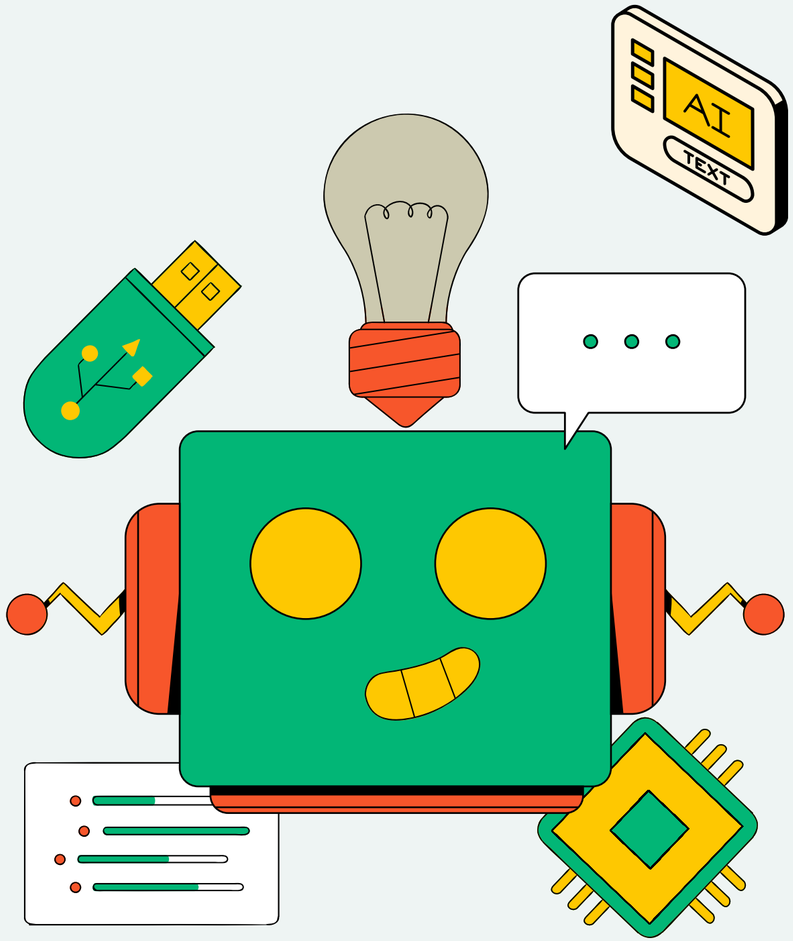
THYNK UNLIMITED

WE LEARN FOR THE FUTURE

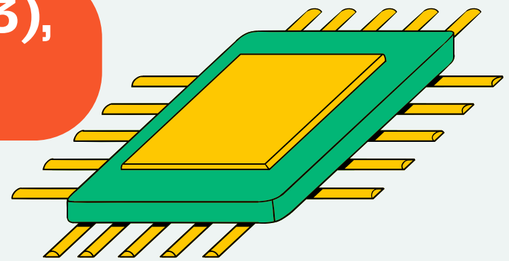


ET0732 MINI PROJECT

4-CLASS CLASSIFIER: PEACH, BROWN ONION, PURPLE ONION, UNKNOWN

PRESENTED BY:

TNG KHAI CHUN TERENCE (P2425713),
JUSTIN LUI JUN WEI (P2425672)



PRESENTATION OUTLINE

1. Use case and project goals
2. Dataset build: proxy to real, challenges, and final composition
3. Data preparation and augmentation strategy
4. Transfer learning models and fair comparison setup
5. Custom CNN evolution (v1 → v4) and what changed each time
6. Model selection using balanced metrics (Balanced Acc, Macro F1, Kappa)
7. Hyperparameter tuning (Hyperband + Macro–Micro strategy)
8. Split comparison (stratified vs manual, deployment realism)
9. Mobile deployment (TensorFlow Lite app pipeline)
10. Final results and key takeaways

FOCUS: DECISIONS, EVIDENCE, AND DEPLOYMENT READINESS.

USE CASE (PROBLEM STATEMENT → SOLUTION)

PROBLEM:

- Ingredient sorting is error-prone when items look similar (brown vs purple onion)
- Real photos vary by lighting, background, distance, and angle
- A safe system must reject “other” objects, not force a wrong label

- A 4-class image classifier that identifies:
  - **Peach**
  - **Brown onion**
  - **Purple onion**
  - **Unknown** (reject non-target objects and empty backgrounds)
- Trained with **transfer learning + custom CNNs**, then systematically tuned
- Exported to **TensorFlow Lite** for fast, portable mobile inference

PROJECT GOALS AND SUCCESS CRITERIA

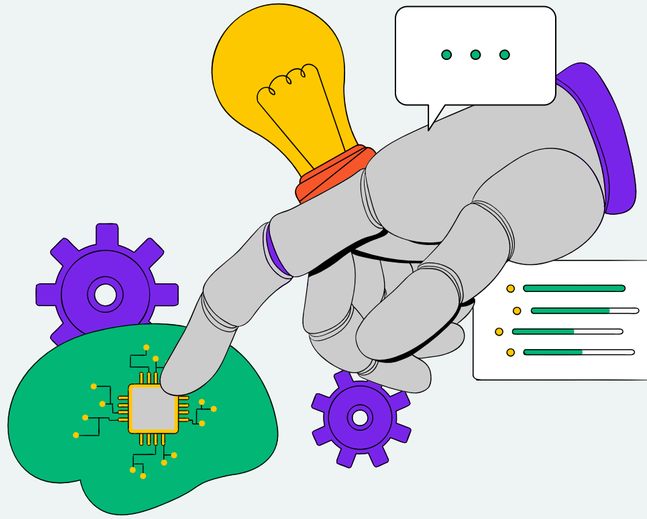
- Build a reliable **4-class** classifier with strong real-world generalisation
- Compare pretrained backbones for **accuracy vs efficiency**

- Design and improve a custom CNN from **baseline → attention-based model**

- Use **evidence-based optimisation** (tuning, not guesswork)

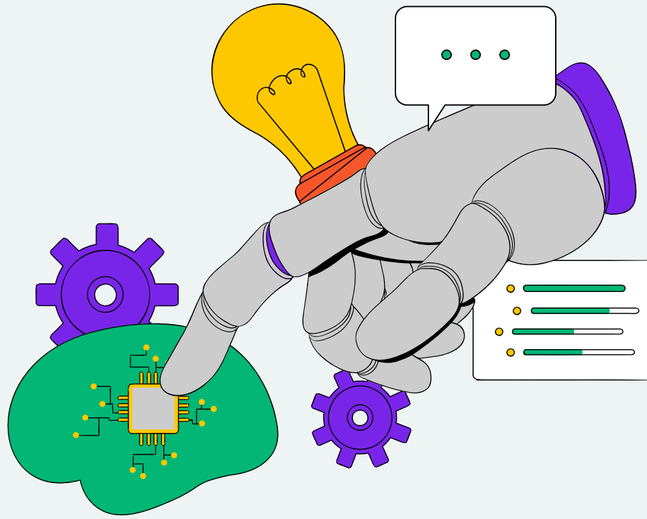
- Prove deployment readiness via **TFLite conversion + low-latency inference**

DATASET SETUP: PROXY DATASET (FLOWERS)



- We started with a proxy dataset because our Peach/Onion dataset was incomplete
- Flowers validated the full pipeline: **loading, splits, augmentation, training, evaluation**
- 4 classes used (Tulips removed) with **70/15/15** split
- **Key benefit**: if performance drops on real data, the likely cause is dataset realism, not broken code

TRANSITION TO OUR REAL DATASET

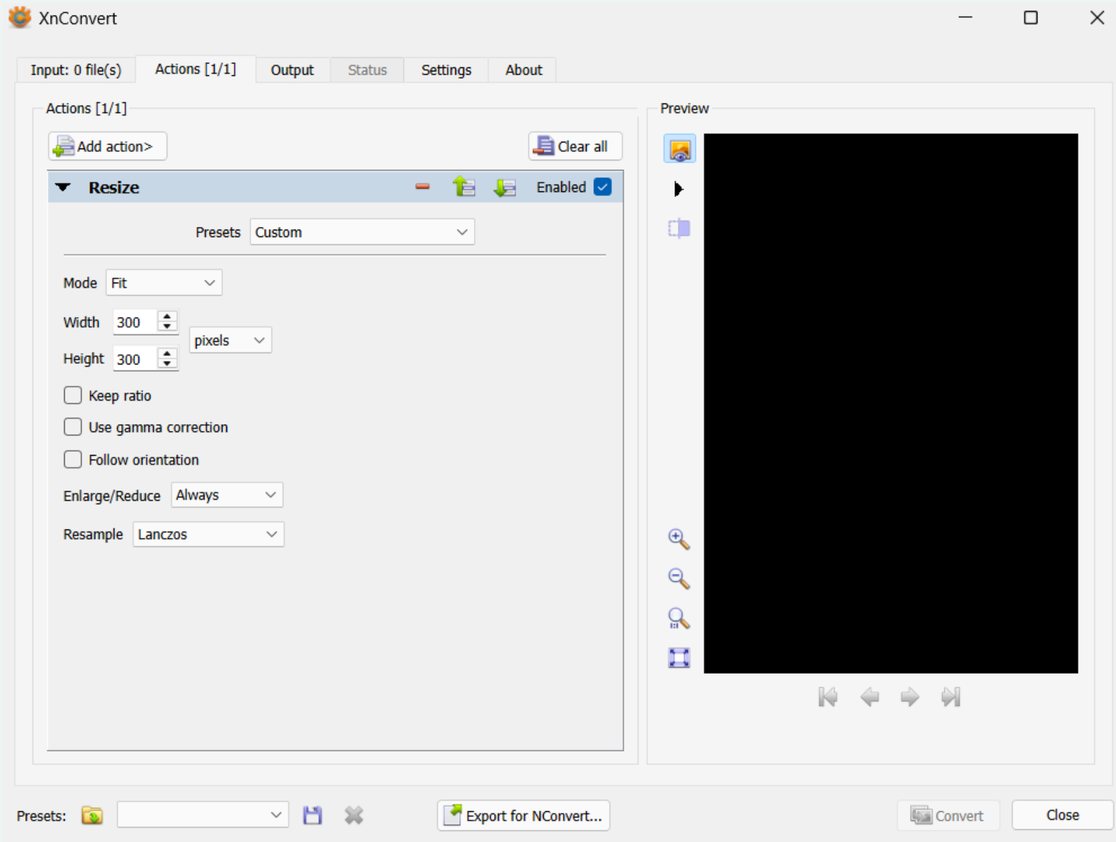


- Switching from Flowers to real images exposed mismatches:
  - earlier hyperparameters did not transfer well (resolution, batch size, augmentation strength)
  - real photos had harder negatives and higher variability

- Actions taken:
  - redesigned augmentation to stay realistic
  - tuned hyperparameters for our dataset
  - validated generalisation using **split comparisons + balanced metrics**

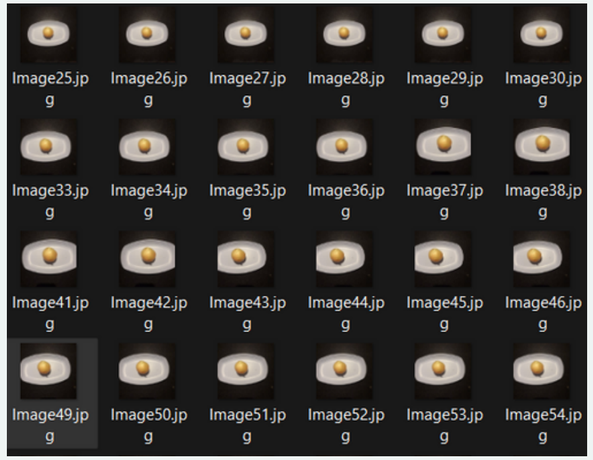
PRE-TRAINING IMAGE RESIZING

- Batch-resized images to **300 × 300** (XnConvert) to reduce storage and speed up I/O
- Keeps a consistent input source before model-specific resizing (224, 299, or 160 later)
- Trade-off: some fine detail can be lost, so we rely on higher-quality capture + augmentation to compensate

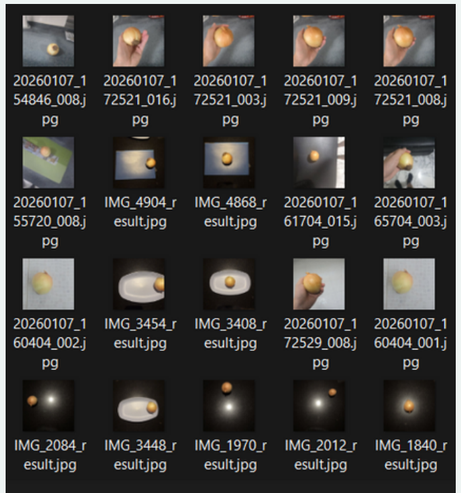


DATASET CHALLENGES AND HOW WE FIXED THEM

OLD



NEW



- Early burst shots created near-duplicate images
- Result: the model memorised backgrounds and angles (overfitting risk)

