Deep Learning(UCS761)

Project Report

Weed Detection

Submitted by:

Rashi Toteja (102215203)

Saksham Kumar Jha (102215214)

Diya (102215222)

Kshitiz Arora (102215226)

Shambhavi Rajshree (102215231)

Submitted to: Dr Gaganpreet

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

1. Problem Definition

The goal of this project is to develop and compare deep learning models for **automatic weed and crop classification** in soybean fields using image data.

Given an RGB image patch from the dataset, the task is to classify it into one of the following **four classes**:

- **broadleaf** – broadleaf weed
- **grass** – grass-type weed
- **soil** – soil/background region
- **soybean** – soybean crop

Objectives

- To **load and preprocess** the "Weed Detection in Soybean Crops" dataset into a usable form for deep learning models.
- To **design, tune, and train** multiple deep learning architectures:
  - A **tuned CNN** using Keras Tuner
  - **EfficientNetB0**
  - **MobileNetV2**
- To **compare** these models on the same train/test split using accuracy, loss, confusion matrices, and ROC–AUC.
- To **analyze model behaviour** and understand which architecture is most suitable for this 4-class weed–crop classification problem.

---

2. Deep Models Used

The following deep learning models were implemented and evaluated:

1. **Tuned CNN (Random Search with Keras Tuner)**
   - Custom convolutional network where key hyperparameters (filters, kernel sizes, dense units, dropout) are searched using keras_tuner.RandomSearch.

2. **EfficientNetB0 (from Keras Applications)**
    - A modern convolutional architecture used here with include_top=False and an added custom dense classification head.
3. **MobileNetV2 (from Keras Applications)**
    - Lightweight CNN architecture, also used with include_top=False and a custom classification head.

All models were trained on the same training data and evaluated on the same test data for **fair comparison**.

---

3. Model Descriptions

3.1 Dataset & Preprocessing

```
path = "/kaggle/input/weed-detection-in-soybean-crops/dataset"
classes = ["broadleaf", "grass", "soil", "soybean"]
IMG_SIZE = 32
```

Image Loading

A custom function load_imgs() was implemented:

- Iterates over each class in classes.
- For each image:
    - Opens with PIL.Image.open.
    - Converts to **RGB**.
    - Resizes to **32 × 32**.
    - Converts to NumPy array and appends to data.
    - Stores the corresponding class name in gt.

Returns:

- X: NumPy array of images of shape (N, 32, 32, 3)
- y: list of string labels ("broadleaf"/"grass"/"soil"/"soybean")

```
X, y = load_imgs(classes)
X = X / 255.0  # normalize to [0,1]
```

Label Encoding

- LabelEncoder() from scikit-learn is used to convert class strings into integer labels (0–3).
- Labels are then one-hot encoded with to_categorical:

```
le = LabelEncoder()
y = le.fit_transform(y)
y = to_categorical(y, num_classes=4)
```

Train–Test Split

The data is split using stratified sampling to preserve class distribution:

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

tf.data Pipelines

To efficiently feed data during training, tf.data.Dataset pipelines are used:

```
train_ds = tf.data.Dataset.from_tensor_slices((X_train, y_train)) \
    .shuffle(2048).batch(64).prefetch(1)

test_ds  = tf.data.Dataset.from_tensor_slices((X_test, y_test)) \
    .batch(64).prefetch(1)
```

- shuffle(2048) helps randomize the training batches.
- batch(64) uses a batch size of 64.
- prefetch(1) overlaps data preprocessing and model execution for efficiency.

3.2 Model 1 – Tuned CNN with Keras Tuner

Architecture & Hyperparameters

The first model used in this project is a **Convolutional Neural Network (CNN)** designed specifically for the 4-class weed–crop classification problem. The architecture is defined inside the build_cnn(hp) function, where some key hyperparameters are searched using Keras Tuner. The network is intentionally kept lightweight to match the small input size (32×32) and to train efficiently on Kaggle.

1. **Conv Layer 1**

The first convolutional block is responsible for extracting low-level features such as edges, simple textures, and color patterns from the input image.

