Computer Vision 2025 Project [SC]

Plant Classification and Disease Recognition

SC_16

Team members:

Section 6	2021170538	ملك رأفت فؤاد
Section 1	2021170161	حبيبة علاء الدين عادل
Section 4	2021170423	مارك جمال أنور
Section 2	2021170195	رنا وحيد تمام
Section 4	2021170421	ماجد سيد محمود
Section 4	2021170442	مايكل نادر عماد الدين

Under Supervision of Dr. Dina Khatab

First : Plant Classification

1. MobileNet
2. ViT (Vision Transformer)
3. VGG (Visual Geometry Group)
4. AlexNet

Second: Plant Disease Recognition

1. Siamese

Third: Segmentation

1. SAM (Segment Anything Model)
2. U-net (U-shaped Network)

First : Plant Classification

Data Preparation

1. Setting Constants

image_size: Defines the target size to which images will be resized (224x224 pixels).

batch_size: The number of images to process at once during training. (64 batches)

2. Loading Data

This function loads images and their corresponding labels from the specified directory.

3. Normalizing Images

The images are normalized by dividing each pixel's value by 255.0, converting the pixel values from the range [0, 255] to [0, 1]. This step helps improve the convergence during model training.

4. Splitting the Data into Train and Validation Sets

- This splits the loaded data into training and validation sets, with 80% of the data used for training and 20% for validation

7. Label Encoding

- The LabelEncoder is used to convert text labels into integer labels

8. Class Weight Calculation

- The class weights are computed to handle class imbalance. The compute_class_weight function calculates weights inversely proportional to class frequencies.

10. Data Augmentation

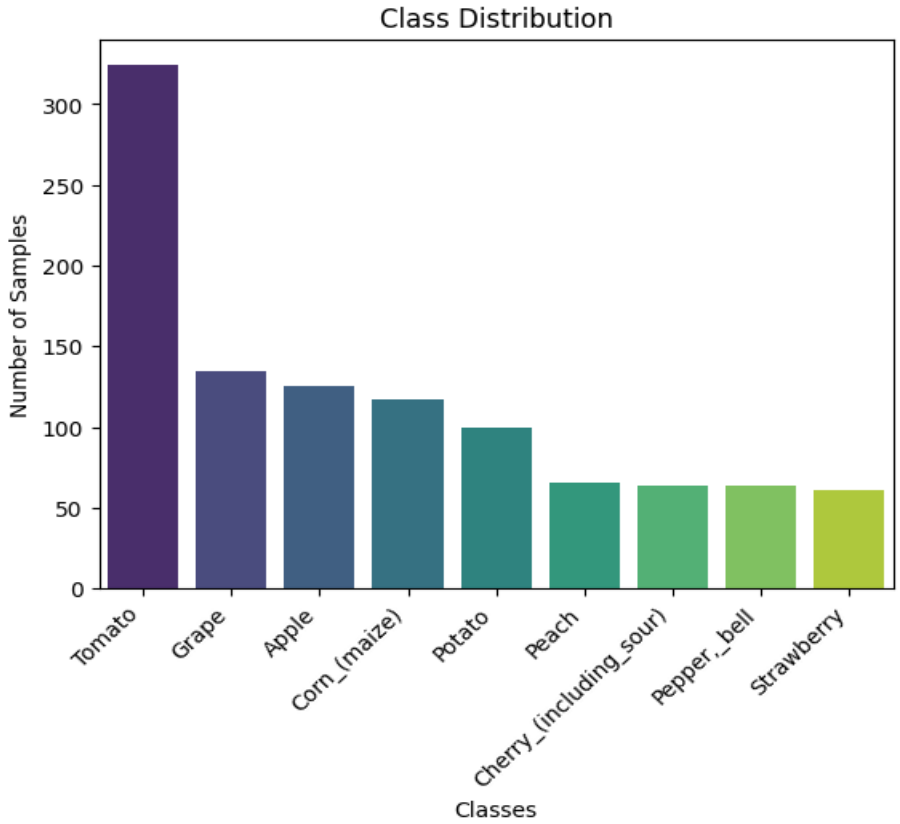
We **apply augmentation** only in **MobileNet, VGG, AlexNet**, as **we don’t apply augmentation** in **ViT**. The train_datagen applies several data augmentation techniques to the training images, including: Random rotation (rotation_range), width/height shifting, shearing, zooming, and flipping the images horizontally. val_datagen doesn't apply any augmentation

These augmentations help improve model generalization by providing more varied input data.

11. Data Generators

train_generator and **validation_generator** and **testing_generator** are instances of ImageDataGenerator that yield batches of images and labels during training and validation, respectively.

Data Visualization



Vision Transformer (ViT)

Architecture

- Divides input images into fixed-size non-overlapping patches (e.g., 16×1616 \times 1616×16).
- Converts patches into 1D vector embeddings via a linear projection.
- Adds positional embeddings to preserve spatial relationships.
- Processes embeddings using Transformer encoder layers (self-attention + feedforward networks).
- Uses a learnable [CLS] token for classification.
- Final output is passed to a classification head for tasks like image classification.

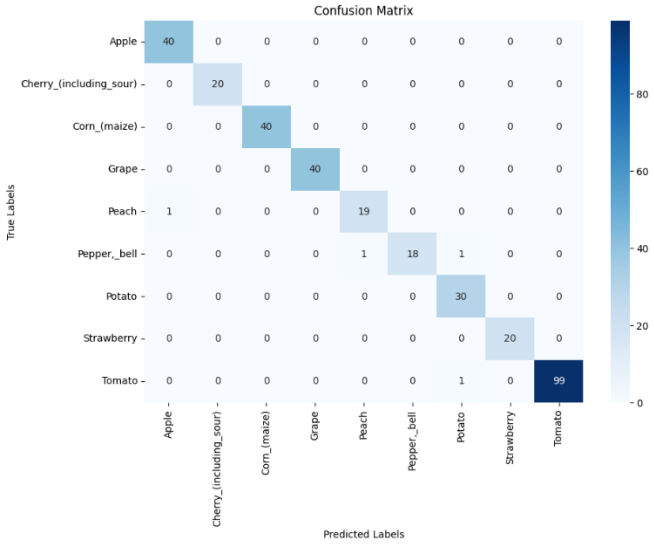
Advantages

- **Scalability**: Performs better with larger datasets.
- **Global Context**: Captures global relationships across the image.
- **Flexibility**: Can adapt to multi-modal tasks beyond vision (e.g., vision + text).
- **Reduced Inductive Bias**: Learns more adaptively compared to CNNs.
- **Improved Performance**: Outperforms CNNs on benchmarks when pre-trained on large datasets.
- **Parallelization**: Faster training due to sequence-level parallel processing.
- **Transfer Learning**: Pre-trained ViTs generalize well to other tasks.