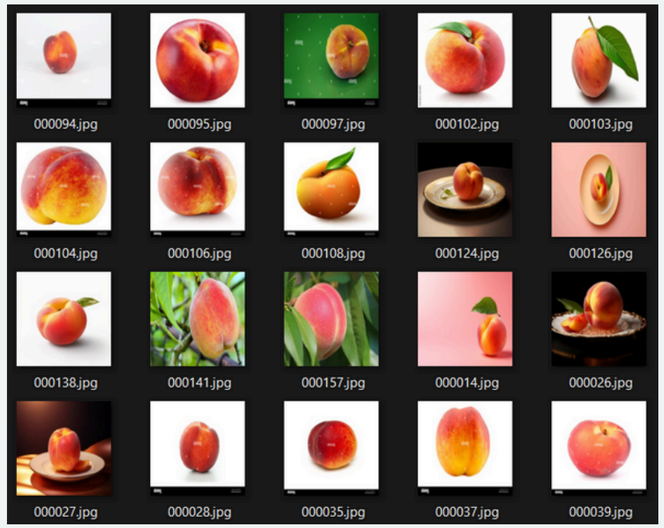
Fix

- Fewer burst shots, more diversity:
  - more angles and distances
  - different lighting conditions
  - more backgrounds and locations
  - Augmentation used to widen variation, but only after improving true capture diversity

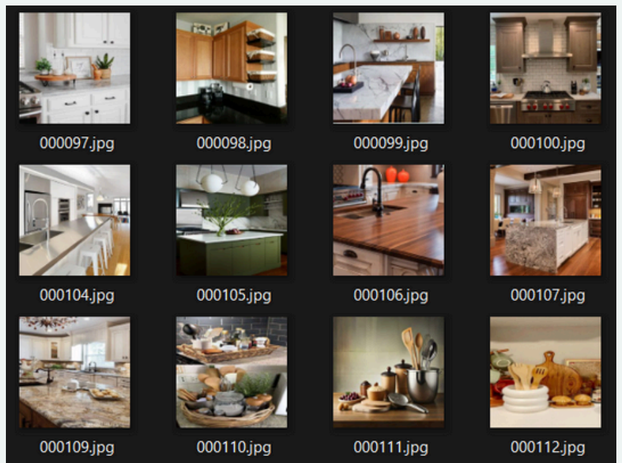
Note: Each dataset has contains the same pool of 10–12 backgrounds to ensure model does not identify objects based on the background

WEB IMAGE COLLECTION FOR DATASET GENERALISATION

PEACH



UNKNOWN



- Problem: our manual photos were partially controlled, risk of overfitting to capture conditions
- Added diverse online images via automated collection to increase variation in:
  - background, lighting, scale, viewpoint
- Added data:
  - ~50 images per food class
  - ~200 images for **unknown** to strengthen rejection
- **Outcome**: better robustness on unseen, real-world inputs

FINAL DATASET COMPOSITION (IMAGE COUNTS) + JUSTIFICATION

FINAL DATASET COMPOSITION **70/15/15 Random Stratified Split**

- Brown onion: **1099**
- Purple onion: **971**
- Peach: **1201**
- Unknown: **1706**

Why unknown is larger

- In real use, most inputs are “other”, false positives are costly (high practical harm)
- We evaluate with **balanced metrics** (Balanced Acc, Macro F1) and per-class recall

```
--- Loading BASELINE Training Data ---
Found 3462 images belonging to 4 classes.

--- Loading BASELINE Validation Data ---
Found 742 images belonging to 4 classes.

--- Loading BASELINE Test Data ---
Found 742 images belonging to 4 classes.

Class indices (train): {'onion_brown': 0, 'onion_purple': 1, 'peach': 2, 'unknown': 3}
Class indices (val):   {'onion_brown': 0, 'onion_purple': 1, 'peach': 2, 'unknown': 3}
Class indices (test):  {'onion_brown': 0, 'onion_purple': 1, 'peach': 2, 'unknown': 3}
```

CONFIGURATION AND HYPERPARAMETER OPTIMISATION FROM ORIGINAL SAMPLE CODE

```
# -------------------------
# Core hyperparameters
# -------------------------
NUM_CLASSES = 4
DEFAULT_BATCH_SIZE = 32
PRETRAINED_EPOCHS = 10

# Baseline input size (intentionally smaller baseline)
IMG_SIZE_BASE = (150, 150)
```

Objective: improve training stability, feature quality, and generalisation

- Image size: **75×75 → 150×150** (captures finer texture)
- Batch size: **214 → 32** (fits hardware, improves stability)
- Steps per epoch: **fixed → dynamic** (len(dataset)//batch) so all images are seen
- **Outcome**: more stable training and fairer evaluation

DATA PREPARATION AND AUGMENTATION

- Why: small datasets overfit, augmentation improves robustness
- Baseline: rotation 30°, flip, zoom 0.2 (no augmentation on val/test)
- Optimised for 4 classes:
  - rotation 40°, shift/shear/zoom capped at 0.1
  - add vertical flip and brightness [0.8, 1.2]
  - avoid colour shifts to preserve class cues
- Val/test: rescaling only

```
AUG_BASE = dict(
    # --- GEOMETRIC TRANSFORMATIONS (Safe for Multi-Object) ---
    rotation_range=40,          # Fruits have no standard orientation
    width_shift_range=0.1,      # REDUCED: Prevent cropping out secondary objects
    height_shift_range=0.1,     # REDUCED: Prevent cropping out secondary objects
    shear_range=0.1,            # REDUCED: Maintain shape integrity
    zoom_range=0.1,             # REDUCED: High zoom risks changing "purple onion" to just "Purple"
    horizontal_flip=True,       # Safe
    vertical_flip=True,         # Safe for top-down fruit shots

    # --- PHOTOMETRIC TRANSFORMATIONS (Color Preservation) ---
    brightness_range=[0.8, 1.2], # Handle lighting changes
    # REMOVED: channel_shift_range (Risk of confusing class colors)

    fill_mode='nearest'
)

train_datagen_base = ImageDataGenerator(
    rescale=1./255,
    **AUG_BASE
)
```

Note: augmentation supports variety, but does not replace true diversity

TRANSFER LEARNING AND MODEL COMPARISON

- Transfer learning models: InceptionV3, MobileNetV2, ResNet50

- Improvements applied:
  - full convolution base used as feature extractor
  - **GlobalAveragePooling2D** replaced Flatten to reduce overfitting
  - dynamic steps per epoch for complete data coverage

- Compared models on:
  - training time, validation accuracy, validation loss

=== BASE MODEL COMPARISON ===

	Model	Training Time (s)	Train Accuracy	Val Accuracy	Loss
0	InceptionV3	175.43	92.32	93.81	0.1598
1	MobileNetV2	188.28	97.60	98.12	0.0601
2	ResNet50	164.23	34.15	35.40	1.2942

MODEL PREPROCESSING AND TRAINING IMPROVEMENTS

- Fair comparison: same classifier head across pretrained backbones
- Correct preprocessing: use each model's **ImageNet preprocessing**
- Use native input sizes:
  - InceptionV3: **299×299**
  - MobileNetV2 / ResNet50: **224×224**
- Stabilisation:
  - Adam LR **0.0003**, max 30 epochs + early stopping
  - monitor **validation loss** (more sensitive than accuracy)

	Model	Training Time (s)	Train Accuracy (last)	Val Accuracy (best)	Val Loss (best)
0	InceptionV3	1159.40	98.67	98.25	0.0563
1	MobileNetV2	867.73	99.62	100.00	0.0050
2	ResNet50	367.32	99.88	99.60	0.0101

FINDINGS AND CONCLUSION FOR PRETRAINED MODELS

Accuracy Highlights

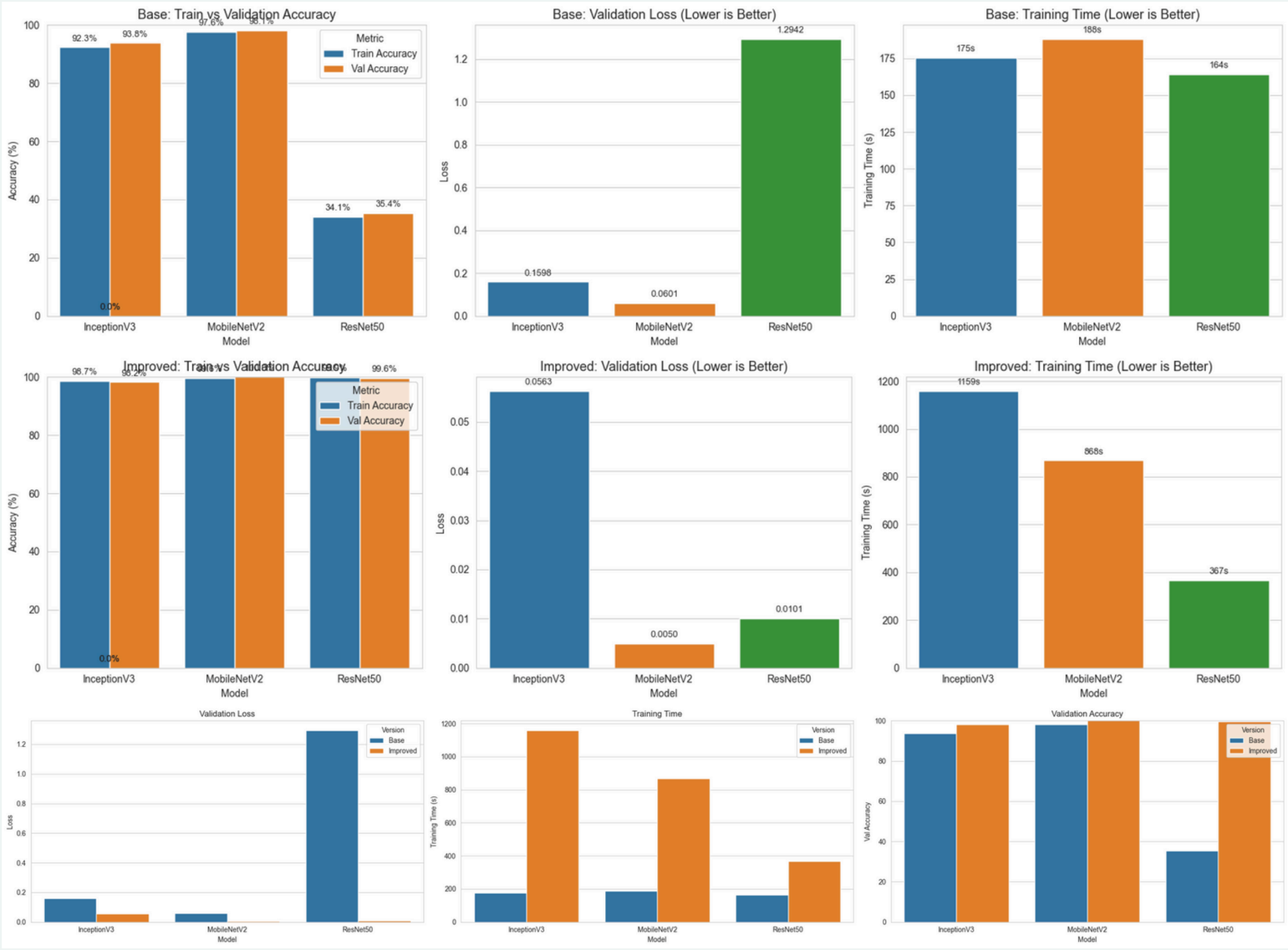
- Best overall accuracy:
- MobileNetV2 (Improved) and ResNet50 (Improved) both reach 99.6% validation accuracy
- Worst baseline performance:
- ResNet50 (Base) at only 35.4% validation accuracy
- Biggest improvement:
- ResNet50 improves by +64.2%, showing optimisation is critical

Validation Loss Highlights

- Lowest validation loss:
- MobileNetV2 (Improved) at 0.0050, indicating the most stable predictions
- Highest validation loss:
- ResNet50 (Base) at 1.2942, showing severe underfitting

Training Time Highlights

- Slowest model after optimisation:
- InceptionV3 (Improved) at 1159 s
- Fastest improved model:
- ResNet50 (Improved) at 367 s



