Achintya Agarwal **102115126** 4O14 TIET Deep Learning Project

VEG-FINDER_V1: Vegetable and Edible Goods Freshness Index for Neural Detection and Evaluation Recognition

1) Introduction:

The increasing demand for fresh and healthy food has heightened the need for accurate and efficient methods to monitor the freshness of fruits and vegetables throughout the supply chain. Traditional methods of assessing food quality, which often rely on subjective human evaluation, can lead to inconsistencies, errors, and time-consuming processes. These limitations are particularly evident in the food industry, where timely and precise evaluations are crucial to maintaining product quality and reducing waste. In recent years, artificial intelligence (AI) and computer vision technologies have emerged as transformative solutions to automate food quality assessment, offering a faster, more accurate, and scalable approach.

VEG-FINDER is an innovative deep learning-based system designed to address these challenges by using neural networks to classify and predict the freshness of fruits and vegetables. Leveraging image data, VEG-FINDER analyzes key visual features such as color, texture, and surface imperfections to provide objective and reliable freshness evaluations. This system not only enhances the accuracy of food quality assessments but also offers a scalable solution for the food industry, from farms to retailers. By automating the process of freshness evaluation, VEG-FINDER optimizes inventory management, reduces food waste, and improves the overall efficiency of the food supply chain, benefiting both industry stakeholders and consumers alike.

2) Problem Statement:

The primary challenge addressed by this project is the need for an automated, reliable, and efficient method for assessing the freshness of fruits and vegetables. Freshness is a critical factor in determining the quality of produce, but traditional methods of evaluation—such as manual inspection or the use of chemical indicators—are often costly, labor-intensive, and prone to errors. The goal is to develop a neural detection model that can automatically evaluate the freshness of produce from images, using deep learning algorithms that provide consistent, objective assessments. This system would assist in determining the shelf life of food items, thereby reducing waste and improving inventory management.

3) Novelty:

This study introduces an advanced deep learning approach to evaluating the freshness of fruits and vegetables based on visual data. What sets this project apart is its focus on the use of a "Freshness Index" that quantifies the level of freshness, offering a unique numerical output rather than a binary classification of fresh vs. rotten. Additionally, the system incorporates confidence scores, enabling a more nuanced prediction of freshness that enhances the reliability of the model.

4) Objective:

- Develop a deep learning model to classify the freshness of fruits and vegetables from images using a convolutional neural network (CNN).
- Create a Freshness Index that quantifies freshness on a scale from 0 to 100, where 100 represents perfectly fresh produce.
- Integrate a confidence scoring mechanism to improve the accuracy and reliability of the freshness predictions.

5) Model Architecture:

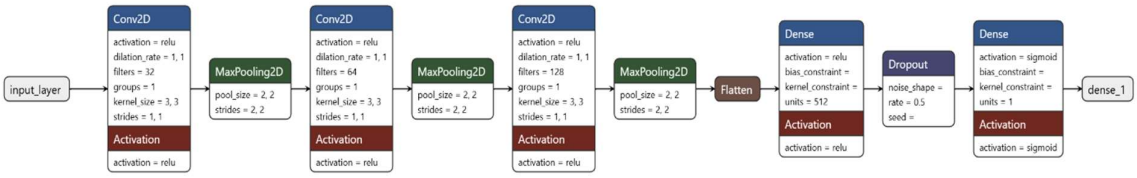


Fig1: Netron Visualization of Tensorflow Model

5.1.0 Model Layers:

5.1.1 Input Shape

The model expects input images of size 150x150 pixels with 3 color channels (RGB). This is the size of the input images the model will process.

5.1.2 First Convolutional Block

- **Conv2D (32 filters, kernel size = (3, 3), activation = 'relu')**

  This is a convolutional layer that applies 32 filters (kernels) of size 3x3 on the input image. The relu activation function is applied element-wise to introduce non-linearity.

  The convolution operation detects local patterns like edges, textures, or simple shapes in the image.

- **MaxPooling2D (pool size = (2, 2))**

A pooling layer that performs **max pooling** with a pool size of 2x2. It reduces the spatial dimensions (height and width) by taking the maximum value from a 2x2 area. This helps reduce the number of parameters and the computational load, and makes the model more invariant to small translations.

5.1.3 Second Convolutional Block

- **Conv2D (64 filters, kernel size = (3, 3), activation = 'relu')**

  Another convolutional layer with 64 filters of size 3x3, followed by the relu activation. This layer learns more complex features (combinations of edges and textures detected in the previous block).

- **MaxPooling2D (pool size = (2, 2))**

  A max pooling layer to down sample the spatial dimensions further. This reduces the computational complexity.

5.1.4. Third Convolutional Block

- **Conv2D (128 filters, kernel size = (3, 3), activation = 'relu')**

  This layer applies 128 filters of size 3x3, further increasing the complexity of the features the model can learn (detecting higher-level structures like parts of objects).

- **MaxPooling2D (pool size = (2, 2))**

  Another max pooling layer to reduce the spatial dimensions even more.

5.1.5 Fully Connected Layers (Dense Layers)

- **Flatten()**

  The Flatten layer converts the 3D output from the previous convolutional and pooling layers into a 1D vector. This is necessary because fully connected layers (Dense) expect 1D input.

- **Dense (512 units, activation = 'relu')**

  A fully connected layer with 512 units and relu activation. This layer learns complex combinations of features learned from the convolutional layers.