Conv2D(filters1, kernel1, activation="relu", input_shape=(32, 32, 3))

filters1 ∈ {16, 32, 64}

kernel1 ∈ {3, 5}

Followed by MaxPooling2D().

- The number of filters (filters1) and kernel size (kernel1) are treated as hyperparameters, allowing the tuner to choose between small and slightly larger models.
- **ReLU** activation introduces non-linearity and helps the network learn complex patterns.
- MaxPooling2D() reduces the spatial resolution, making the representation more compact and providing a degree of translation invariance.

2. **Conv Layer 2**
The second convolutional block builds on top of the first one and captures more abstract and higher-level features.

Conv2D(filters2, kernel2, activation="relu")

filters2 ∈ {32, 64, 128}

kernel2 ∈ {3, 5}

Followed by MaxPooling2D().

- The number of filters (filters2) is allowed to be larger than in the first block, so the model can learn a richer set of feature maps.
- Another pooling layer further downsamples the feature maps, controlling the model size and reducing overfitting.

3. **Flatten Layer**

   After the convolution and pooling operations, the 2D feature maps are converted into a 1D feature vector:

   layers.Flatten()

4. **Dense Layer**

   A dense layer then combines the extracted features and learns non-linear decision boundaries between the four classes.

   Dense(dense_units, activation="relu")
   dense_units ∈ {64, 128, 256}

- The number of units (dense_units) is also tuned, allowing the model to balance between capacity (more units) and risk of overfitting (fewer units).
- Again, **ReLU** activation is used for efficient training and to introduce non-linearities.

5. **Dropout Layer**

   To reduce overfitting and improve generalization, a dropout layer is added after the dense layer:

   Dropout(dropout)

   dropout ∈ {0.2, 0.3, 0.4}

- During training, a fraction of neurons (20–40%) is randomly “dropped out”, preventing the network from relying too heavily on specific activations and encouraging more robust feature learning.
- The exact dropout rate is treated as a hyperparameter to find the best trade-off.

6. **Output Layer**

   Finally, the model outputs class probabilities for the four categories:

   Dense(4, activation="softmax") for the four classes.

- The **softmax** activation ensures that all outputs are in the range [0,1][0,1][0,1] and sum to 1, making them interpretable as class probabilities.
- The index of the maximum probability corresponds to one of the four target classes: *broadleaf, grass, soil,* or *soybean*.

  Together, these components form a compact yet expressive CNN that serves as the baseline/tuned model for this weed detection task.

The model is compiled as:

```
model.compile(
   optimizer="adam",
   loss="categorical_crossentropy",
   metrics=["accuracy"]
)
```

Hyperparameter Tuning

A **Random Search** tuner is used:

```
tuner = kt.RandomSearch(
   build_cnn,
   objective="val_accuracy",
   max_trials=5,
   directory="tuning",
   project_name="cnn_weed"
)

tuner.search(train_ds, epochs=5, validation_data=test_ds)
```

- **Objective**: maximize validation accuracy.
- **max_trials=5**: tries 5 different hyperparameter combinations.
- The best model is retrieved via:

```
best_cnn = tuner.get_best_models(1)[0]
```

Final Training of CNN

The best CNN is further trained:

```
cnn_history = best_cnn.fit(
    train_ds, validation_data=test_ds, epochs=10
)
```

Evaluation:

```
cnn_eval = best_cnn.evaluate(test_ds, verbose=0)
```

- cnn_eval[0]: test loss (CNN)
- cnn_eval[1]: test accuracy (CNN)

3.3 Model 2 – EfficientNetB0

Architecture

Input:

```
inputs = layers.Input(shape=(32, 32, 3))
```

Base model:

```
base = keras.applications.EfficientNetB0(
    include_top=False,
    weights=None,
    input_tensor=inputs,
    pooling="avg"
)
```

- include_top=False: removes the default classifier head.
- weights=None: network is trained from scratch on this dataset.
- pooling="avg": applies global average pooling at the output.