FINDINGS AND CONCLUSION FOR PRETRAINED MODELS

Key findings

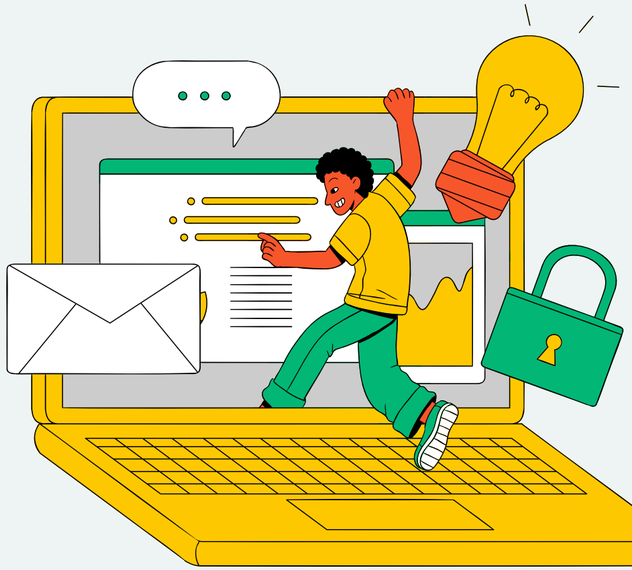
- Correct preprocessing is critical for **transfer learning reliability**
- Base ResNet50 failed under generic preprocessing
- Transfer learning reaches **plateau faster** than training from scratch

Performance summary

- **Best overall**: ResNet50 (Improved), highest accuracy and macro F1 with lowest cost
- **Best efficiency**: MobileNetV2 (Improved), strong accuracy with low cost
- **Most expensive**: InceptionV3 (Improved), highest training time

```
=== TEST SET SUMMARY (BASE) ===
                  Model File  Accuracy (%)  Macro F1
0  model_inception_v3_base.h5         93.00    0.9248
1   model_MobileNetV2_base.h5         99.06    0.9903
2      model_ResNet50_base.h5         34.72    0.2104

=== TEST SET SUMMARY (IMPROVED) ===
                      Model File  Accuracy (%)  Macro F1
0  model_InceptionV3_improved.h5         99.06    0.9901
1  model_MobileNetV2_improved.h5         99.73    0.9974
2     model_ResNet50_improved.h5         99.87    0.9987
```



CUSTOM CNN

TRAINING CONFIGURATION JUSTIFICATION

Epoch selection with EarlyStopping

- Maximum of 100 epochs allows **full convergence**
- EarlyStopping **halts training** once validation performance stops improving
- Prevents **premature stopping** without risking overfitting
- Models typically converge within 30–50 epochs

Optimiser choice (Adam)

- Fast, **stable** convergence from random initialisation
- Adaptive learning rates **reduce manual tuning**
- **More reliable** than SGD, RMSprop, or Adagrad for this setting

Why monitor validation loss

- More **informative** than accuracy for generalisation
- Reflects **confidence and class separation**, not just correctness
- Continued loss reduction after accuracy plateaus indicates **better calibration**
- Especially suitable when using focal loss

NOTE: VALIDATION ACCURACY OVER EPOCHS, ALONG WITH OTHER EVALUATION AND TEST GRAPHS, ARE GENERATED IN THE JUPYTER NOTEBOOK BUT OMITTED FROM THE SLIDES DUE TO TIME CONSTRAINTS.

INITIAL CUSTOM CNN (BASELINE)

Objective

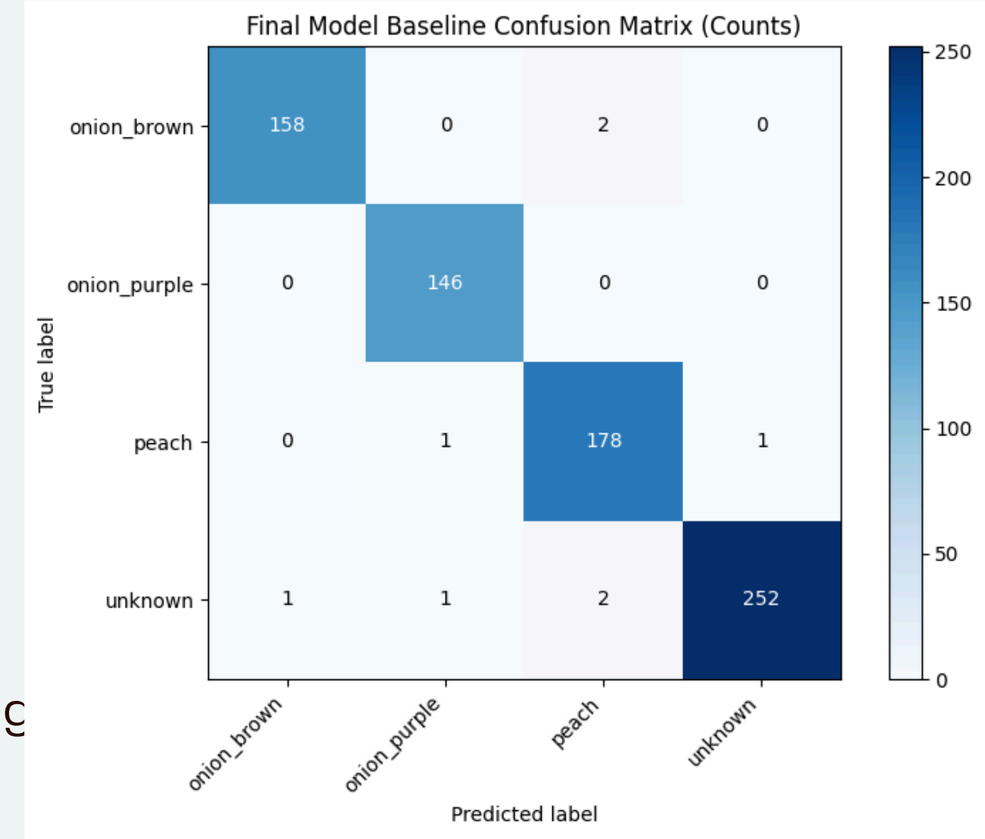
- Scratch trained baseline to demonstrate CNN architecture understanding

Architecture

- Three convolution blocks, **32, 64, 128,** with MaxPooling
- Flatten + Dense (**512**) to increase parameter count
- Dropout (**0.5**) for regularisation

Purpose

- Controlled reference model
- Highlights **limitations** of scratch training vs transfer learning



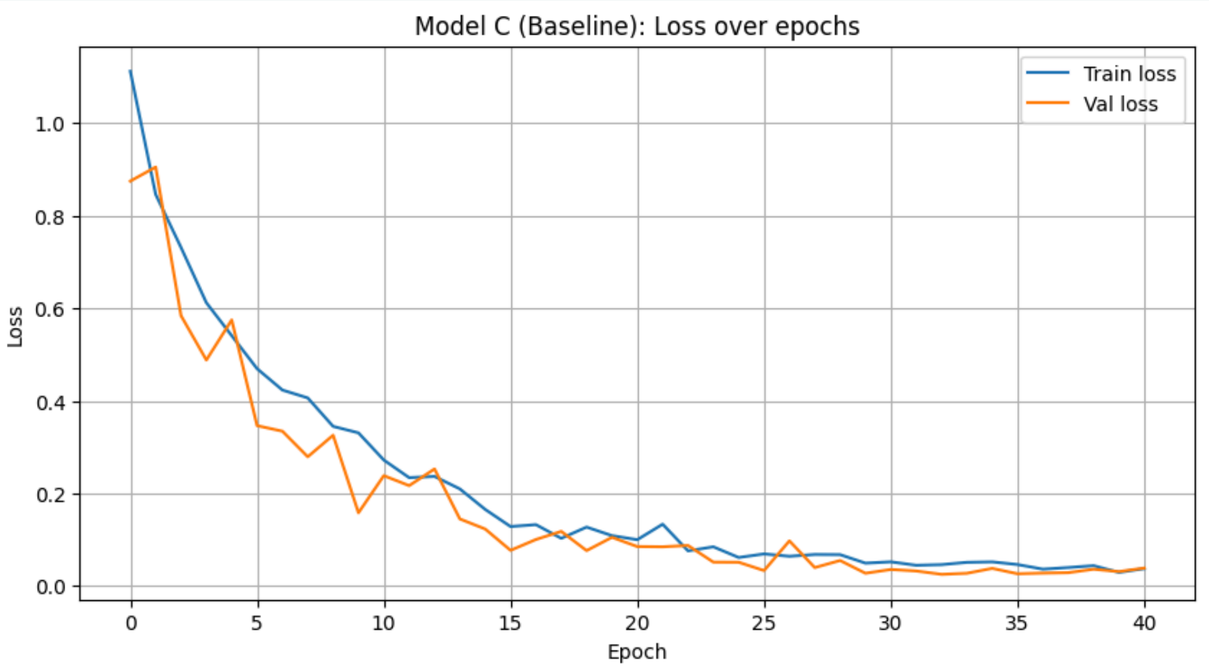
INITIAL CUSTOM CNN (BASELINE)

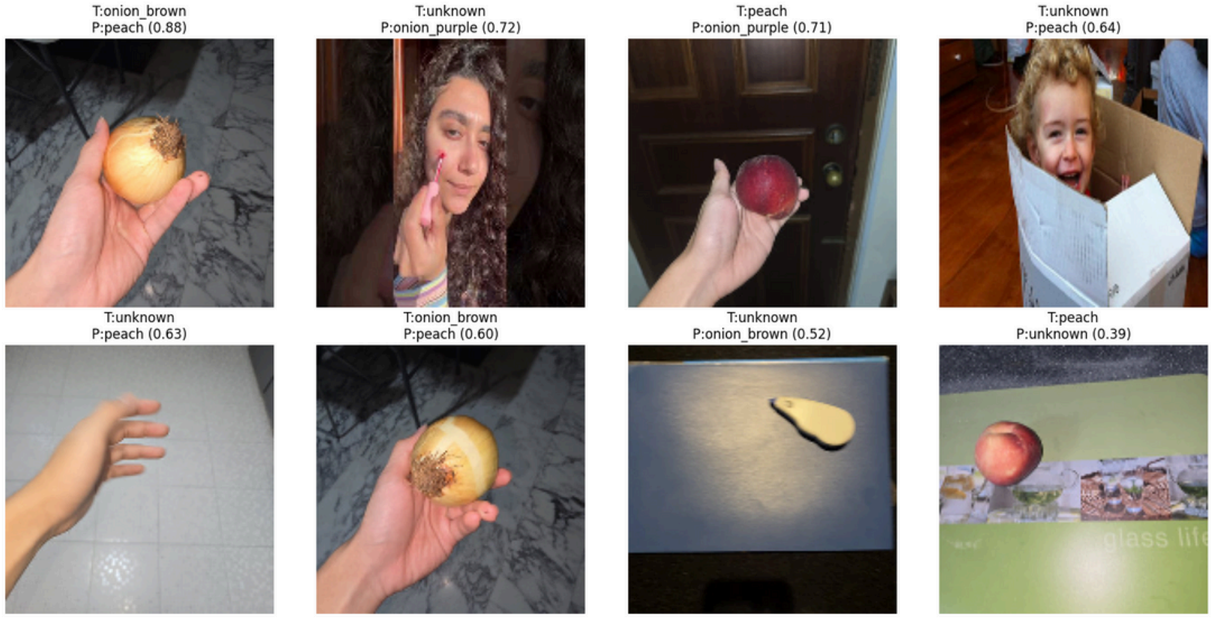
Training behaviour

- Training and validation loss drop rapidly, then taper **smoothly**
- Loss curves remain close, indicating **good generalisation**
- Low, stable loss in later epochs shows convergence

Misclassification analysis

- Bright or beige objects often predicted as peach
- Indicates **reliance on colour cues** over shape or texture
- Flatten layer increases parameters, amplifying **overfitting** on limited datasets