Challenges

- **Data Requirements**: Needs large-scale datasets for effective training.
- **Computational Cost**: High memory and computation demands due to quadratic self-attention complexity.
- **Overfitting**: Prone to overfitting on smaller datasets.
- **Interpretability**: Harder to interpret learned features compared to CNNs.

ViT: **accuracy** and **time** during training and testing



Epoch	Training Loss	Validation Loss	Training Accuracy	Validation Accuracy
1	1.929000	1.083248	0.823864	0.594697
2	0.700100	0.354976	0.987689	0.810606
3	0.169800	0.100587	0.999053	0.905303
4	0.073400	0.034159	1.000000	0.928030
5	0.022500	0.016373	1.000000	0.928030
6	0.012300	0.010918	1.000000	0.939394
7	0.010600	0.008500	1.000000	0.939394
8	0.007800	0.006946	1.000000	0.946970
9	0.006300	0.005979	1.000000	0.950758
10	0.005300	0.005264	1.000000	0.958333
11	0.005000	0.004696	1.000000	0.954545
12	0.004300	0.004301	1.000000	0.958333
13	0.003900	0.003975	1.000000	0.958333
14	0.004000	0.003715	1.000000	0.958333
15	0.003500	0.003523	1.000000	0.958333
16	0.003300	0.003375	1.000000	0.958333
17	0.003400	0.003262	1.000000	0.958333
18	0.003100	0.003186	1.000000	0.958333
19	0.003100	0.003138	1.000000	0.958333
20	0.003100	0.003122	1.000000	0.958333

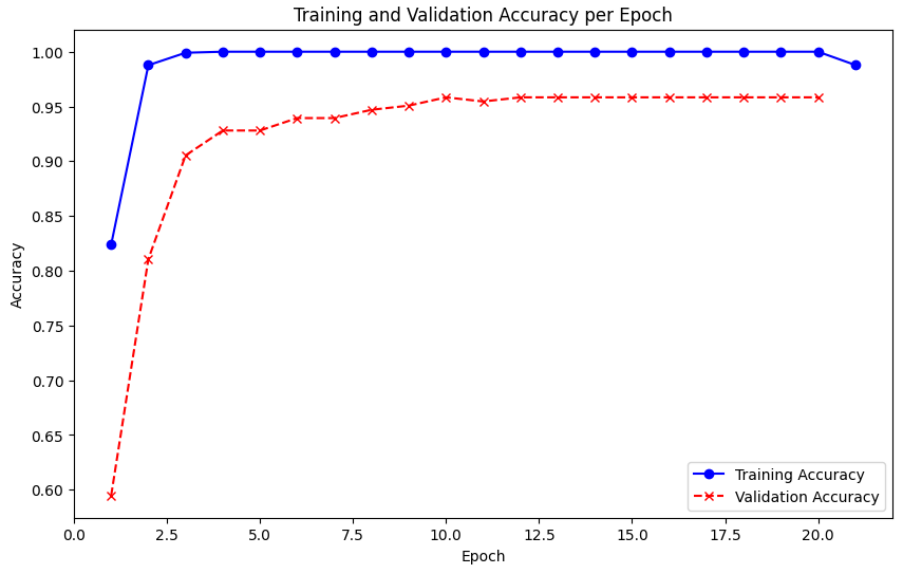
Classification Report of ViT:

	precision	recall	f1-score	support
Apple	0.98	1.00	0.99	40
Cherry_(including_sour)	1.00	1.00	1.00	20
Corn_(maize)	1.00	1.00	1.00	40
Grape	1.00	1.00	1.00	40
Peach	0.95	0.95	0.95	20
Pepper,_bell	1.00	0.90	0.95	20
Potato	0.94	1.00	0.97	30
Strawberry	1.00	1.00	1.00	20
Tomato	1.00	0.99	0.99	100
accuracy			0.99	330
macro avg	0.98	0.98	0.98	330
weighted avg	0.99	0.99	0.99	330

```
Training Time: 549.77 seconds
Training Accuracy: 0.9905

Validation Time: 454.55 seconds
Validation Accuracy: 0.9333

Testing Accuracy: 0.99
```



MobileNet

MobileNet is a lightweight deep learning model designed for mobile and embedded devices, prioritizing efficiency and speed. It uses depthwise separable convolutions to reduce the number of parameters and computations. This architecture is well-suited for tasks like image classification and object detection on resource-constrained devices. Despite its simplicity, it achieves competitive accuracy compared to larger models.

Architecture:

Input: Images of size **224x224x3** (RGB).

Base Model:

- **MobileNet** (pre-trained on ImageNet, without the top classification layers).
- Lightweight and efficient architecture, designed with depthwise separable convolutions for reduced computational complexity.
- Base model layers are frozen (not trainable).

Custom Layers:

- Global Average Pooling (GAP): Reduces the spatial dimensions of the feature maps to a single vector for each channel, summarizing the spatial information globally.
- Dense Layer: Fully connected layer with 1024 units and ReLU activation.
- Dropout Layer: Dropout with a rate of 0.5 to reduce overfitting.
- Output Layer: Dense layer with num_classes units and softmax activation for classification.

Optimization:

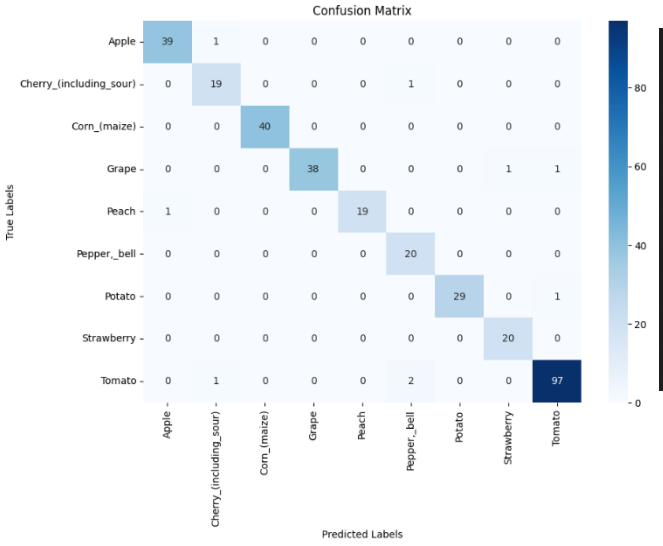
- Uses Adam optimizer, categorical cross-entropy loss, and accuracy as a performance metric.