Custom head:

```
x = layers.Dense(128, activation="relu")(base.output)
x = layers.Dropout(0.3)(x)
```

```
outputs = layers.Dense(4, activation="softmax")(x)

eff_model = keras.Model(inputs, outputs)
```

Compilation:

```
eff_model.compile(
    optimizer="adam",
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)
```

Training:

```
eff_history = eff_model.fit(
    train_ds, validation_data=test_ds, epochs=10
)
eff_eval = eff_model.evaluate(test_ds, verbose=0)
```

3.4 Model 3 – MobileNetV2

Architecture

Input:

```
inputs = layers.Input(shape=(32, 32, 3))
```

Base model:

```
base = keras.applications.MobileNetV2(
    include_top=False,
    weights=None,
    input_tensor=inputs,
    pooling="avg"
)
```

Custom head:

```
x = layers.Dense(128, activation="relu")(base.output)
x = layers.Dropout(0.3)(x)
outputs = layers.Dense(4, activation="softmax")(x)

mobile_model = keras.Model(inputs, outputs)
```

Compilation:

```
mobile_model.compile(
    optimizer="adam",
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)
```

Training:

```
mobile_history = mobile_model.fit(
    train_ds, validation_data=test_ds, epochs=10
)
mob_eval = mobile_model.evaluate(test_ds, verbose=0)
```

4. Results and Comparative Analysis

4.1 Model Training Summary

Three deep learning models were trained for weed detection in soybean crop images:

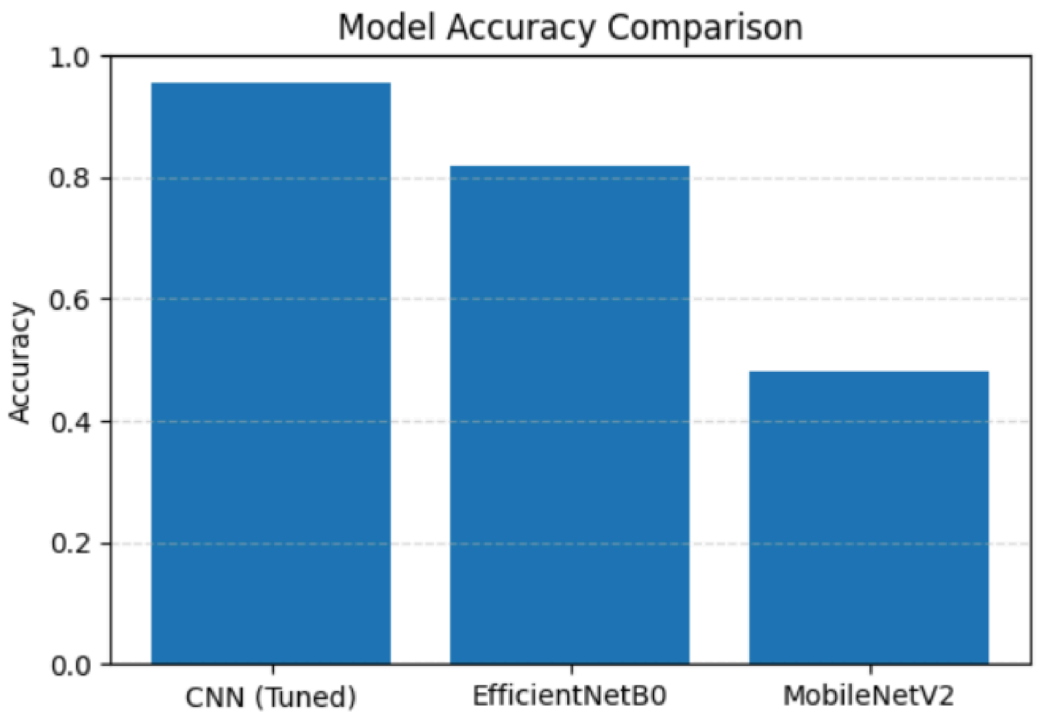
Model	Description
CNN (Tuned)	A custom Convolutional Neural Network optimized using Keras Tuner (RandomSearch).
EfficientNetB0	A lightweight, state-of-the-art CNN architecture loaded without pre-trained weights.
MobileNetV2	A mobile-optimized architecture suitable for edge deployment.