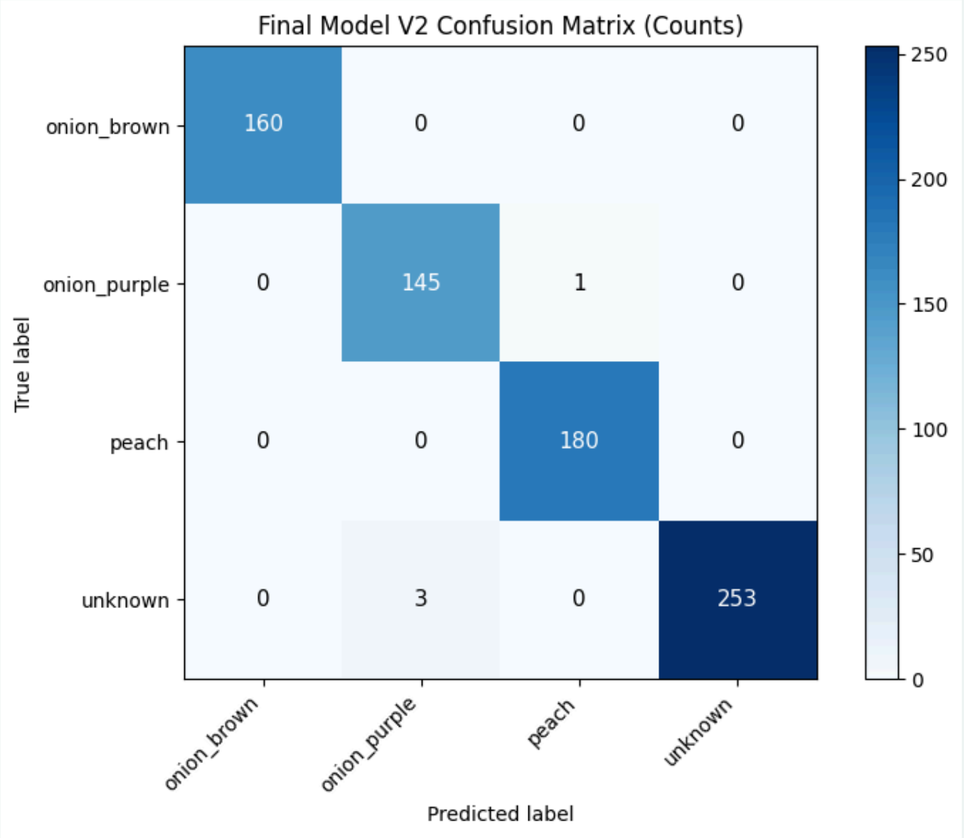


CUSTOM CNN V2: SHAPE-FIRST AND STABILITY

Goal
- Reduce colour dependence
- Promote **shape and texture** learning

Architecture
- VGG style double Conv2D blocks
- Batch Normalisation throughout
- Global Average Pooling instead of Flatten

Outcome
- Stronger structural feature extraction
- Improved **stability and generalisation**



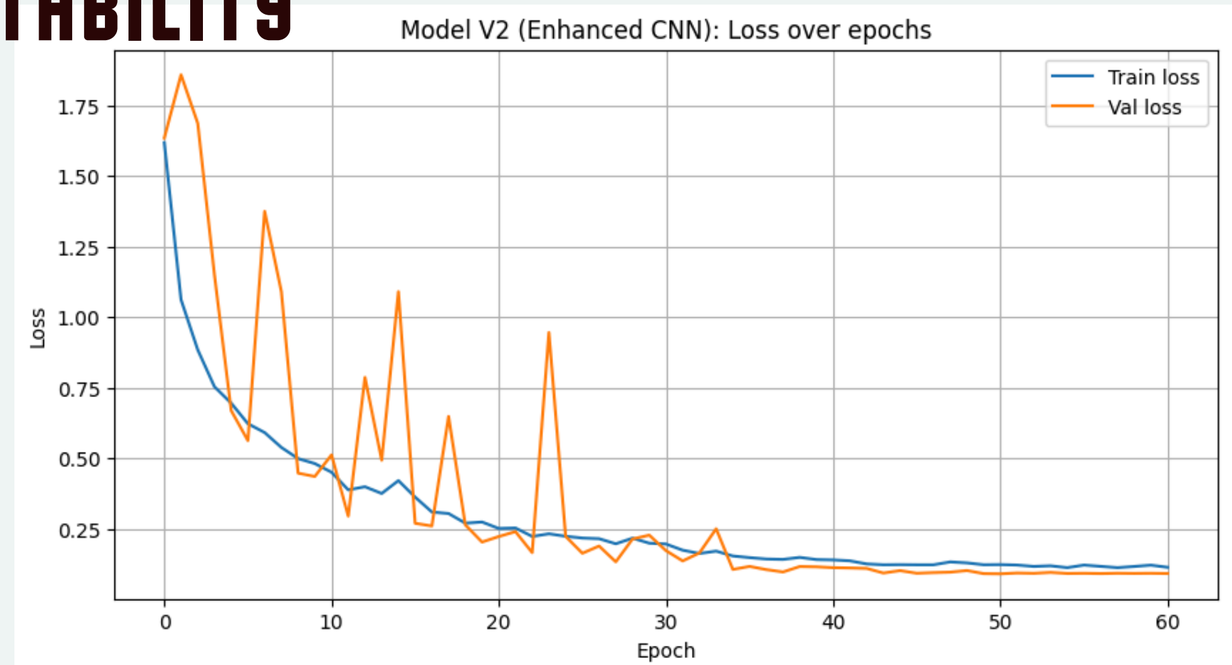
CUSTOM CNN V2: SHAPE-FIRST AND STABILITY

Training behaviour

- Training loss decreases steadily
- Validation loss is initially noisy, then stabilises and closely follows training
- Indicates **improved generalisation** rather than overfitting
- Slower early convergence reflects **deeper feature learning**
- Batch normalisation and global average pooling improve stability and **reduce overfitting**

Misclassification analysis

- No longer strongly confuses beige objects with peach
- A small number of unknown images containing purple tones are **misclassified as purple onion**, suggesting residual colour influence rather than shape confusion



CUSTOM CNN V3: ATTENTION AND ACTIVATION

Goal

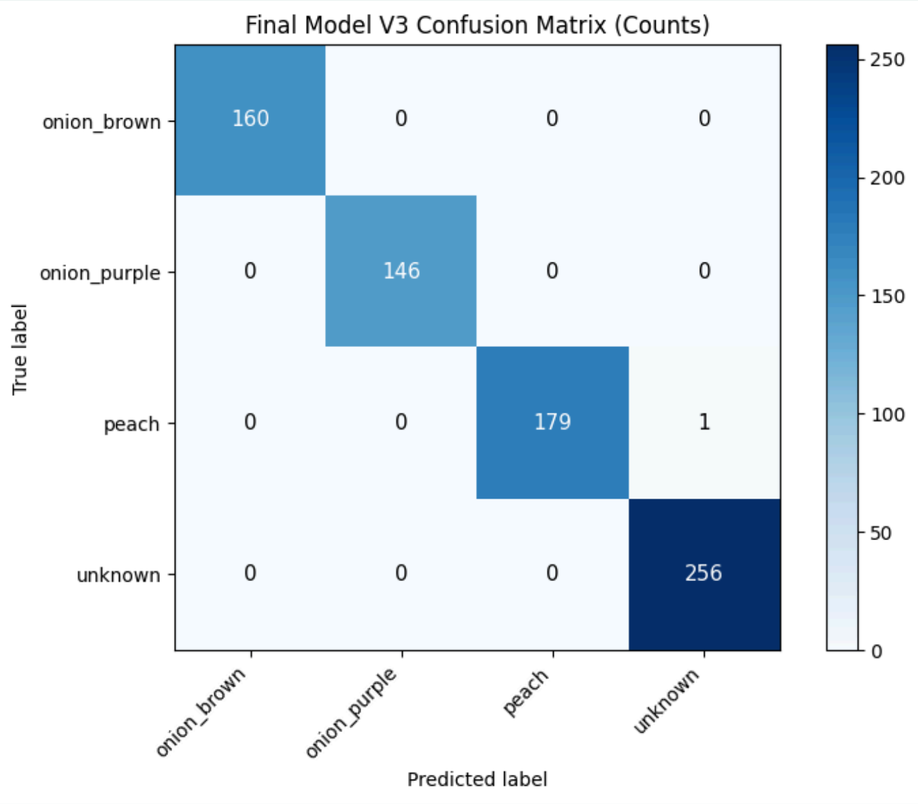
- Reduce colour driven false positives
- Prioritise **shape and texture** over colour cues

Key Improvements

- Squeeze and Excitation blocks to reweight feature channels
- Swish activation to preserve subtle features and improve gradients
- Balanced class weighting to reduce class bias

Intended Result

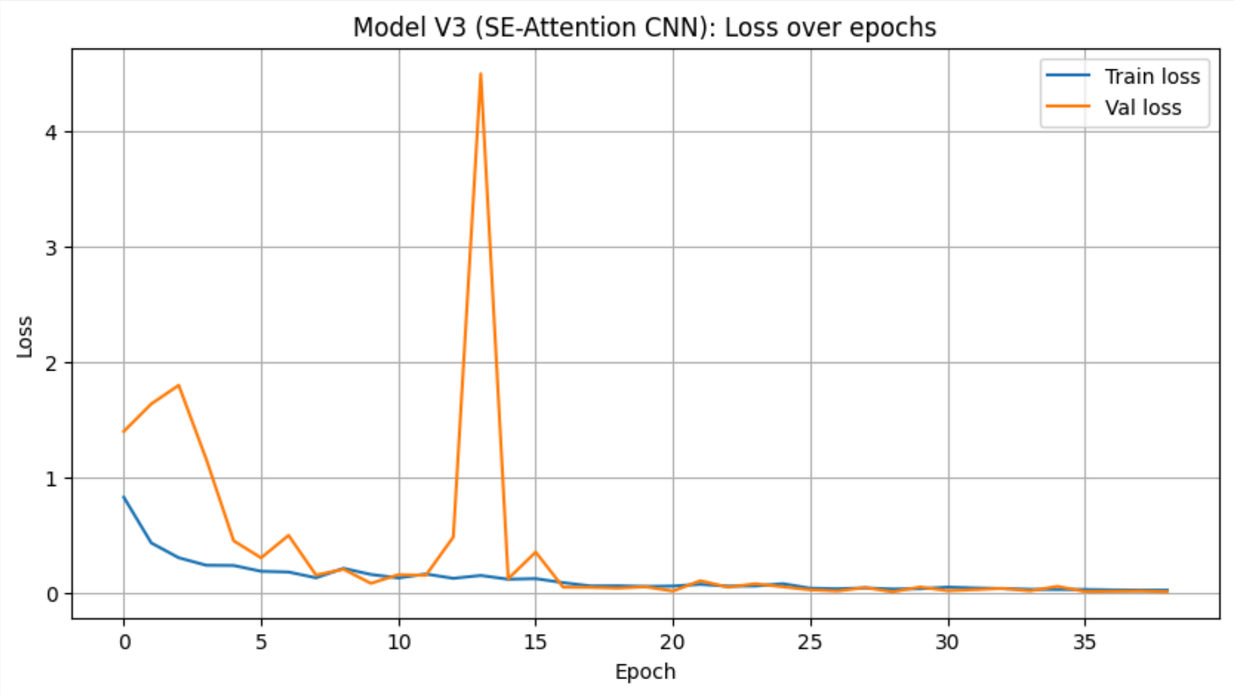
- Better separation of colour and shape features
- Fewer colour based misclassifications
- Improved **robustness** on ambiguous inputs



CUSTOM CNN V3: ATTENTION AND ACTIVATION

Training behaviour

- Training loss decreases smoothly and remains low
- Validation loss shows an early spike, then quickly aligns with training
- Early spike reflects **SE attention adjustment** and class rebalancing, not overfitting
- SE blocks downweight colour-dominant cues when shape information is weak
- Swish activation improves gradient flow and stabilises recovery
- Balanced class weights **reduce majority-class bias** in the decision boundary
- Overall, training is stable with reduced overfitting and strong generalisation



Misclassification analysis

- **No significant** systematic errors
- Only one peach misclassified as unknown, indicating robust discrimination