Output: Class probabilities for the given number of output classes

Best for Resource-Constrained Devices: MobileNet

- **Why:** MobileNet is optimized for efficiency and speed, making it ideal for mobile and embedded devices. Despite its smaller size, it delivers competitive performance on tasks like image classification and object detection**.**

MobileNet: **accuracy** and **time** during training and testing



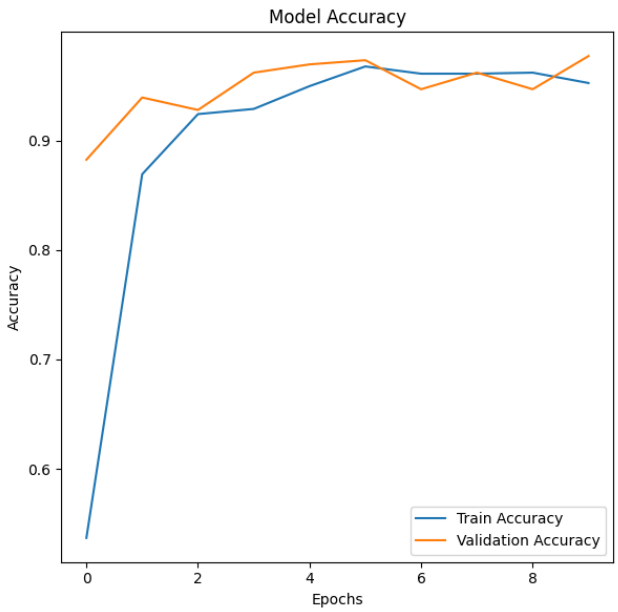

```
1/17 ━━━━━━━━━━━━━━━━━━━━ 3:27 13s/step - accuracy: 0.0469 - loss: 2.8073
I0000 00:00:1734538685.973019      91 device_compiler.h:188] Compiled cluster using XLA!  This line is logged at most
17/17 ━━━━━━━━━━━━━━━━━━━━ 24s 663ms/step - accuracy: 0.3641 - loss: 2.2047 - val_accuracy: 0.8826 - val_loss: 0.3543
Epoch 2/10
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 366ms/step - accuracy: 0.8486 - loss: 0.4785 - val_accuracy: 0.9394 - val_loss: 0.2005
Epoch 3/10
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 367ms/step - accuracy: 0.9137 - loss: 0.2727 - val_accuracy: 0.9280 - val_loss: 0.2055
Epoch 4/10
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 365ms/step - accuracy: 0.9209 - loss: 0.1726 - val_accuracy: 0.9621 - val_loss: 0.1049
Epoch 5/10
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 371ms/step - accuracy: 0.9543 - loss: 0.1235 - val_accuracy: 0.9697 - val_loss: 0.0959
Epoch 6/10
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 369ms/step - accuracy: 0.9604 - loss: 0.1105 - val_accuracy: 0.9735 - val_loss: 0.0845
Epoch 7/10
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 372ms/step - accuracy: 0.9613 - loss: 0.0992 - val_accuracy: 0.9470 - val_loss: 0.1212
Epoch 8/10
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 370ms/step - accuracy: 0.9542 - loss: 0.1082 - val_accuracy: 0.9621 - val_loss: 0.0791
Epoch 9/10
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 389ms/step - accuracy: 0.9567 - loss: 0.1100 - val_accuracy: 0.9470 - val_loss: 0.1221
Epoch 10/10
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 387ms/step - accuracy: 0.9487 - loss: 0.0915 - val_accuracy: 0.9773 - val_loss: 0.0619
```

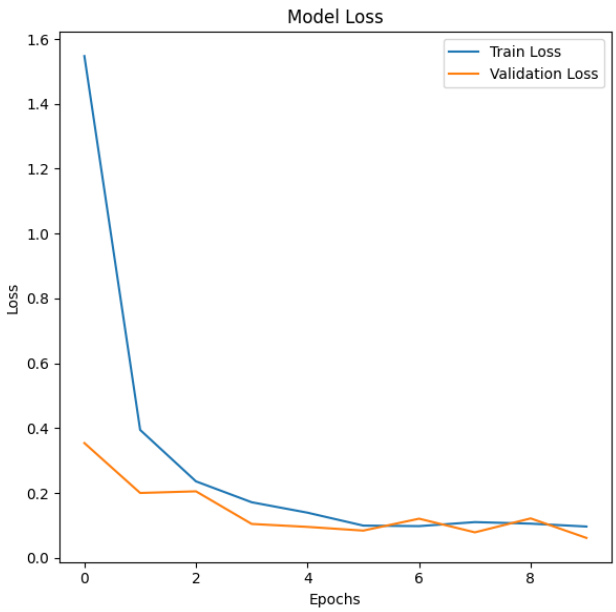
```
Training Time: 122.75 seconds
Training Accuracy: 0.90
```

```
Testing Time: 1.93 seconds
Test Accuracy: 0.97, Test Loss: 0.08
```

Classification Report of MobileNET:

	precision	recall	f1-score	support
Apple	0.97	0.97	0.97	40
Cherry_(including_sour)	0.90	0.95	0.93	20
Corn_(maize)	1.00	1.00	1.00	40
Grape	1.00	0.95	0.97	40
Peach	1.00	0.95	0.97	20
Pepper,_bell	0.87	1.00	0.93	20
Potato	1.00	0.97	0.98	30
Strawberry	0.95	1.00	0.98	20
Tomato	0.98	0.97	0.97	100
accuracy			0.97	330
macro avg	0.96	0.97	0.97	330
weighted avg	0.97	0.97	0.97	330





VGG16

VGG is a deep convolutional neural network known for its simplicity and uniform architecture, consisting of sequential 3x3 convolutional layers followed by fully connected layers. It comes in variations like VGG-16 and VGG-19, named for the number of layers. VGG models are computationally expensive but deliver high accuracy in image classification. Their deep and uniform structure has influenced the design of many subsequent models.

Architecture:

Input:

- Images of size **224x224x3** (RGB).

Base Model:

- **VGG16** (pre-trained on ImageNet, without the top classification layers).
- Contains 13 convolutional layers grouped into 5 blocks, each followed by max-pooling layers for feature extraction.

Custom Layers:

- Flatten: Converts feature maps from VGG16 into a 1D vector.
- Dense Layer 1: Fully connected layer with 4096 units and ReLU activation.
- Dropout Layer 1: Dropout with a rate of 0.5 to reduce overfitting.
- Dense Layer 2: Fully connected layer with 4096 units and ReLU activation.
- Dropout Layer 2: Another dropout with a rate of 0.5.
- Output Layer: Dense layer with num_classes units and softmax activation for classification.