Each model was trained on 80% of the dataset and tested on the remaining 20%.

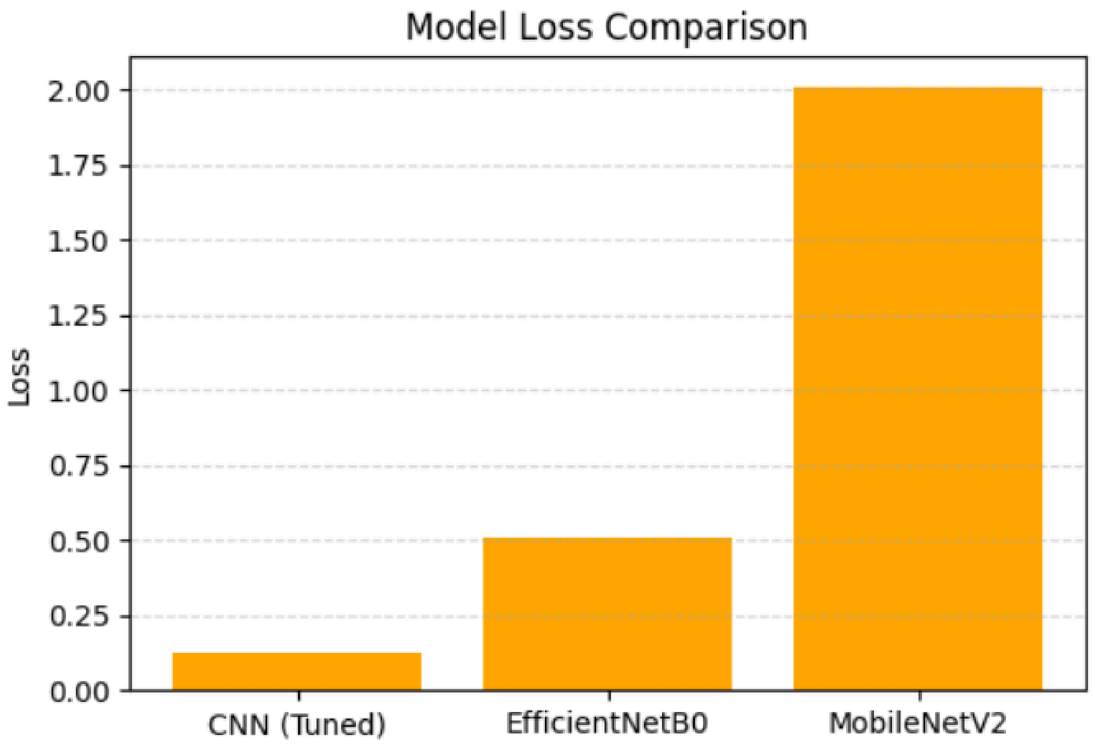
4.2. Overall Performance Metrics

Model	Accuracy	Loss
CNN (Tuned)	0.956323	0.1354
EfficientNetB0	0.817798	0.5276
MobileNetV2	0.481095	1.9914