CUSTOM CNN V4: HARD MINING AND ROBUSTNESS

Goal

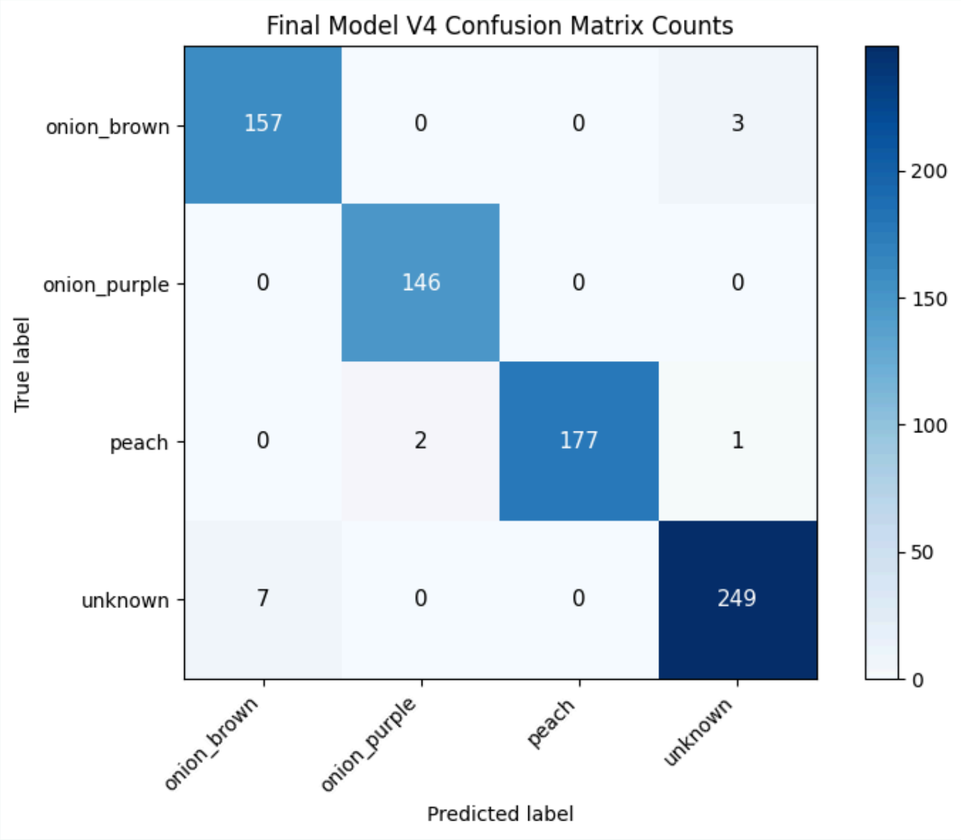
- Address **hard negatives** such as peach misclassified as unknown
- Force learning on difficult and ambiguous examples

Key Improvements

- **Focal Loss** to focus training on hard, misclassified samples
- SpatialDropout2D to reduce reliance on dominant colour features
- SE attention and Swish activation retained for feature selection and stability

Intended Result

- Improved handling of confusing inputs
- Fewer **hard negative errors**
- Most robust custom CNN in this study



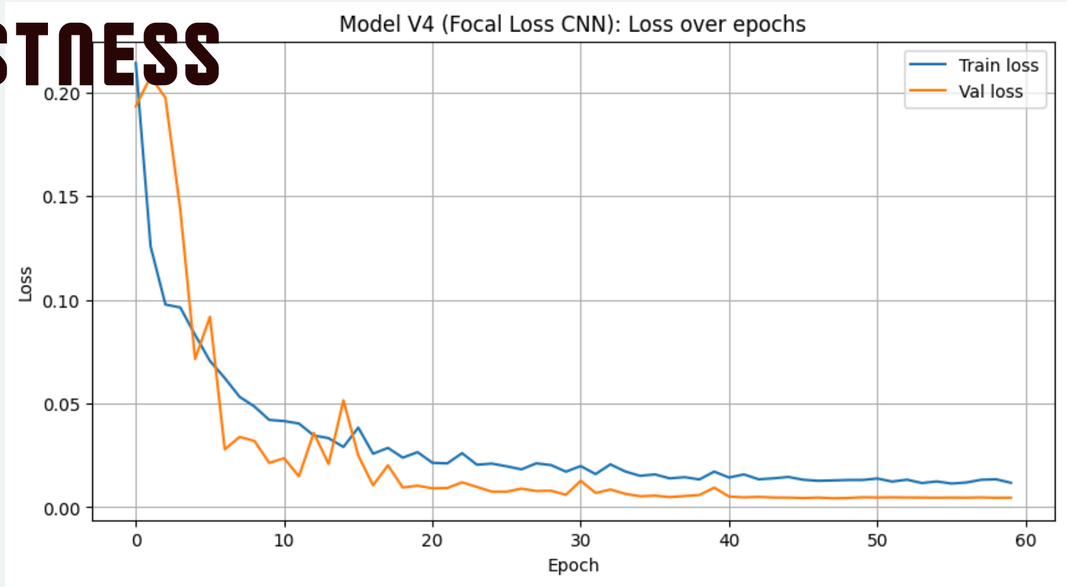
CUSTOM CNN V4: HARD MINING AND ROBUSTNESS

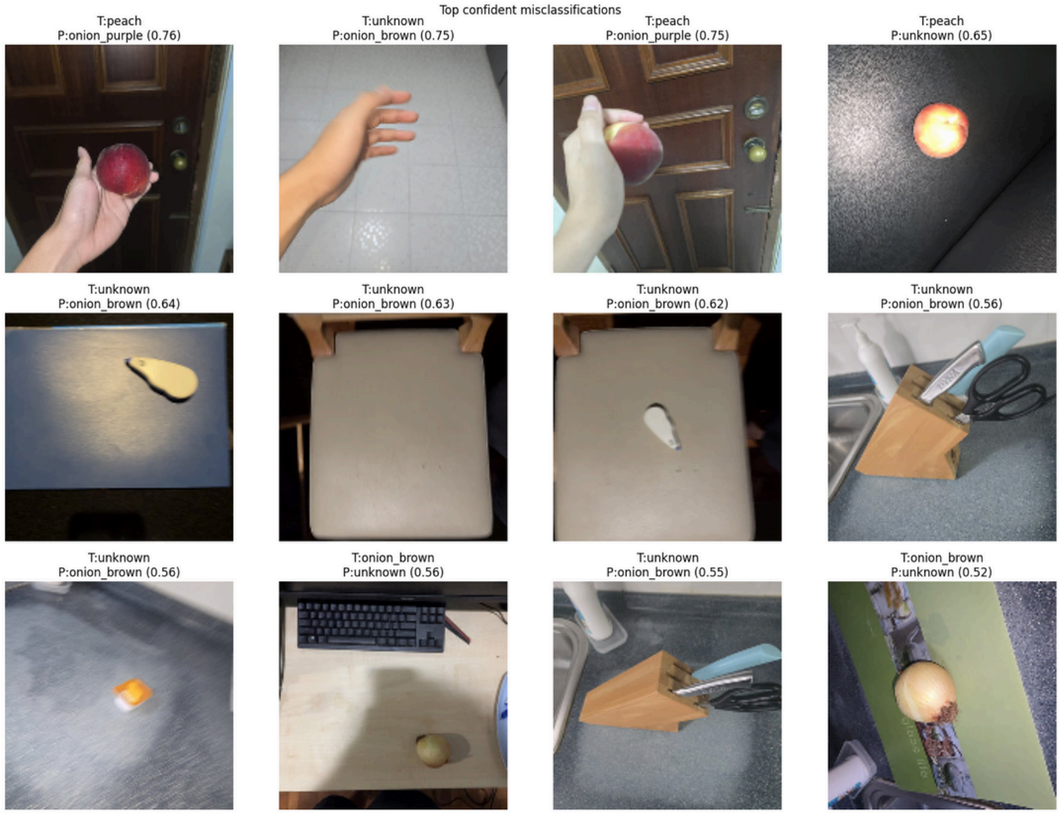
Training behaviour

- Training and validation loss decrease steadily and remain close
- **Higher loss plateau than v3** indicates **weaker** discrimination, not better robustness
- Focal Loss over-emphasises hard examples in a small dataset
- SpatialDropout2D further suppresses feature learning, causing over-regularisation
- Despite SE blocks and Swish activation, v4 converges to a sub-optimal solution

Misclassification analysis

- Brown backgrounds misclassified as brown onion
- Darker peaches misclassified as purple onion
- Some brown onion samples predicted as unknown

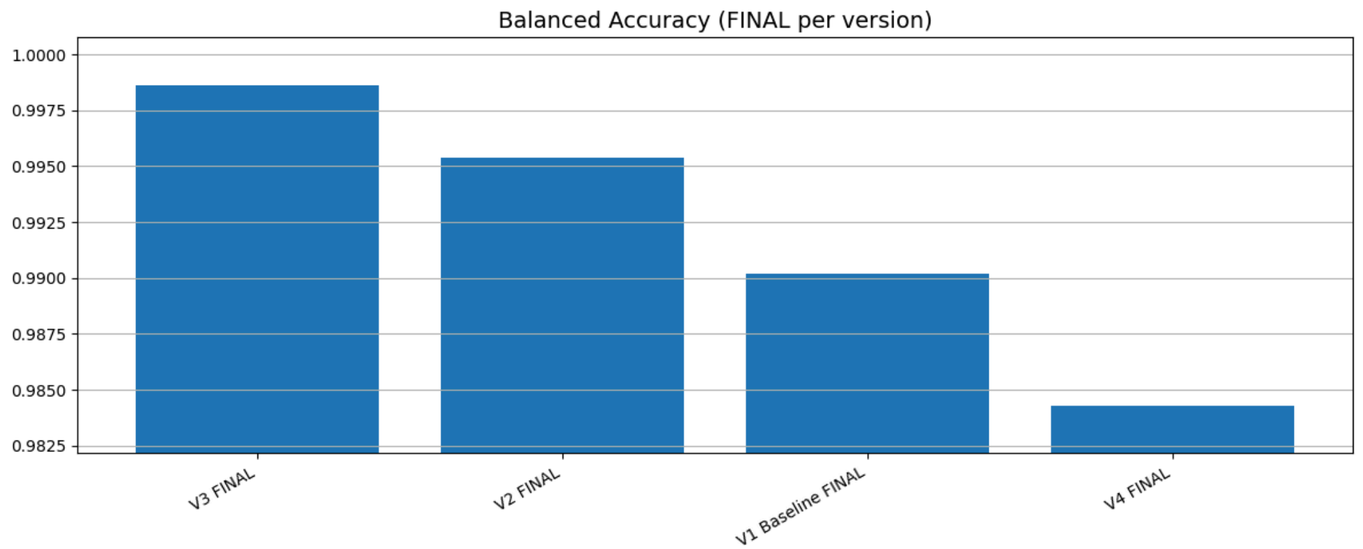


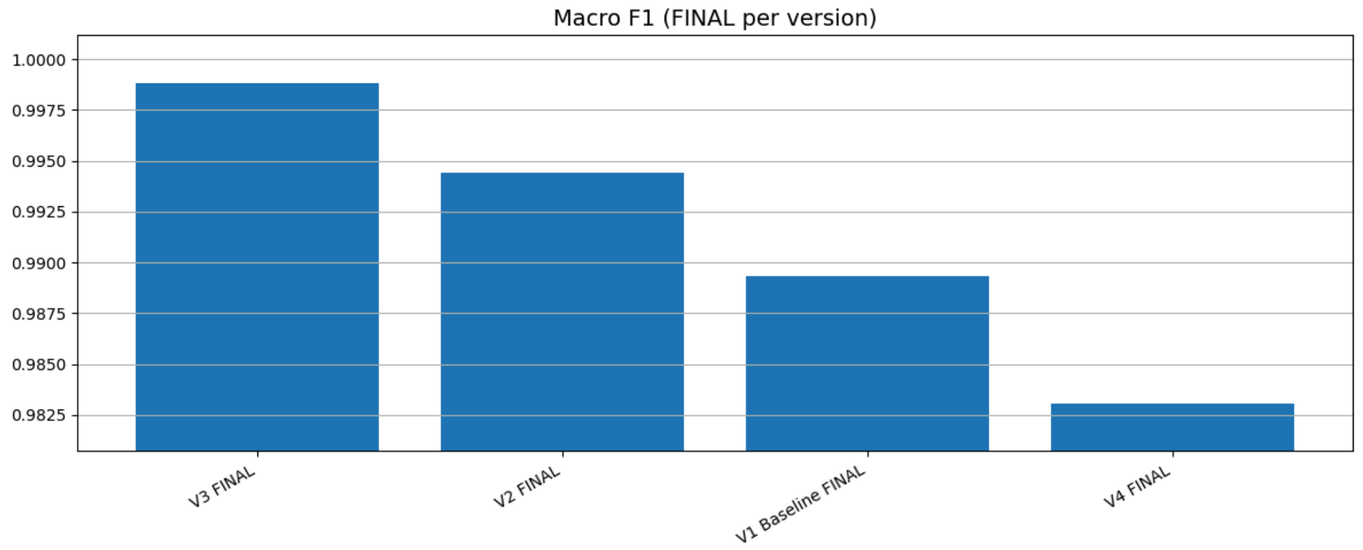


MODEL COMPARISON VISUALISATION AND SELECTION

How we selected the best architecture for hyperparameter tuning:

- All models evaluated on the **same validation** set for fairness (Test set unused to ensure no bias)
- Used **balanced accuracy** and macro F1 to handle class imbalance (as unknown is larger)
- Cohen's kappa confirms non-random learning
- Considered model size and inference speed for deployment trade-offs