Optimization:

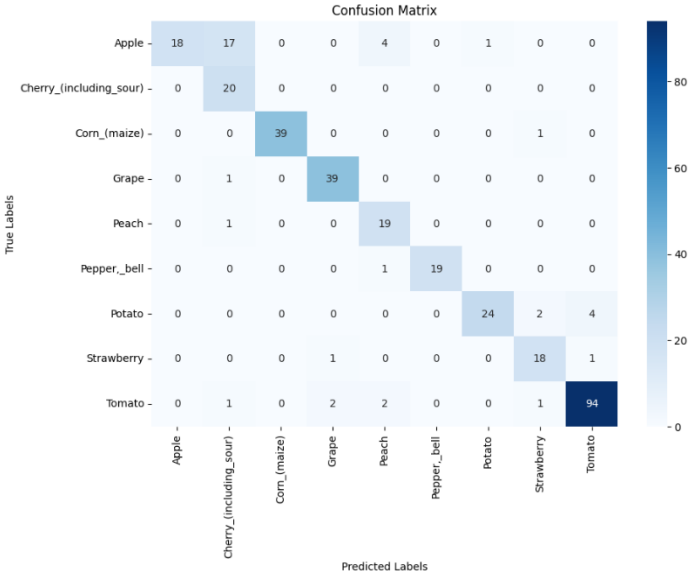
- Uses Adam optimizer, categorical cross-entropy loss, and accuracy as a performance metric.

Output: Class probabilities for the given number of output classes

Best for High Accuracy on Large Datasets: VGG

- **Why:** VGG models, particularly VGG-16 and VGG-19, provide high accuracy due to their deeper architecture and consistent design. They are well-suited for applications requiring precise feature extraction.

VGG : **accuracy** and **time** during training and testing




```
Epoch 1/10
17/17 ━━━━━━━━━━━━━━━━━━━━ 23s 545ms/step - accuracy: 0.1533 - loss: 19.3983 - val_accuracy: 0.2576 - val_loss: 1.9804
Epoch 2/10
17/17 ━━━━━━━━━━━━━━━━━━━━ 12s 420ms/step - accuracy: 0.2863 - loss: 1.9541 - val_accuracy: 0.5682 - val_loss: 1.1340
Epoch 3/10
17/17 ━━━━━━━━━━━━━━━━━━━━ 12s 427ms/step - accuracy: 0.5178 - loss: 1.2809 - val_accuracy: 0.7803 - val_loss: 0.7127
Epoch 4/10
17/17 ━━━━━━━━━━━━━━━━━━━━ 12s 448ms/step - accuracy: 0.6076 - loss: 1.0726 - val_accuracy: 0.7614 - val_loss: 0.6343
Epoch 5/10
17/17 ━━━━━━━━━━━━━━━━━━━━ 12s 421ms/step - accuracy: 0.6469 - loss: 0.9360 - val_accuracy: 0.7348 - val_loss: 0.6309
Epoch 6/10
17/17 ━━━━━━━━━━━━━━━━━━━━ 12s 423ms/step - accuracy: 0.6846 - loss: 0.9048 - val_accuracy: 0.8598 - val_loss: 0.4267
Epoch 7/10
17/17 ━━━━━━━━━━━━━━━━━━━━ 12s 425ms/step - accuracy: 0.7067 - loss: 0.7273 - val_accuracy: 0.8523 - val_loss: 0.4017
Epoch 8/10
17/17 ━━━━━━━━━━━━━━━━━━━━ 12s 421ms/step - accuracy: 0.7123 - loss: 0.7875 - val_accuracy: 0.8598 - val_loss: 0.4038
Epoch 9/10
17/17 ━━━━━━━━━━━━━━━━━━━━ 12s 439ms/step - accuracy: 0.7719 - loss: 0.7077 - val_accuracy: 0.8371 - val_loss: 0.4108
Epoch 10/10
17/17 ━━━━━━━━━━━━━━━━━━━━ 12s 456ms/step - accuracy: 0.7215 - loss: 0.7467 - val_accuracy: 0.8750 - val_loss: 0.3103
```

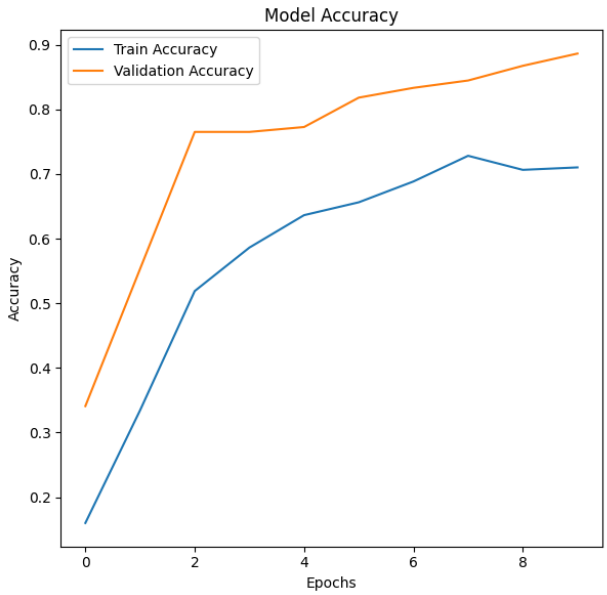
```
Training Time: 130.87 seconds
Training Accuracy: 0.60
```

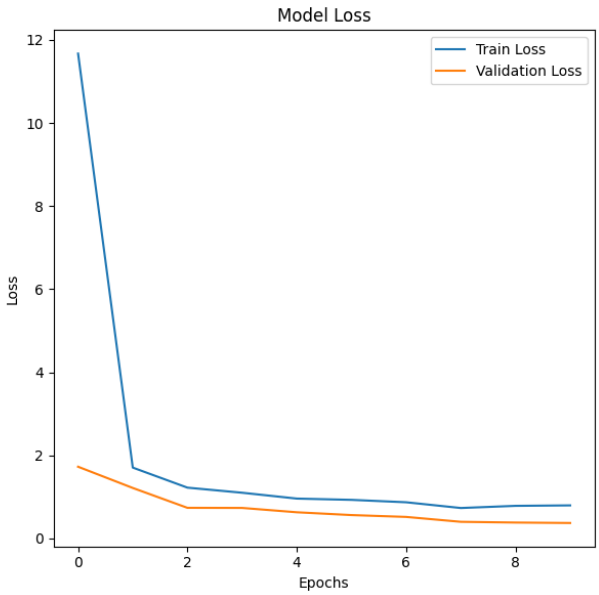
```
Testing Time: 1.10 seconds
Test Accuracy: 0.88, Test Loss: 0.35
```

```
Classification Report of VGG:
                        precision    recall  f1-score   support

                 Apple       1.00      0.45      0.62        40
Cherry_(including_sour)      0.50      1.00      0.67        20
            Corn_(maize)       1.00      0.97      0.99        40
                 Grape       0.93      0.97      0.95        40
                 Peach       0.73      0.95      0.83        20
          Pepper,_bell       1.00      0.95      0.97        20
                Potato       0.96      0.80      0.87        30
            Strawberry       0.82      0.90      0.86        20
                Tomato       0.95      0.94      0.94       100

              accuracy                           0.88       330
             macro avg       0.88      0.88      0.86       330
          weighted avg       0.91      0.88      0.88       330
```





AlexNet

AlexNet is a pioneering deep learning model that popularized convolutional neural networks in the 2012 ImageNet competition. It uses five convolutional layers, followed by three fully connected layers, and employs techniques like ReLU activation, dropout, and data augmentation. AlexNet significantly reduced error rates at the time and laid the foundation for modern deep learning in computer vision**.**

Architecture:

Input:

- Accepts images of size **224x224x3** (RGB).