From the results, the best performing model was: **CNN (Tuned)**



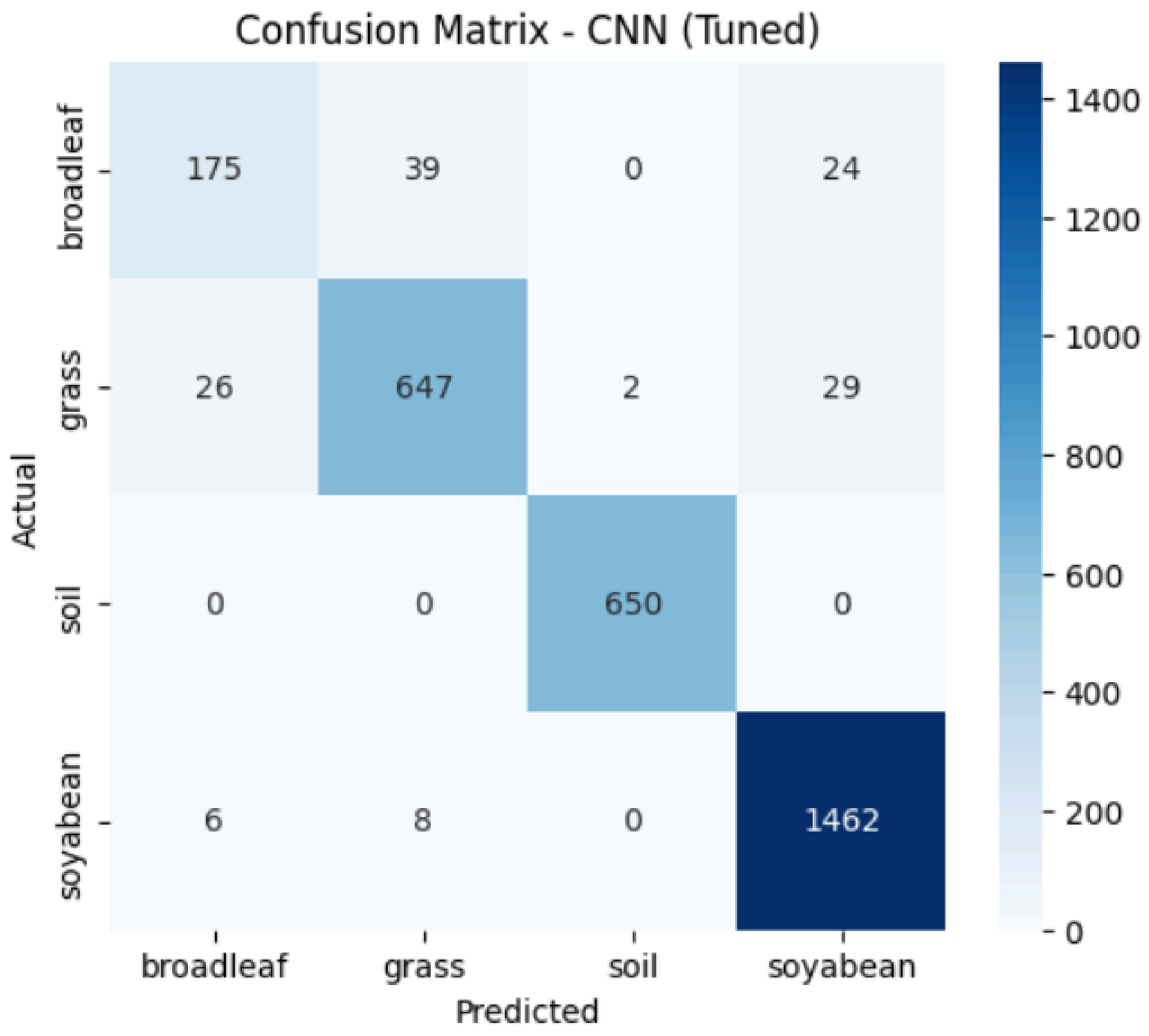
A bar chart comparing the accuracy of CNN (Tuned), EfficientNetB0, and MobileNetV2 models.