MODEL COMPARISON VISUALISATION AND SELECTION

- **V3 best overall:** most consistent, strongest unknown handling
- **V2 near perfect:** slight drop on purple onion recall
- **V1 solid:** weaker on peach and purple onion
- **V4 weakest:** poor on unknown and brown onion

Key takeaway

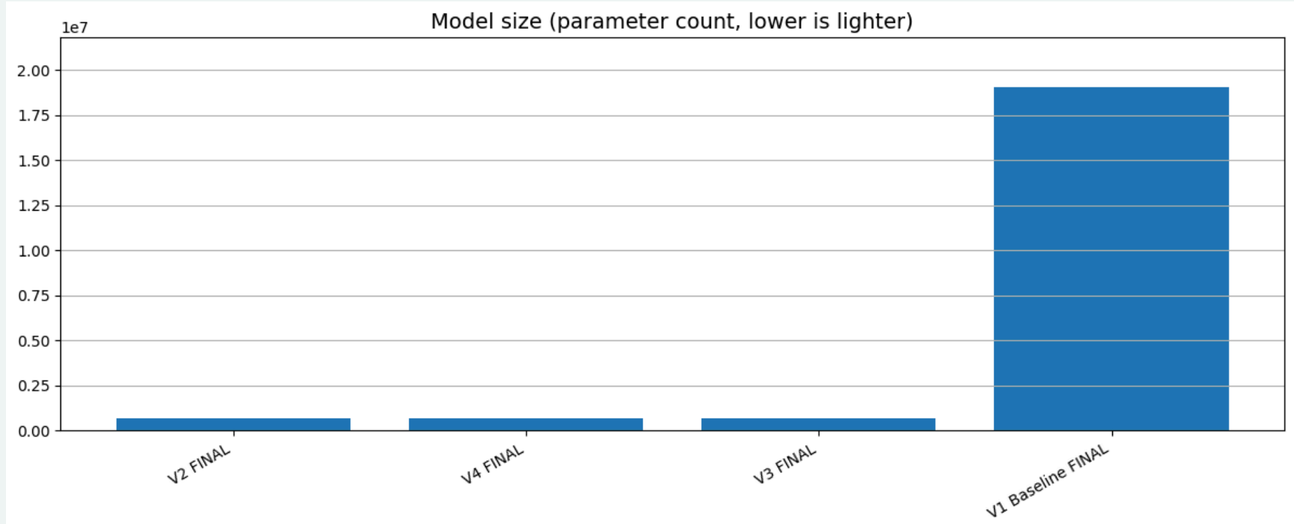
- V3 clearly outperforms all models

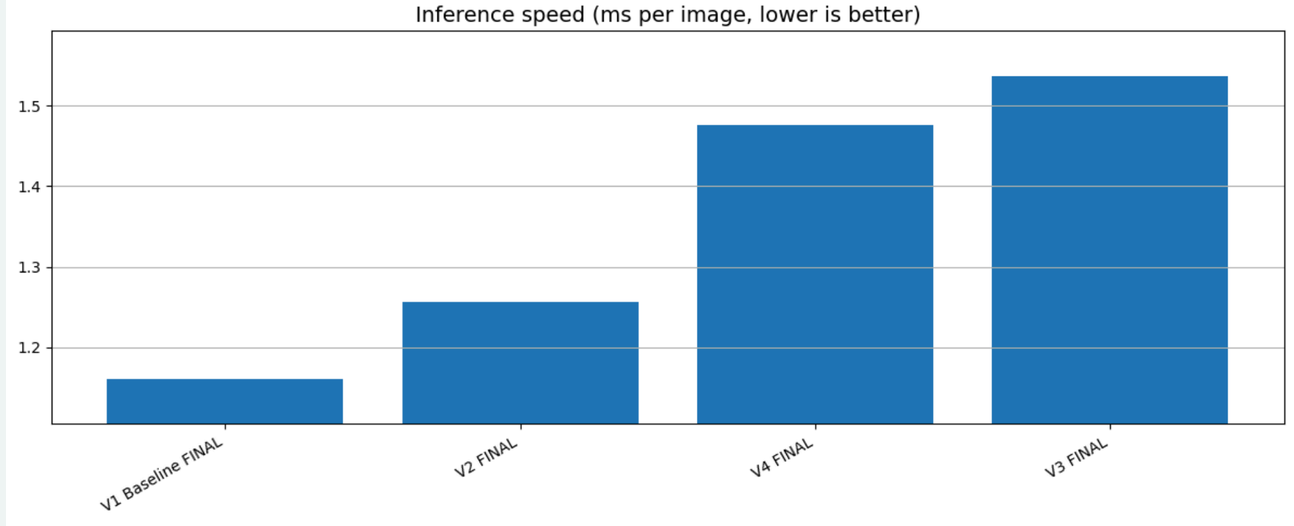
	Model	Class	Precision	Recall	F1	Support
0	V1 Baseline FINAL	onion_brown	0.993711	0.987500	0.990596	160
1	V1 Baseline FINAL	onion_purple	0.986486	1.000000	0.993197	146
2	V1 Baseline FINAL	peach	0.978022	0.988889	0.983425	180
3	V1 Baseline FINAL	unknown	0.996047	0.984375	0.990177	256
4	V2 FINAL	onion_brown	1.000000	1.000000	1.000000	160
5	V2 FINAL	onion_purple	0.979730	0.993151	0.986395	146
6	V2 FINAL	peach	0.994475	1.000000	0.997230	180
7	V2 FINAL	unknown	1.000000	0.988281	0.994106	256
8	V3 FINAL	onion_brown	1.000000	1.000000	1.000000	160
9	V3 FINAL	onion_purple	1.000000	1.000000	1.000000	146
10	V3 FINAL	peach	1.000000	0.994444	0.997214	180
11	V3 FINAL	unknown	0.996109	1.000000	0.998051	256
12	V4 FINAL	onion_brown	0.957317	0.981250	0.969136	160
13	V4 FINAL	onion_purple	0.986486	1.000000	0.993197	146
14	V4 FINAL	peach	1.000000	0.983333	0.991597	180
15	V4 FINAL	unknown	0.984190	0.972656	0.978389	256

MODEL COMPARISON VISUALISATION AND SELECTION

Selection result:

- v3 final ranked highest across balanced accuracy, macro F1, and stability
- v2 strong but slightly weaker class balance
- v1 baseline over-parameterised, higher overfitting risk
- v4 underperformed (over-regularisation)





Why inference speed was not prioritised:

- All models were within roughly **1.2 to 1.5 ms** per image
- Reliability and class discrimination mattered more than sub-millisecond differences

FINAL VERDICT: FOUR-CLASS CNN COMPARISON (DONT PRESENT)

1. v3 final: Balanced Acc 0.9986, Macro F1 0.9988, Kappa 0.9981, ~0.66M params
2. v2 final: Balanced Acc 0.9954, Macro F1 0.9944, Kappa 0.9927, ~0.65M params
3. v1 baseline: Balanced Acc 0.9902, Macro F1 0.9893, Kappa 9854, ~19.0M params
4. v4 final: Balanced Acc 0.9843, Macro F1 0.9831, Kappa 0.9825, ~0.66M params

Decision: v3 final, best balance of class consistency, stability, and efficiency

	Model	Evaluated On	Params	Accuracy	Balanced Acc	F1 Macro	F1 Weighted	Kappa	Pred Time (s)	ms/img	Train Time (s)	Best Epoch (history)	Hist Best Val Acc	Hist Final Val Acc
2	V3 FINAL	val_gen_base	662756	0.998652	0.998611	0.998816	0.998652	0.998171	1.139938	1.536305	687.952584	29	0.998652	0.995957
1	V2 FINAL	val_gen_base	652836	0.994609	0.995358	0.994433	0.994617	0.992690	0.932060	1.256146	1056.293972	38	0.997305	0.991914
0	V1 Baseline FINAL	val_gen_base	19035716	0.989218	0.990191	0.989349	0.989224	0.985379	0.861605	1.161193	674.869678	32	0.990566	0.986523
3	V4 FINAL	val_gen_base	662628	0.982480	0.984310	0.983080	0.982511	0.976247	1.095498	1.476412	1014.580374	37	0.983827	0.982480

HYPERPARAMETER TUNING GUIDE THAT WE FOLLOWED

Goal

- **Optimise** the Enhanced Custom CNN, searching around the current best settings

Key point

- Search space is constrained around the Enhanced CNN, so improvements come from **better parameter selection, not architecture changes**

Hyperparameter	Variable Name	Current Enhanced CNN	Tuning Range	Effect
Image Size	`img_size`	`150 x 150`	`128, 160, 192, 224`	Controls input resolution. Higher resolution captures finer details but increases computational cost. Lower resolution trains faster but may lose small features.
Batch Size	`batch_size`	`32`	`16, 32, 64`	Number of samples per update. Smaller batches introduce noise that helps generalisation; larger batches provide stable gradients and faster training speed.
Filters Block 1	`filters_1`	`32`	`32, 64`	Controls the number of low-level feature maps (edges, textures). Higher values increase representational capacity but raise computation cost.
Filters Block 2	`filters_2`	`64`	`64, 96, 128`	Affects intermediate feature extraction. More filters improve feature diversity but increase model size.
Filters Block 3	`filters_3`	`128`	`128, 192, 256`	Captures higher-level visual patterns. Larger values allow richer representations but risk overfitting.
Filters Block 4	`filters_4`	`256`	`256, 384, 512`	Deepest convolutional block, responsible for abstract semantic features. Increasing this significantly raises model capacity.
Dense Units	`dense_units`	`256`	`128, 256, 384, 512`	Controls the capacity of the fully connected layer that combines extracted features before classification.
Dropout Rate	`dropout`	`0.5`	`0.2 – 0.6`	Regularisation strength. Higher values reduce overfitting but may slow convergence or cause underfitting.
L2 Regularisation	`l2_rate`	`0.001`	`0.01, 0.001, 0.0001`	Penalises large weights to prevent memorisation. Stronger values improve generalisation but can limit learning capacity.
Learning Rate	`learning_rate`	`0.001`	`0.01, 0.001, 0.0001`	Determines the step size during optimisation. Too high causes instability, too low slows convergence.

WOW FACTOR: HYPERPARAMETER TUNING V1 (HYPERBAND)

Objective

- Optimise CNN v3 with **Keras Tuner (Hyperband)**
- Improve **stability** and generalisation without changing architecture

Why Hyperband

- **Grid search** is computationally infeasible
- Compute aware, prunes weak trials early
- Achieves similar results in far fewer trials

Tuning Setup

- **EarlyStopping** to terminate weak runs
- No learning rate scheduling for fair comparison

Outcome

- Efficient and interpretable optimisation
- Resources focused on high performing configurations

```
    img_dim = hp.Int("img_size", min_value=128, max_value=224, step=32)

    # Keep filters fixed to match your v3 architecture
    f1, f2, f3, f4 = 32, 64, 128, 256

    # Optional: tune SE ratio (kept conservative)
    se_ratio = hp.Choice("se_ratio", values=[8, 16])

    # Tune head size around 256 (still aligned with v3 intent)
    head_units = hp.Choice("head_units", values=[128, 256, 384])

    # Tune dropouts around your v3 values (small safe band)
    d2 = hp.Choice("dropout_b2", values=[0.1, 0.2, 0.3])
    d3 = hp.Choice("dropout_b3", values=[0.2, 0.3, 0.4])
    d4 = hp.Choice("dropout_b4", values=[0.2, 0.3, 0.4])
    dh = hp.Choice("dropout_head", values=[0.3, 0.4, 0.5])
```

```
tuner = DataAugmentationTunerV3(
    build_tunable_attention_cnn_v3,
    objective=kt.Objective("val_loss", direction="min"),
    max_epochs=15,
    factor=3,
    directory="tuning_results",
    project_name="v3_attention_hyperband",
    overwrite=True,
)
```

WOW FACTOR: HYPERPARAMETER OPTIMISATION V2 (MACRO-MICRO PYRAMID STRATEGY)

Problem

- Single stage tuning mixes architecture and regularisation, which could cause bias

Approach

- Two stage Macro–Micro strategy with enforced **filter pyramid**

Stage 1: **Macro (Hyperband)**

- Tune image size, batch size, base filters, learning rate
- Enforced 1×, 2×, 4×, 8× filter pyramid
- Aggressive pruning, large search space reduction