Feature Extraction (Convolutional and Pooling Layers):

- 5 convolutional layers: filters with ReLU activation.Followed by MaxPooling

Flatten and Dense Layers:

- Flatten: Converts the extracted features into a 1D vector.
- Dense Layer 1 & Dense Layer 2: 4096 units, with ReLU activation.Followed by Dropout (rate 0.5) to reduce overfitting.

Output Layer: A dense layer with num_classes units and softmax activation

Optimization:

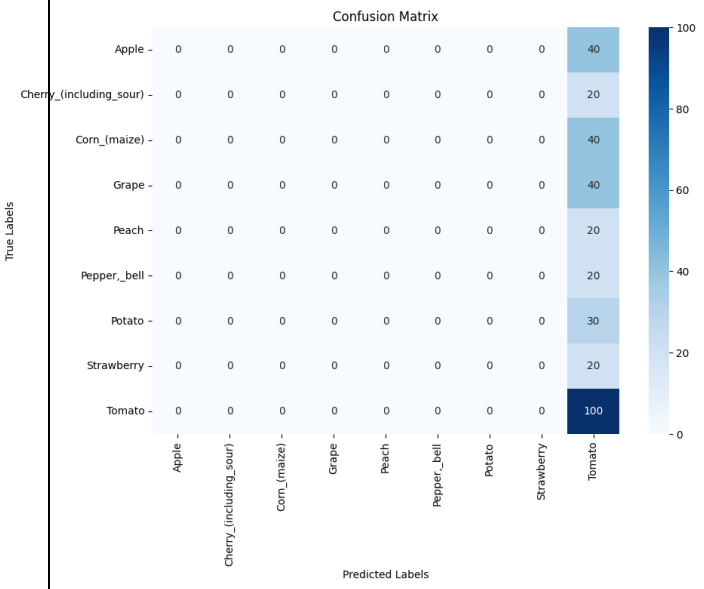
- Uses Adam optimizer, categorical cross-entropy loss, and accuracy as a performance metric.

Output: Produces class probabilities for classification tasks

Worst Model: Context Matters

- **Why:** While AlexNet was groundbreaking in 2012, its architecture is now considered outdated compared to more efficient and deeper models like VGG and MobileNet. It has fewer layers, lower accuracy, and lacks optimizations like depthwise separable convolutions.
- **Drawback**: Inefficiencies and limitations make it less competitive in scenarios where computational resources and accuracy are critical.

AlexNet: accuracy and **time** during training and testing



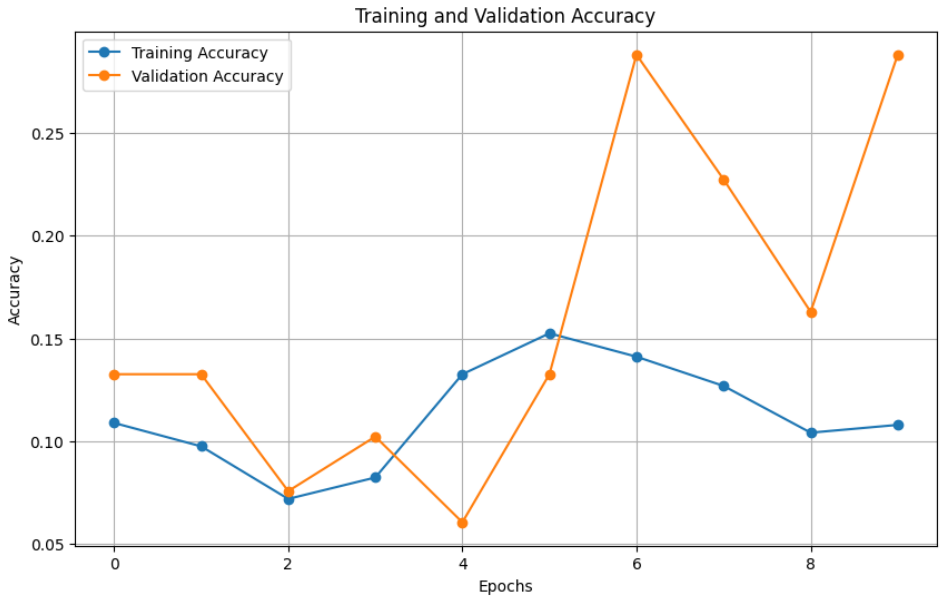

```
Epoch 1/10
17/17 ━━━━━━━━━━━━━━━━━━━━ 18s 500ms/step - accuracy: 0.1103 - loss: 5.4200 - val_accuracy: 0.1326 - val_loss: 2.1928
Epoch 2/10
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 395ms/step - accuracy: 0.1141 - loss: 2.2161 - val_accuracy: 0.1326 - val_loss: 2.1990
Epoch 3/10
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 378ms/step - accuracy: 0.0816 - loss: 2.2461 - val_accuracy: 0.0758 - val_loss: 2.1987
Epoch 4/10
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 395ms/step - accuracy: 0.0839 - loss: 2.1632 - val_accuracy: 0.1023 - val_loss: 2.1998
Epoch 5/10
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 382ms/step - accuracy: 0.1461 - loss: 2.1605 - val_accuracy: 0.0606 - val_loss: 2.1866
Epoch 6/10
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 392ms/step - accuracy: 0.1533 - loss: 2.1608 - val_accuracy: 0.1326 - val_loss: 2.1878
Epoch 7/10
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 375ms/step - accuracy: 0.1417 - loss: 2.2222 - val_accuracy: 0.2879 - val_loss: 2.1932
Epoch 8/10
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 401ms/step - accuracy: 0.1603 - loss: 2.2032 - val_accuracy: 0.2273 - val_loss: 2.1731
Epoch 9/10
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 375ms/step - accuracy: 0.1228 - loss: 2.2356 - val_accuracy: 0.1629 - val_loss: 2.1962
Epoch 10/10
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 373ms/step - accuracy: 0.1113 - loss: 2.1526 - val_accuracy: 0.2879 - val_loss: 2.1907
Training Time: 118.67 seconds
Training Accuracy: 0.11
```

```
Training Time: 118.67 seconds
Training Accuracy: 0.11
```

```
Testing Time: 0.79 seconds
Test Accuracy: 0.30, Test Loss: 2.19
```