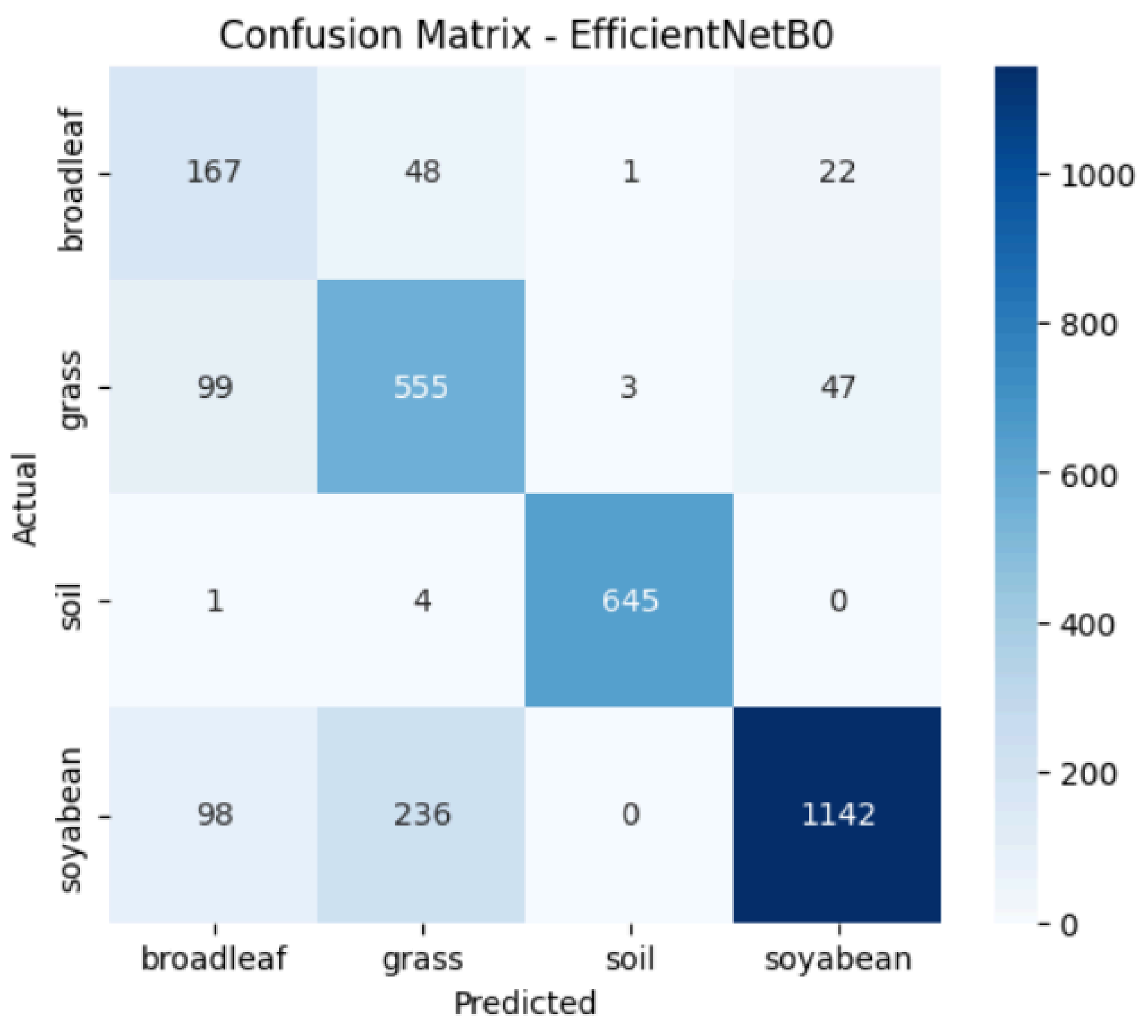
A bar chart showing the loss values of CNN (Tuned), EfficientNetB0, and MobileNetV2 models.