Stage 2: **Micro (Bayesian)**

- Architecture fixed
- Tune dense units, dropout, L2 regularisation

Result

- Faster and more **stable optimisation**
- Reduced bias and deployment ready tuning

```
    img_dim = hp.Choice("img_size", values=[128, 176, 224])
    base_filter = hp.Choice("base_filter", values=[32, 48, 64])
    lr = hp.Choice("learning_rate", values=[1e-3, 1e-4])
```

```
tuner1 = Stage1Tuner(
    build_macro_model_v3,
    objective="val_accuracy",
    max_epochs=12,
    factor=3,
    directory="tuning_results",
    project_name="stage1_macro_reduced_v3_arch",
    overwrite=True
)
```

```
  hp_l2 = hp.Choice("l2_rate", values=[1e-2, 1e-3, 1e-4])
  dense_units = hp.Int("dense_units", 128, 384, step=128)
  dropout_rate = hp.Float("dropout", 0.2, 0.6, step=0.1)
```

```
tuner2 = kt.BayesianOptimization(
    build_micro_model_v3,
    objective="val_accuracy",
    max_trials=12,
    directory="tuning_results",
    project_name="stage2_micro_fast_v3_arch",
    overwrite=True
)
```

HYPERPARAMETER RESULTS FROM V1 AND V2

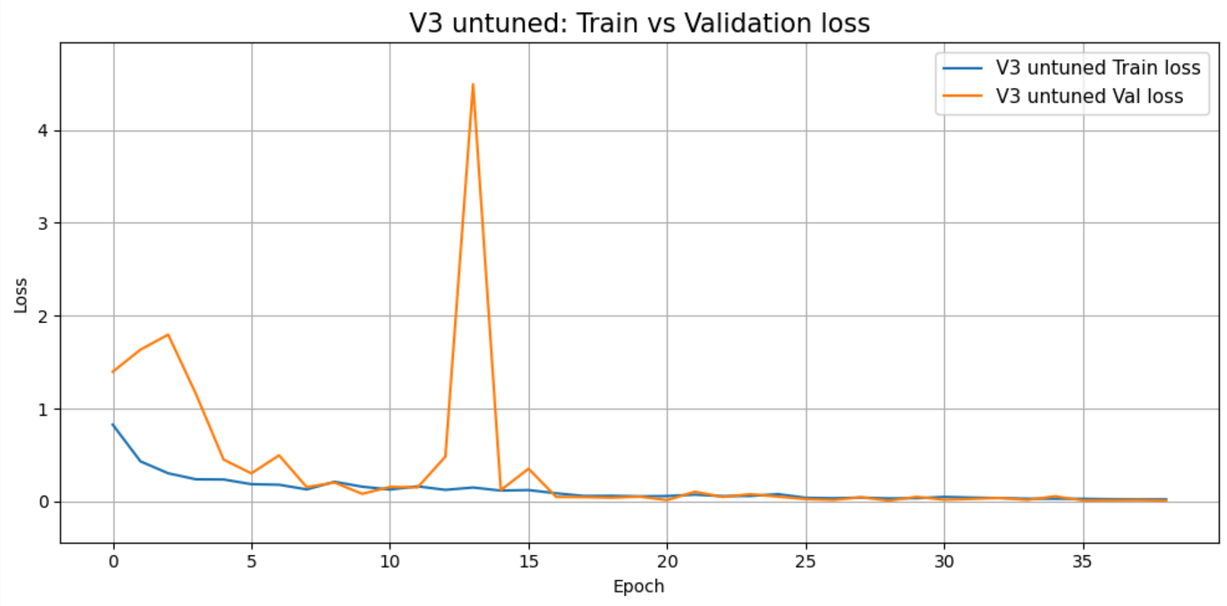
V1

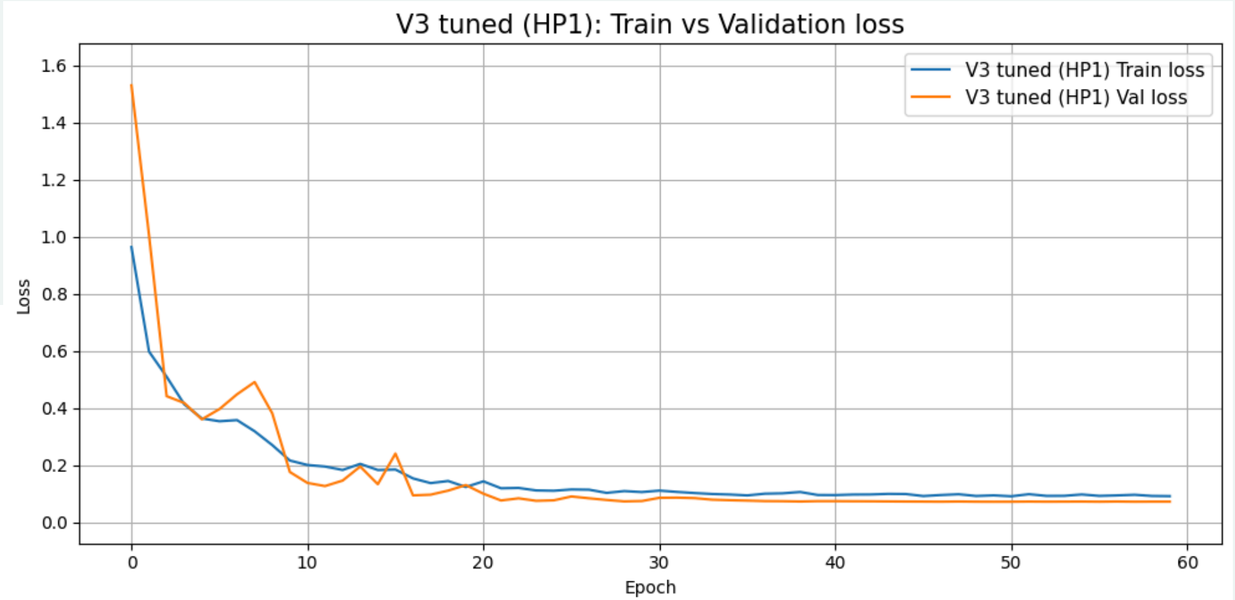
Rank	Input Dims	Network Capacity	Regularisation	Learning Rate	Val Acc	Val F1	Specificity
1	128x128 (Batch 16)	**Filters:** [32, 64, 128, 512] **Dense:** 128	**Dropout:** 0.2 **L2:** 0.001	0.0001	**0.9586**	0.9517	0.9898
2	160x160 (Batch 16)	**Filters:** [32, 128, 192, 256] **Dense:** 512	**Dropout:** 0.3 **L2:** 0.001	0.0001	0.8984	0.8973	0.9719
3	128x128 (Batch 16)	**Filters:** [64, 128, 256, 256] **Dense:** 128	**Dropout:** 0.2 **L2:** 0.001	0.0005	0.8850	0.8847	0.9648
4	128x128 (Batch 16)	**Filters:** [32, 64, 128, 512] **Dense:** 128	**Dropout:** 0.2 **L2:** 0.001	0.0001	0.5802	0.5563	0.8904
5	128x128 (Batch 32)	**Filters:** [64, 64, 256, 256] **Dense:** 512	**Dropout:** 0.5 **L2:** 0.0001	0.0005	0.4866	0.0000	1.0000

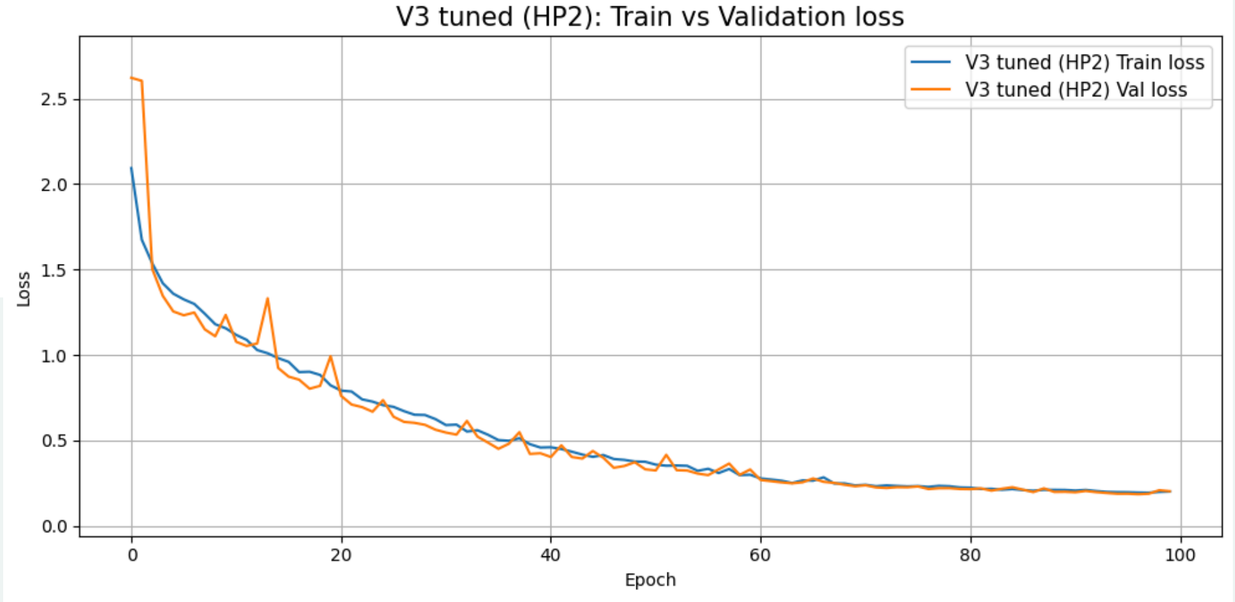
V2

Rank	Trial ID	Image Size	Batch Size	Filters (B1–B4)	Learning Rate	Dense Units	Dropout	L2 Rate	Val Accuracy	Val F1 Score	Specificity
1	06	176×176	16	[48, 96, 192, 384]	0.0001	384	0.3	0.001	0.9596	0.9581	0.9874
2	02	176×176	16	[48, 96, 192, 384]	0.0001	128	0.2	0.001	0.9501	0.9477	0.9856
3	10	176×176	16	[48, 96, 192, 384]	0.0001	384	0.3	0.001	0.9488	0.9464	0.9843
4	08	176×176	16	[48, 96, 192, 384]	0.0001	128	0.3	0.001	0.9474	0.9414	0.9843
5	04	176×176	16	[48, 96, 192, 384]	0.0001	256	0.4	0.001	0.9394	0.9350	0.9802

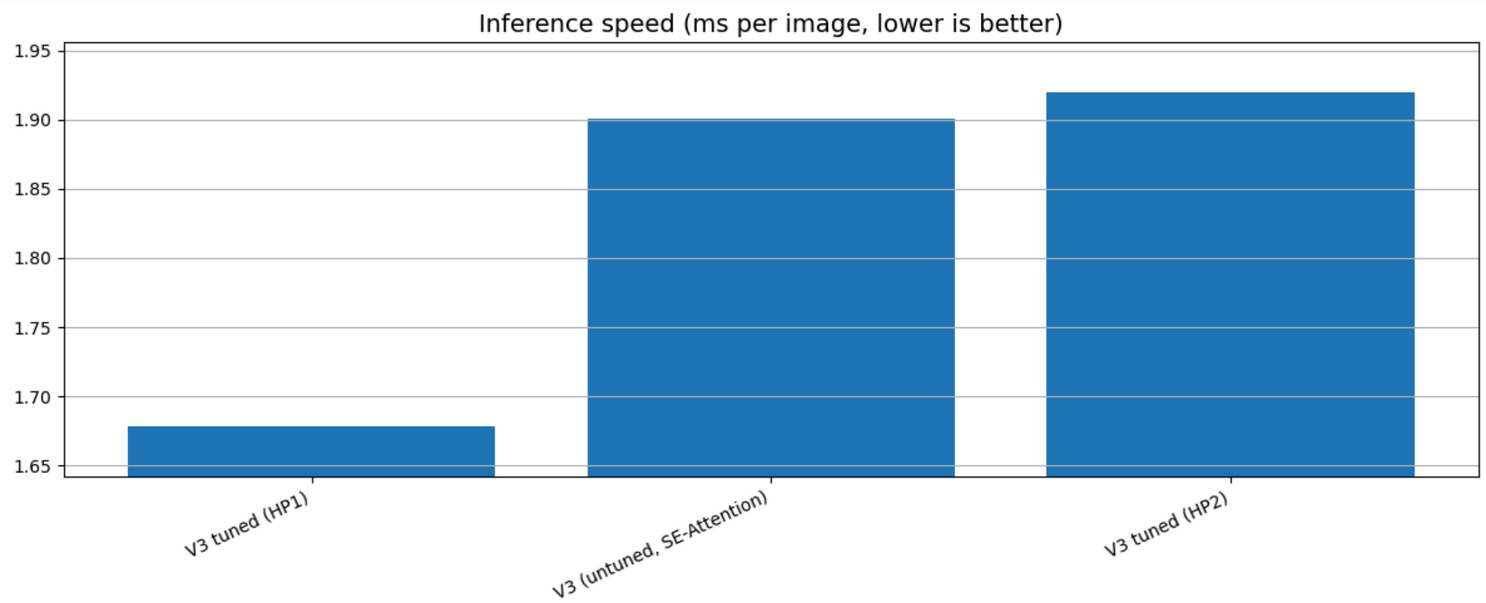
MODEL COMPARISON VISUALISATION AND SELECTION



