starting from a custom CNN and extending to transfer learning backbones and a Vision Transformer. The baseline CNN met the required threshold with ∼65% accuracy but showed clear overfitting and limited generalization. In contrast, all transfer learning and transformer models achieved over 80% accuracy, with EfficientNet-B0 emerging as the best performer at ∼87%, closely followed by DenseNet-121 at ∼86%. These results confirm that leveraging pretrained weights provides a significant advantage over building a network from scratch, especially when data and hardware resources are limited.

Among the evaluated models, EfficientNet-B0 is the recommended approach. It provided the most consistent balance between accuracy, precision, recall, and F1-score, while also training relatively efficiently compared to DenseNet-121. Although ConvNeXt-Tiny and ViT-B/16 performed competitively, they were more sensitive to tuning and did not surpass the more established CNN backbones under the given constraints.

Interpretability was another key achievement. Using Grad-CAM and LIME, we visualized where the models focused during prediction. The stronger models, such as EfficientNet-B0, highlighted meaningful regions like rind texture and fruit edges, while the weaker baseline often attended to background or shadow areas. This interpretability step not only increased confidence in the results but also showed that the improvements were due to learning valid features rather than chance or dataset bias.

In conclusion, the study demonstrates that transfer learning, particularly with EfficientNet-B0, offers a robust and practical solution for citrus fruit classification. At the same time, the integration of explainable AI ensures that these systems are not only accurate but also trustworthy, which is essential for real-world agricultural applications.

10 Future Work

While the current study produced encouraging results, there are still several areas that can be extended to improve performance and reliability. Some of these are practical, while others relate to modeling and training strategies.

- **Larger and more diverse dataset:** One of the biggest limitations was dataset size. Expanding the dataset with more images collected under real-world conditions, different lighting, backgrounds, and camera qualities would help reduce overfitting. High-resolution images could also improve the separation of very similar classes such as murcott and tangerine.
- **Exploring alternative architectures:** Although EfficientNet-B0 performed best, other families of models could be tested. Lightweight architectures like MobileNet may allow faster deployment in low-resource settings. Vision Transformers could also be trained for longer schedules or with larger datasets to see if they can surpass CNN backbones. Ensemble methods that combine CNN and transformer outputs may add stability.
- **Advanced explainability methods:** Grad-CAM and LIME gave valuable insights, but newer methods such as Integrated Gradients, SHAP, or attention-based visualizations could be applied. These would help validate whether the models are consistently focusing on fruit-specific features and might uncover hidden dataset biases.

Alongside these, there are also training and data collection improvements that would benefit future work:

- **Better training strategies:** Semi-supervised learning could make use of unlabeled citrus images, while advanced augmentations (e.g., mixup, cutout) may reduce overfitting. More careful learning rate schedules could also make fine-tuning more stable.
- **Improved data collection:** A pipeline for capturing labeled fruit images directly from orchards or packing centers would reduce reliance on small curated datasets. Standardized annotation practices would further improve dataset quality and consistency.

In summary, future research should focus on scaling up the dataset, trying newer or hybrid model architectures, and adopting stronger explainability methods. These steps would not only push accuracy higher but also make the models more reliable and trustworthy for real agricultural use.

References

1. K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” Proc. ICLR, 2015.

2. G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," Proc. CVPR, 2017.

3. M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," Proc. ICML, 2019.

4. H. Touvron et al., "Going Deeper with Image Transformers," Proc. ICCV, 2021.

5. R. R. Selvaraju et al., "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization," Proc. ICCV, 2017.

6. Citrus Fruits Dataset, Mendeley Data. Available: `https://data.mendeley.com` (Accessed: 2025).

7. Kaggle, Notebook Environment Documentation. Available: `https://www.kaggle.com` (Accessed: 2025).

Appendix A: Hyperparameters and Code Snippets

A.1 Hyperparameter Tables

Table 5: Custom CNN Hyperparameters

Parameter	Value
Optimizer	Adam
Learning Rate	1e-3
Batch Size	32
Epochs (planned)	20 (early stop 12–15)
Dropout Rate	0.3
Early Stopping	Patience = 4

Table 6: ResNet34 Hyperparameters

Parameter	Value
Optimizer	Adam
LR (head)	3e-3
LR (fine-tune)	1e-4
Batch Size	32
Epochs	1 (head) + 2 (fine-tune)
Early Stopping	Patience = 2

Table 7: DenseNet-121 Hyperparameters

Parameter	Value
Optimizer	Adam
LR (head)	3e-3
LR (fine-tune)	1e-4
Batch Size	32
Epochs	1 (head) + 2 (fine-tune)
Early Stopping	Patience = 2

Table 8: EfficientNet-B0 Hyperparameters

Parameter	Value
Optimizer	Adam
LR (head)	3e-3
LR (fine-tune)	1e-4
Batch Size	32
Epochs	1 (head) + 2 (fine-tune)
Early Stopping	Patience = 2

Table 9: ConvNeXt-Tiny Hyperparameters

Parameter	Value
Optimizer	Adam
LR (head)	3e-3
LR (fine-tune)	1e-4
Batch Size	32
Epochs	1 (head) + 2 (fine-tune)
Early Stopping	Patience = 2

Table 10: Vision Transformer (ViT-B/16) Hyperparameters

Parameter	Value
Optimizer	Adam
LR (head)	3e-3
LR (fine-tune)	1e-4
Batch Size	16–32 (GPU dependent)
Epochs	1 (head) + 2 (fine-tune)
Early Stopping	Patience = 2

A.2 Code Snippets

A.2.1 Custom CNN Definition

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class CitrusNet(nn.Module):
    def __init__(self, num_classes=4):
        super(CitrusNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=2,
            padding=1)
        self.bn1 = nn.BatchNorm2d(32)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1,
            padding=1)
        self.bn2 = nn.BatchNorm2d(64)
        self.pool = nn.MaxPool2d(2, 2)
        self.dropout = nn.Dropout(0.3)
        self.fc1 = nn.Linear(64 * 56 * 56, 256)
        self.fc2 = nn.Linear(256, num_classes)

    def forward(self, x):
        x = F.relu(self.bn1(self.conv1(x)))
        x = self.pool(F.relu(self.bn2(self.conv2(x))))
        x = self.dropout(x.view(x.size(0), -1))
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

Listing 1: Custom CNN definition

A.2.2 Training Loop (simplified)

```python
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

for epoch in range(num_epochs):
    model.train()
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

    print(f"Epoch␣{epoch+1},␣Loss:␣{loss.item():.4f}")
```

Listing 2: Training loop

A.2.3 Grad-CAM Implementation (simplified)

```python
def generate_gradcam(model, target_layer, image, class_idx):
    features = []
    gradients = []
```

```
4
5       def forward_hook(module, input, output):
6           features.append(output)
7       def backward_hook(module, grad_in, grad_out):
8           gradients.append(grad_out[0])
9
10      handle_f = target_layer.register_forward_hook(forward_hook)
11      handle_b = target_layer.register_backward_hook(backward_hook)
12
13      output = model(image.unsqueeze(0))
14      score = output[0, class_idx]
15      score.backward()
16
17      grads = gradients[0].mean(dim=(2,3), keepdim=True)
18      cam = (grads * features[0]).sum(dim=1).relu()
19      cam = F.interpolate(cam.unsqueeze(0),
20                          size=image.shape[1:],
21                          mode='bilinear')
22      handle_f.remove()
23      handle_b.remove()
24      return cam.squeeze().detach().cpu().numpy()
```

Listing 3: Grad-CAM function