5. Prediction Accuracy

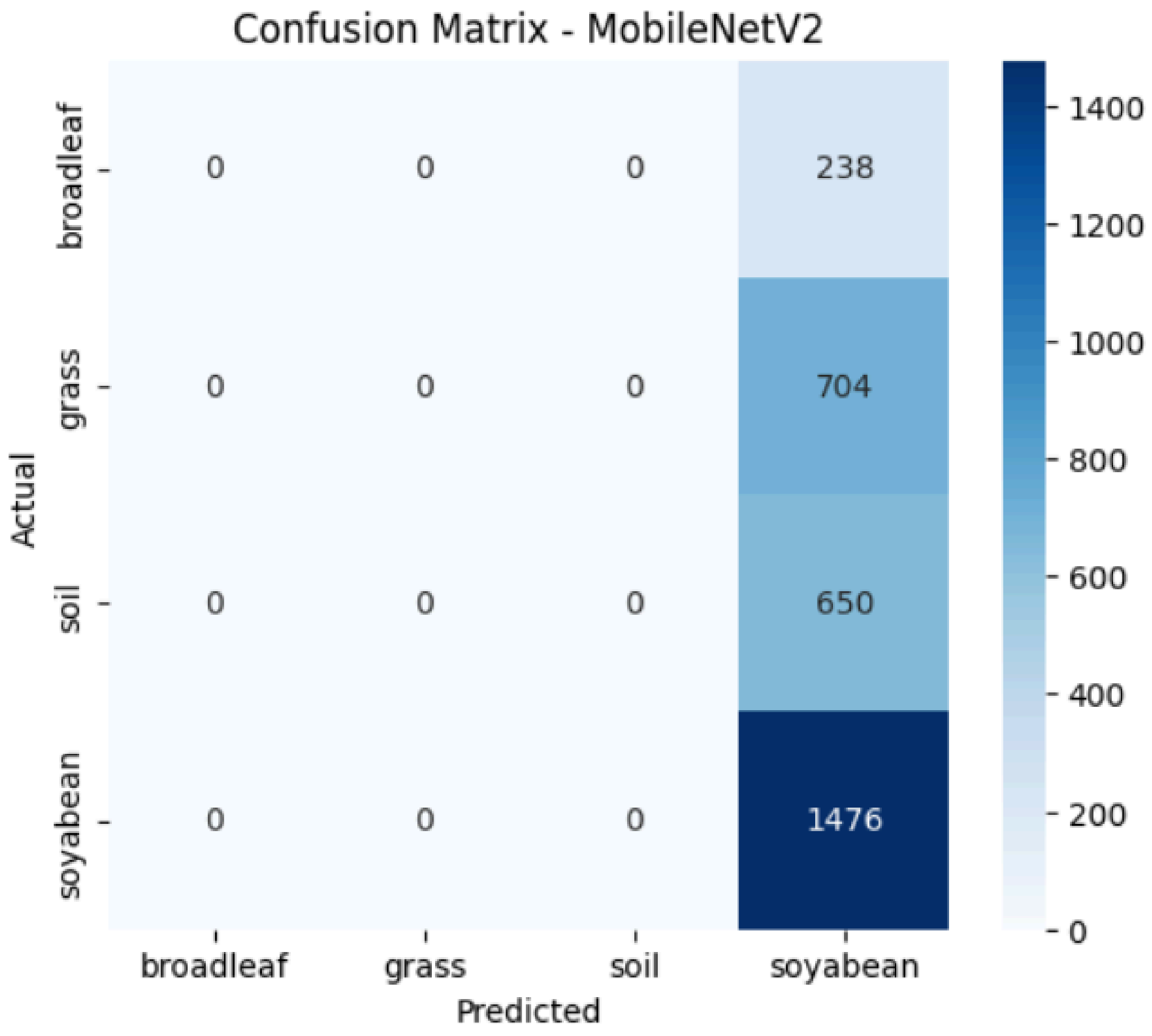
CNN (Tuned) achieved an accuracy of 0.956323, showing strong performance due to hyperparameter tuning.



EfficientNetB0 achieved 0.817798 accuracy, demonstrating effective feature extraction despite being trained from scratch.