Classification Report of AlexNet:

	precision	recall	f1-score	support
Apple	0.00	0.00	0.00	40
Cherry_(including_sour)	0.00	0.00	0.00	20
Corn_(maize)	0.00	0.00	0.00	40
Grape	0.00	0.00	0.00	40
Peach	0.00	0.00	0.00	20
Pepper,_bell	0.00	0.00	0.00	20
Potato	0.00	0.00	0.00	30
Strawberry	0.00	0.00	0.00	20
Tomato	0.30	1.00	0.47	100
accuracy			0.30	330
macro avg	0.03	0.11	0.05	330
weighted avg	0.09	0.30	0.14	330



Second: Plant Disease Recognition

Siamese Architecture: A neural network designed to determine the similarity or dissimilarity between two inputs.

Twin Networks: Consists of two identical sub-networks that share the same weights and parameters.

Shared Weights: Both sub-networks learn the same features from the input data, ensuring consistent comparisons.

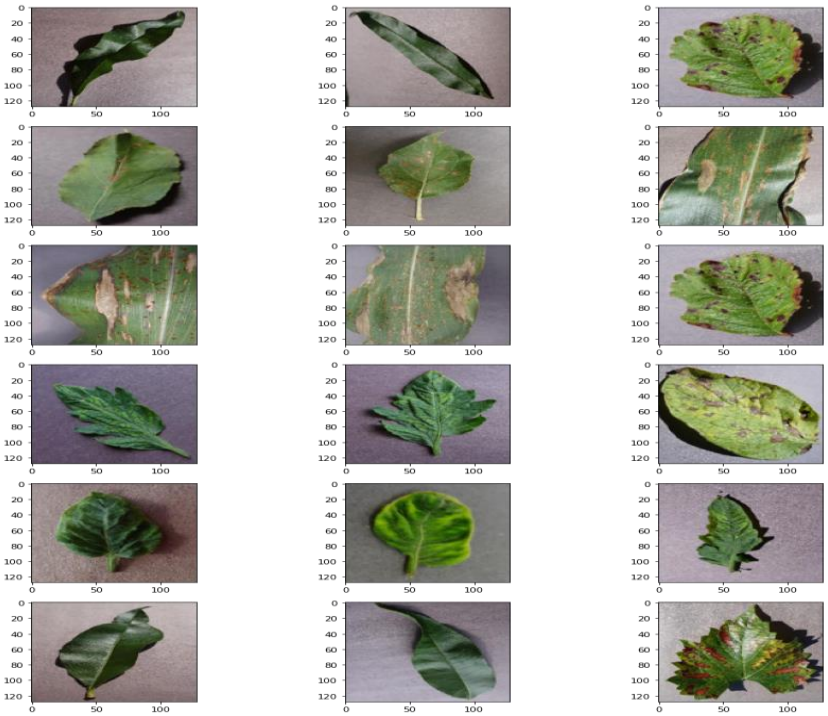
Distance Metric: Outputs (feature vectors) from the sub-networks are compared using a distance metric like Euclidean distance or cosine similarity.

Training: Network is trained with pairs of images labeled as similar or dissimilar, adjusting parameters to bring similar images closer and dissimilar ones farther apart.

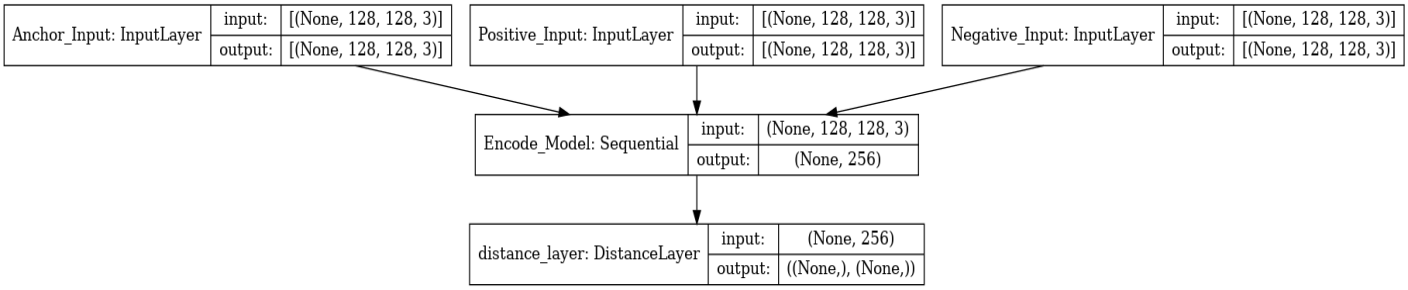
Application: Commonly used in tasks such as plant recognition or image matching where pairwise comparisons are necessary

Advantages of One-shot Learning in Plant Recognition:

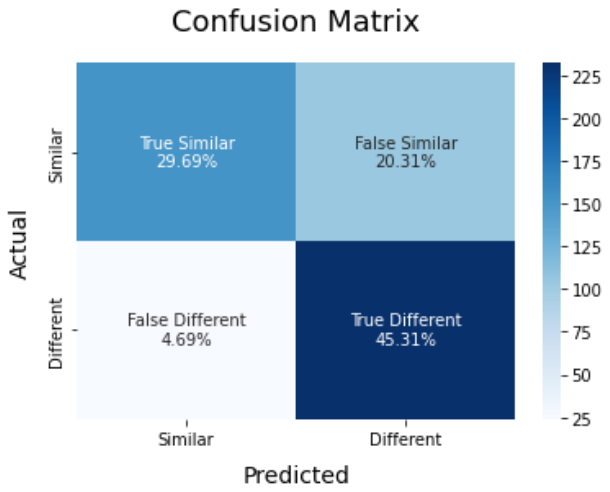
- **Reduced Data Requirements**: Recognizes plant species with just one image per species, reducing the need for large labeled datasets.
- **Generalization**: Effectively generalizes to new, unseen plant species, especially with models like Siamese or Prototypical Networks.