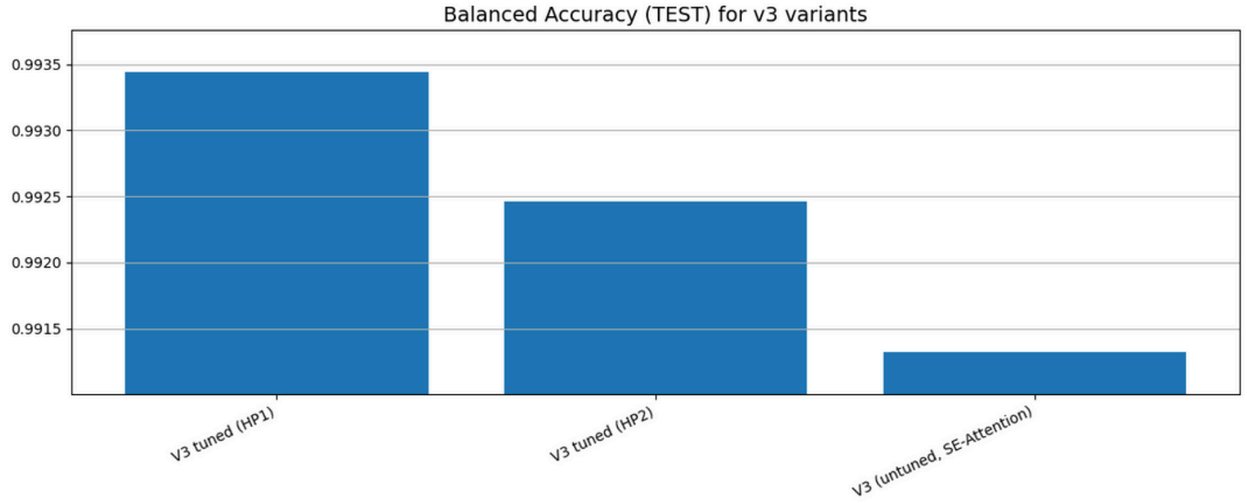


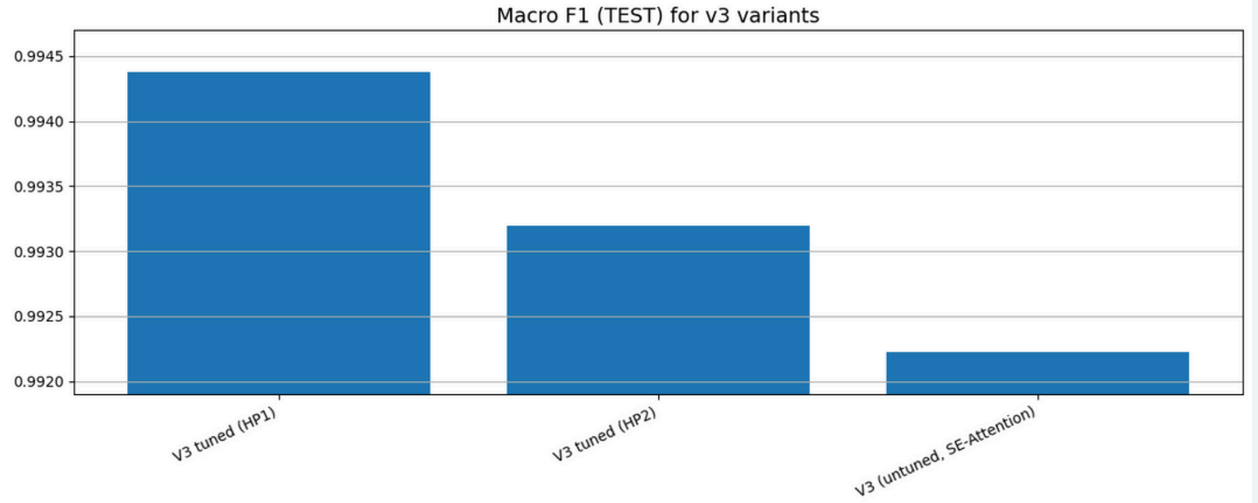


MODEL COMPARISON VISUALISATION AND SELECTION



	Model	Evaluated On	Class (ref)	Precision	Recall	F1	Support
0	V3 (untuned, SE-Attention)	test_gen_base	onion_brown	0.993789	0.993789	0.993789	161
1	V3 (untuned, SE-Attention)	test_gen_base	onion_purple	1.000000	0.979310	0.989547	145
2	V3 (untuned, SE-Attention)	test_gen_base	peach	0.994475	1.000000	0.997230	180
3	V3 (untuned, SE-Attention)	test_gen_base	unknown	0.984496	0.992188	0.988327	256
4	V3 tuned (HP1)	test_gen_v3_tuned_hp1	onion_brown	1.000000	1.000000	1.000000	161
5	V3 tuned (HP1)	test_gen_v3_tuned_hp1	onion_purple	0.993007	0.979310	0.986111	145
6	V3 tuned (HP1)	test_gen_v3_tuned_hp1	peach	1.000000	0.994444	0.997214	180
7	V3 tuned (HP1)	test_gen_v3_tuned_hp1	unknown	0.988417	1.000000	0.994175	256
8	V3 tuned (HP2)	test_gen_v3_tuned_hp2	onion_brown	1.000000	1.000000	1.000000	161
9	V3 tuned (HP2)	test_gen_v3_tuned_hp2	onion_purple	0.993007	0.979310	0.986111	145
10	V3 tuned (HP2)	test_gen_v3_tuned_hp2	peach	0.994444	0.994444	0.994444	180
11	V3 tuned (HP2)	test_gen_v3_tuned_hp2	unknown	0.988372	0.996094	0.992218	256





MODEL COMPARISON VISUALISATION AND SELECTION

Training Behaviour

- Untuned V3 converges fast but shows unstable validation loss due to unsuitable hyperparameters
- HP1 is smooth and well aligned, indicating strong generalisation
- HP2 is stable but slower with higher final loss

```
=== V3 VARIANTS: TEST SUMMARY (ALL MODELS, sorted by Balanced Acc then Macro F1) ===

Model: V3 tuned (HP1)
  Balanced Acc (test): 0.9934 | Macro F1 (test): 0.9944 | Acc (test): 0.9946 | Kappa (test): 0.9927
  Params: 676,548 | Speed: 1.678 ms/img
  Train Time (s): 1149.90
  Best Epoch (history): 22 | Hist Best Val Acc: 1.0000 | Hist Final Val Acc: 0.9987

Model: V3 tuned (HP2)
  Balanced Acc (test): 0.9925 | Macro F1 (test): 0.9932 | Acc (test): 0.9933 | Kappa (test): 0.9908
  Params: 1,486,900 | Speed: 1.920 ms/img
  Train Time (s): 2323.05
  Best Epoch (history): 87 | Hist Best Val Acc: 1.0000 | Hist Final Val Acc: 0.9946

Model: V3 (untuned, SE-Attention)
  Balanced Acc (test): 0.9913 | Macro F1 (test): 0.9922 | Acc (test): 0.9919 | Kappa (test): 0.9890
  Params: 662,756 | Speed: 1.901 ms/img
  Train Time (s): 687.95
  Best Epoch (history): 29 | Hist Best Val Acc: 0.9987 | Hist Final Val Acc: 0.9960

=== WINNER (by Balanced Acc, then Macro F1) ===
Model: V3 tuned (HP1)
Balanced Acc (test): 0.9934 | Macro F1 (test): 0.9944 | Acc (test): 0.9946
```

Inference Speed

- Fastest: HP1
- Then untuned V3
- Slowest: HP2

Balanced Accuracy and Macro F1

- Highest: HP1
- Then HP2
- Lowest: untuned V3

Class Level Insights

- HP1 achieves near perfect precision and recall across all classes
- Best performance on the Unknown class
- HP2 shows no per class gains despite higher complexity

Overall

- Best configuration: HP1
- Untuned is unstable, HP2 is conservative
- HP1 offers the best balance of stability, speed, and generalisation

WOW FACTOR: DATASET SPLIT COMPARISON

Purpose
- Test whether the model **generalises beyond familiar images**

Split A: Stratified (What We Used)
- Random, class-balanced **70/15/15** split
- Used for training and hyperparameter tuning

Split B: Context-Separated (Stress Test)
- Manual split by **background and capture context**
- All sets contain unseen environments

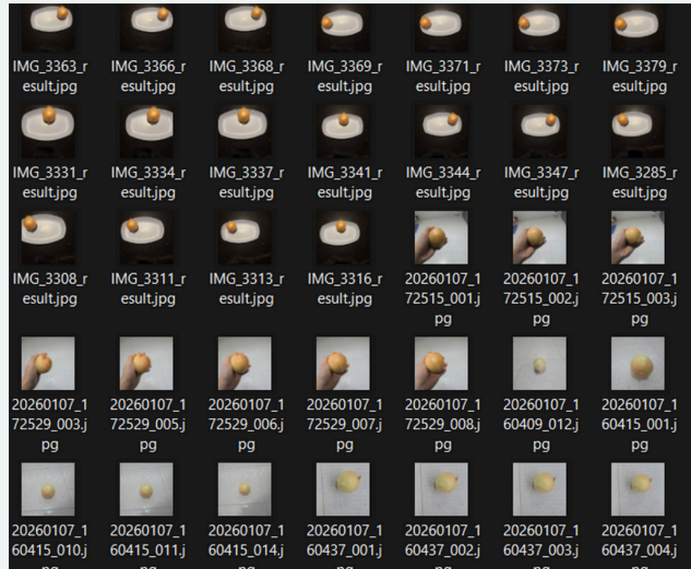
Why Both
- Prevents reliance on **background or lighting cues**
- Improves evaluation validity

Results
- **Near-perfect performance** across both splits
- Context-separated split is harder, with a small expected drop
- High cross-split consistency confirms **robust generalisation**

```
=== Split Robustness Comparison (2x2) ===
                              Run  Accuracy  Balanced Acc  F1 Macro  F1 Weighted
   Strat-trained on Strat-test  0.994609      0.993439  0.994375     0.994600
  Strat-trained on Manual-test  1.000000      1.000000  1.000000     1.000000
Manual-trained on Manual-test  0.960864      0.960919  0.961619     0.960841
  Manual-trained on Strat-test  0.986523      0.984527  0.986147     0.986506
```

Split A (Stratified): VAL

Split B (Manual): VAL



WOW FACTOR: MOBILE DETECTION APP OVERVIEW

Model
- Custom CNN with SE attention
- Converted to TensorFlow Lite for mobile deployment

App Stack
- Kotlin with Jetpack Compose
- CameraX for real time camera input

Processing Flow
- Live frames captured and processed in background
- On device preprocessing, centre crop, resize to 160 × 160, normalisation

Inference
- Predicts brown onion, purple onion, peach, or unknown
- Confidence scores generated

Output
- Prediction and confidence overlaid on camera view
- Near real time updates

Design Goals
- Fully offline, low latency
- Real time performance with reliable accuracy

PROJECT CONCLUSION AND FINAL RESULTS

Task

- 4-class classification using V3 CNN with SE-Attention
- Evaluated with Balanced Accuracy and Macro F1

Models compared

- V3 (untuned)
- V3 tuned (HP1, single-pass Hyperband)
- V3 tuned (HP2, two-stage tuning)

Best model: V3 tuned (HP1)

- Balanced Acc 99.34%
- Macro F1 99.44%
- Accuracy 99.46%
- ~676k parameters
- ~1.68 ms/image inference

Outcome

- HP1 outperformed HP2 and untuned V3 on all key metrics

Why HP1 won instead of HP2

- Early good hyperparameters beat longer training
- SE-Attention already provides strong representation
- Smaller model acts as implicit regulariser

HP1 vs HP2

- HP1: Efficient, compact, strong generalisation
- HP2: Larger, slower, no test-set gains

Final decision

- Use V3 tuned (HP1) as the final model

Key takeaway

- Optimisation strategy and metric choice matter more than raw model size

THANK YOU

NOTE: ADDITIONAL ANALYSES, EXPLANATIONS, AND EVALUATION GRAPHS ARE PROVIDED IN THE JUPYTER NOTEBOOK BUT OMITTED HERE FOR CONCISENESS.