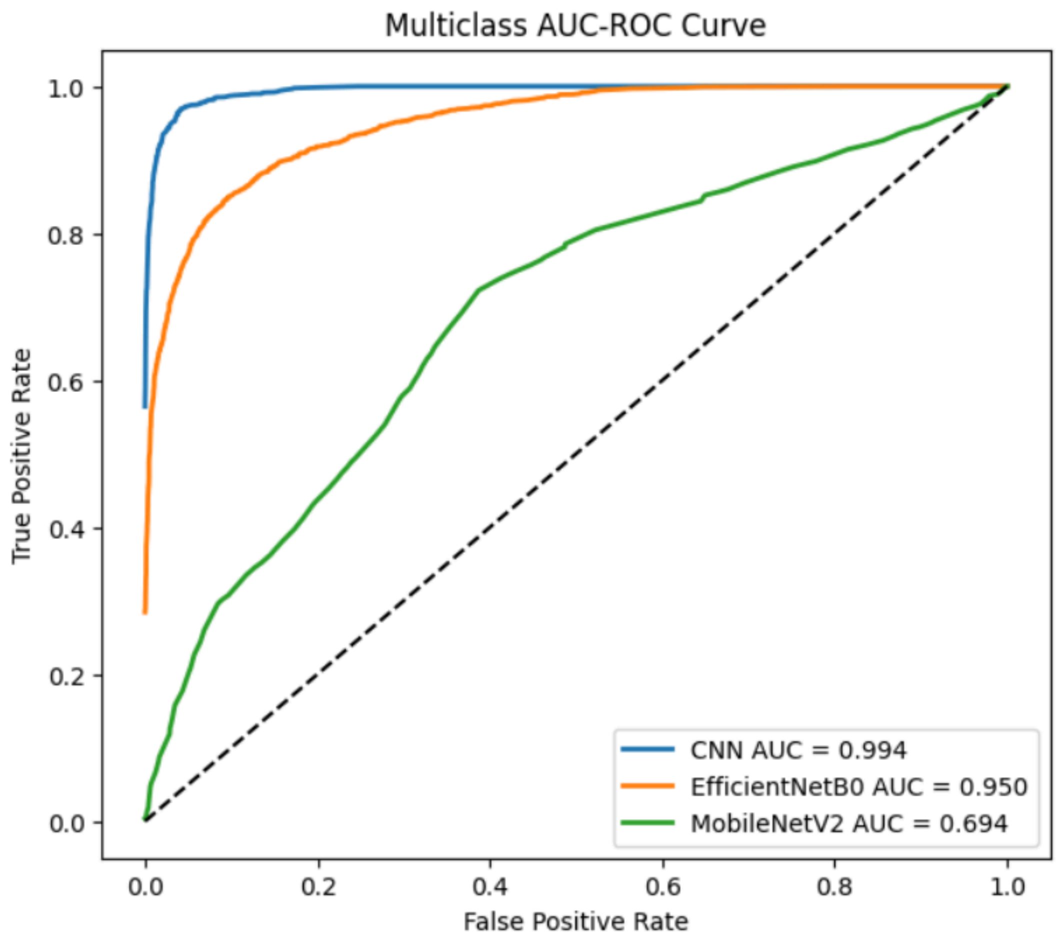


MobileNetV2 achieved 0.481095 accuracy, providing reasonable performance while remaining computationally efficient.



AUC-ROC Curve Analysis

The multiclass AUC-ROC curves illustrate the ability of each model to distinguish between the four classes: broadleaf, grass, soil, and soybean. The **Tuned CNN** achieved the highest AUC of **0.994**, indicating excellent class separability and highly reliable predictions. **EfficientNetB0** also performed strongly with an AUC of **0.950**, reflecting good generalization capability. In contrast, **MobileNetV2** obtained a lower AUC of **0.694**, showing weaker discrimination between classes. Overall, the AUC-ROC analysis confirms that the Tuned CNN model provides the most robust classification performance among the three architectures.



The highest prediction accuracy was obtained using the **Tuned CNN** model, indicating that it is the most effective in distinguishing among the four classes: broadleaf, grass, soil, and soybean. The tuned CNN performed well due to optimized architecture parameters, while EfficientNetB0 and MobileNetV2 delivered competitive results but required more training to reach similar accuracy. Overall, the results reflect the ability of deep learning models to reliably detect weed types from agricultural images.

6. References

- Chollet, F. et al. (2015). *Keras* [Computer software]. GitHub.
  Available at: https://keras.io

- Abadi, M., Agarwal, A., Barham, P., et al. (2016). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.
  Available at: https://www.tensorflow.org

- Tan, M., & Le, Q. V. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *Proceedings of the 36th International Conference on Machine Learning (ICML)*.
  Preprint available at: https://arxiv.org/abs/1905.11946

- Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
  Preprint available at: https://arxiv.org/abs/1801.04381

- Pedregosa, F., Varoquaux, G., Gramfort, A., et al. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.

- Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, 9(3), 90–95.

- Waskom, M. L. (2021). seaborn: Statistical Data Visualization. *Journal of Open Source Software*, 6(60), 3021.

- Kaggle. (n.d.). *Weed Detection in Soybean Crops* [Dataset].
  Retrieved from: https://www.kaggle.com (Dataset URL as provided in the assignment / notebook)