Siamese Architecture



Siamese Evaluation



Accuracy of model: 0.75

```
EPOCH: 1          (Epoch done in 242 sec)
Loss on train     = 0.64235
Accuracy on test = 0.76296

EPOCH: 2          (Epoch done in 222 sec)
Loss on train     = 0.43632
Accuracy on test = 0.77980

EPOCH: 3          (Epoch done in 222 sec)
Loss on train     = 0.29168
Accuracy on test = 0.83502

EPOCH: 4          (Epoch done in 222 sec)
Loss on train     = 0.18078
Accuracy on test = 0.80875

EPOCH: 5          (Epoch done in 221 sec)
Loss on train     = 0.12648
Accuracy on test = 0.81818

EPOCH: 6          (Epoch done in 222 sec)
Loss on train     = 0.08728
Accuracy on test = 0.82963
...

EPOCH: 15         (Epoch done in 222 sec)
Loss on train     = 0.01784
Accuracy on test = 0.82290
```

Third: Segmentation

Data Preparation :

Data Loading:

- Images and corresponding masks were loaded from separate directories.

```
Image shape: (1320, 256, 256)
Mask shape: (1320, 256, 256)
```

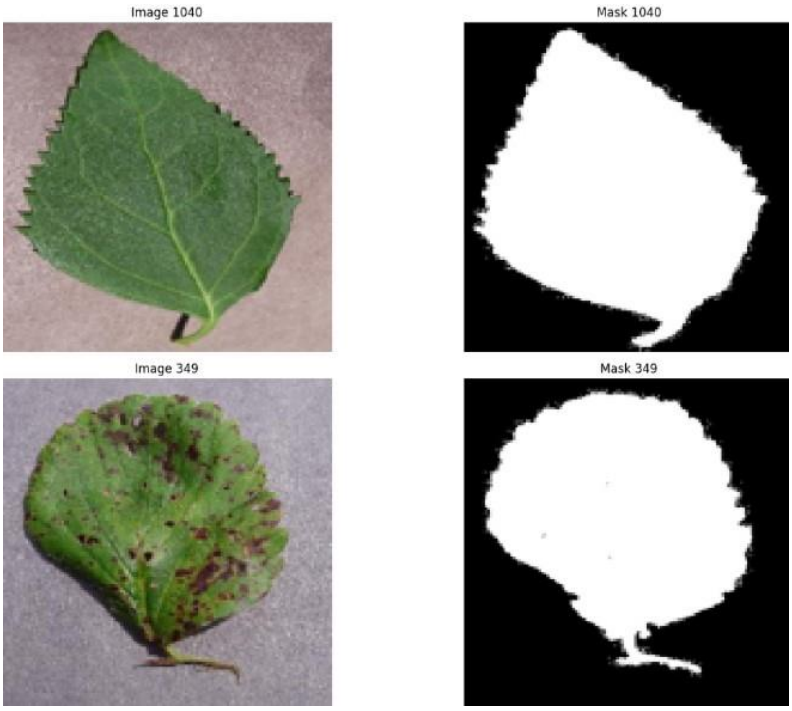
- Images and masks were **resized** to a predefined target dimension for consistency.

Data Preprocessing:

- Images were **normalized** by scaling pixel values to the range [0, 1].
- Masks were binarized, converting pixel values to 0 or 1 based on intensity.
- The **class weights** are computed to handle class imbalance. The compute_class_weight function calculates weights inversely proportional to class frequencies.
- The **LabelEncoder** is used to convert text labels into integer labels

Dataset Splitting:

- The dataset was split into training and validation sets, with 80% used for training and 20% for validation.
- A fixed random state was used to ensure reproducibility of the split.



U-Net Model

U-Net is a convolutional neural network architecture specifically designed for biomedical image segmentation. It has a symmetrical encoder-decoder structure, where the encoder extracts features, and the decoder reconstructs the image with segmentation masks. Skip connections link corresponding layers in the encoder and decoder to preserve spatial information. U-Net is highly efficient and performs well on small datasets, making it a popular choice in medical imaging tasks.

Architecture

Define U-Net Blocks:

- Implemented a **convolutional block** (conv_block) that includes two convolutional layers with ReLU activation, kernel initialization, and dropout for regularization.
- Created an **upsampling block** (upsample_block) using transposed convolution for upsampling and concatenation of features from previous layers.

Contracting Path:

- Used sequential convolutional blocks (conv_block) and max-pooling layers to reduce spatial dimensions while increasing the number of feature channels:
- **Encoder**: Extracts and compresses features from the input (downsampling).

Expanding Path:

- Applied upsampling blocks to reconstruct spatial dimensions and combine features from the contracting path:
- **Decoder**: Reconstructs the spatial dimensions and combines extracted features
- These stages are connected by the **bottleneck layer** (c5), which acts as the transition point between the encoder and decoder.

Output Layer:

- Added a final convolutional layer with 1 filter and sigmoid activation to produce a probability map for binary segmentation.
- **Model Training**:
- Defined callbacks for early stopping and saving the best model:
    - EarlyStopping monitored validation loss with a patience of 5 epochs.
    - ModelCheckpoint saved the best model during training.

Model Saving:

- Saved the trained model in HDF5 format (model.h5).

Model Evaluation:

Precision:0.9655

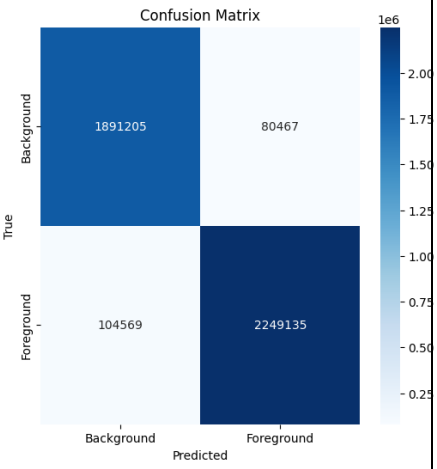
Recall: 0.9556

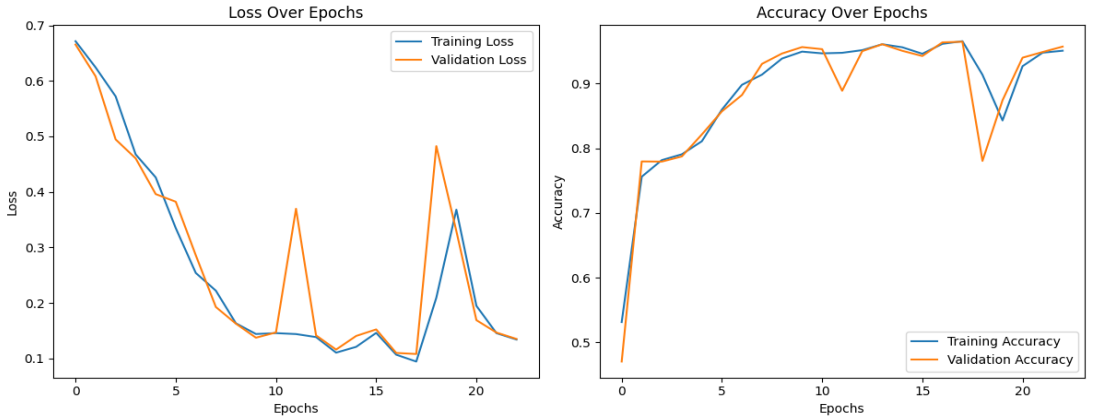
Accuracy: 0.9572

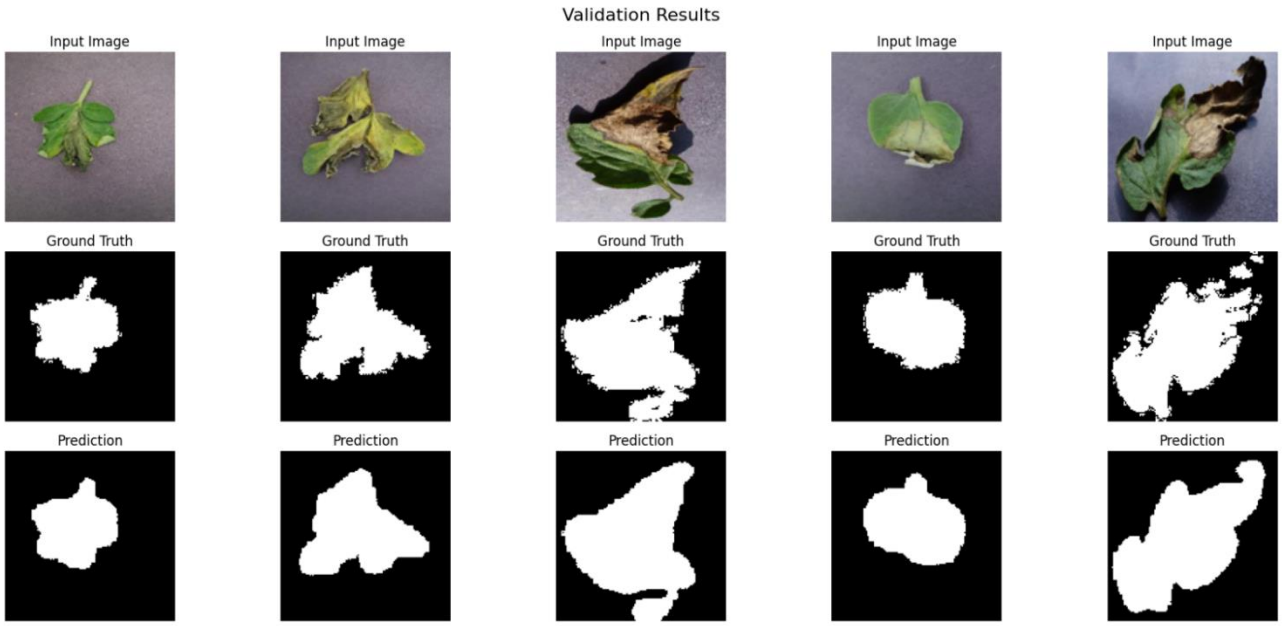
F1 Score: 0.9605

`

```
I0000 00:00:1734214580.581956    2223 service.cc:153]   StreamExecutor device (0): Tesla P100-PCIE-16GB, Compute Capability 6.0
 0/66 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.5083 - loss: 0.6929
I0000 00:00:1734214596.906854    2223 device_compiler.h:188] Compiled cluster using XLA!  This line is logged at most once for the lifetime of the process.
66/66 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.5554 - loss: 0.6702
Epoch 1: val_loss improved from inf to 0.66551, saving model to model.keras
66/66 ━━━━━━━━━━━━━━━━━━━━ 25s 60ms/step - accuracy: 0.5550 - loss: 0.6702 - val_accuracy: 0.4706 - val_loss: 0.6655
Epoch 2/25
64/66 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 0.7207 - loss: 0.6520
Epoch 2: val_loss improved from 0.66551 to 0.60828, saving model to model.keras
66/66 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step - accuracy: 0.7223 - loss: 0.6507 - val_accuracy: 0.7798 - val_loss: 0.6083
Epoch 3/25
64/66 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 0.7866 - loss: 0.5907
Epoch 3: val_loss improved from 0.60828 to 0.49459, saving model to model.keras
66/66 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step - accuracy: 0.7864 - loss: 0.5899 - val_accuracy: 0.7795 - val_loss: 0.4946
Epoch 4/25
64/66 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 0.7889 - loss: 0.4748
Epoch 4: val_loss improved from 0.49459 to 0.46041, saving model to model.keras
66/66 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step - accuracy: 0.7890 - loss: 0.4745 - val_accuracy: 0.7875 - val_loss: 0.4604
Epoch 5/25
64/66 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.7948 - loss: 0.4591
Epoch 5: val_loss improved from 0.46041 to 0.39600, saving model to model.keras
66/66 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step - accuracy: 0.7955 - loss: 0.4576 - val_accuracy: 0.8215 - val_loss: 0.3960
Epoch 6/25
64/66 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 0.8460 - loss: 0.3619
Epoch 6: val_loss improved from 0.39600 to 0.38228, saving model to model.keras
66/66 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step - accuracy: 0.8466 - loss: 0.3607 - val_accuracy: 0.8572 - val_loss: 0.3823
Epoch 7/25
64/66 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 0.8919 - loss: 0.2700
...
Epoch 23/25
64/66 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 0.9501 - loss: 0.1347
Epoch 23: val_loss did not improve from 0.10791
66/66 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.9502 - loss: 0.1347 - val_accuracy: 0.9572 - val_loss: 0.1350
```







SAM Model

SAM is based on a foundation of **transformer models**, leveraging the power of attention mechanisms to learn spatial relationships within images for precise segmentation. SAM uses a **vision transformer (ViT)** as its backbone. Vision transformers have self-attention mechanisms that allow the model to capture long-range dependencies between pixels.

Architecture:

The main parts of the SAM architecture include:

- **Backbone (Vision Transformer - ViT)**: This is the core architecture of SAM, where image features are extracted.
- **Prompt Encoder**: This component processes the different types of input prompts (points, boxes, and masks) to guide the segmentation.
- **Segmentation Decoder**: This part decodes the model's predictions into final segmentation masks.

Dice Loss Advantages:

- **Handling Imbalanced Data**: Dice Loss is particularly useful when the dataset is imbalanced.

Model Evaluation:

```
EPOCH: 0
Mean loss: 0.24324197495977085
Mean pixel-wise accuracy: 82.59%
Mean IoU: 5.0603
```