- **Dropout (rate = 0.5)**

  The dropout layer randomly sets 50% of the output of the previous dense layer to 0 during training. This is a form of regularization to prevent overfitting by making the model less reliant on specific neurons.

- **Dense (1 unit, activation = 'sigmoid')**

The final dense layer with 1 output unit. Since this is a **binary classification** problem (output is either class 0 or 1), the sigmoid activation function is used to output a probability between 0 and 1.

5.1.6 Summary of the Model Architecture**:**

1. **Conv2D + MaxPooling2D**: Extracts spatial features from the image by applying filters and down sampling.
2. **Flatten**: Converts 2D feature maps into a 1D vector for the fully connected layers.
3. **Dense + Dropout**: Learns higher-level features and applies regularization to prevent overfitting.
4. **Dense (final layer)**: Outputs a probability for binary classification.

This architecture is typical for image classification tasks, where the model learns hierarchical features of the image (from edges to complex structures) and makes a final classification based on these features.

5.2.1 Sigmoid Activation Function

- Formula: $\sigma(x)= \frac{1}{1+e^{-x}}$
- Range: 0 to 1.
- Usage: Sigmoid is typically used for binary classification problems (e.g., to model probabilities). It outputs values between 0 and 1, making it ideal for representing probabilities.
- Characteristics:
    - Non-linear: Helps the network learn non-linear relationships.
    - Output bounded: The output is always between 0 and 1, which can be interpreted as a probability.
    - Vanishing gradient problem: For very large or very small inputs, the gradients (used for backpropagation) become very small, slowing down the learning process.

5.2.2 ReLU (Rectified Linear Unit)

- Formula: $ReLU(x)=max(0,x)$
- Range: 0 to ∞.
- Usage: ReLU is one of the most widely used activation functions, especially in hidden layers of deep neural networks.
- Characteristics:
    - Non-linear: Provides non-linearity while being computationally simple.
    - Sparsity: ReLU outputs 0 for negative values, which means a large portion of the neurons can be inactive (sparse activation), improving model efficiency.
    - No vanishing gradient for positive inputs: For positive values, the gradient is constant, which helps with faster convergence.
    - Dying ReLU problem: ReLU can sometimes "die" during training, meaning some neurons never activate (when their inputs are always negative), resulting in no gradient and no updates for those neurons. Variants like Leaky ReLU and Parametric ReLU address this.

6. Dataset and Freshness Calculation:

The dataset consists of images of various fruits and vegetables, including bananas, apples, oranges, mangos, strawberries, potatoes, cucumbers, tomatoes, carrots, okra, and bell peppers. The images are categorized into two classes: fresh and rotten. This dataset was sourced from Kaggle and is used for binary classification, where the goal is to predict the freshness of the produce based on the image. The dataset serves as a foundation for training a neural network to classify produce based on visual cues.

After classifying an image as either "Fresh" or "Rotten" based on the model's **confidence score**, the freshness index is calculated. If the confidence score is greater than or equal to **0.5**, the produce is classified as "**Rotten**," and the freshness index is computed as $(1-\text{confidence score}) \times 100$, reflecting how much the product deviates from being fresh. Conversely, if the confidence score is less than **0.5**, the produce is classified as "**Fresh**," and the freshness index is calculated as $50 + (\text{confidence score} \times 100)$, indicating how fresh the produce is, with **100** being completely fresh. The resulting freshness index is rounded to four decimal places for accuracy.

7. Performance Metrics:

Loss Function:

- Binary Cross Entropy: This is the loss function used for binary classification tasks. It measures the difference between the true labels and the predicted probabilities, guiding the model's learning process.

  Binary Cross Entropy= $-\frac{1}{N}\sum_{i=1}^{N}[y_i log(p_i) + (1-y_i) log(1-p_i)]$

Accuracy: This metric measures the proportion of correct predictions (both fresh and rotten) made by the model. It is calculated as:

Accuracy= $\frac{True\ Positives + True\ Negatives}{Total\ Samples}$

This is the primary metric you will see during training and validation. It will give you an overall understanding of how well the model is performing.

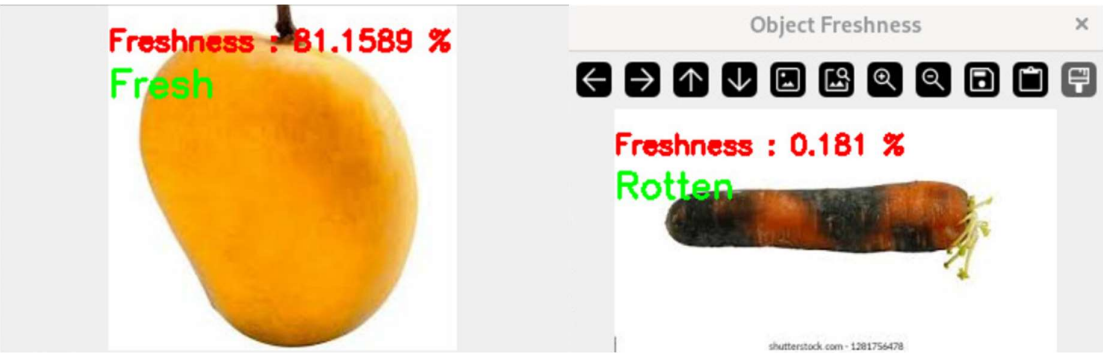


Fig3(Inference on Sample Images)

In **(fig3)** we can see the model has correctly classified the 2 fruits into fresh and rotten based on the freshness index

8. Results:

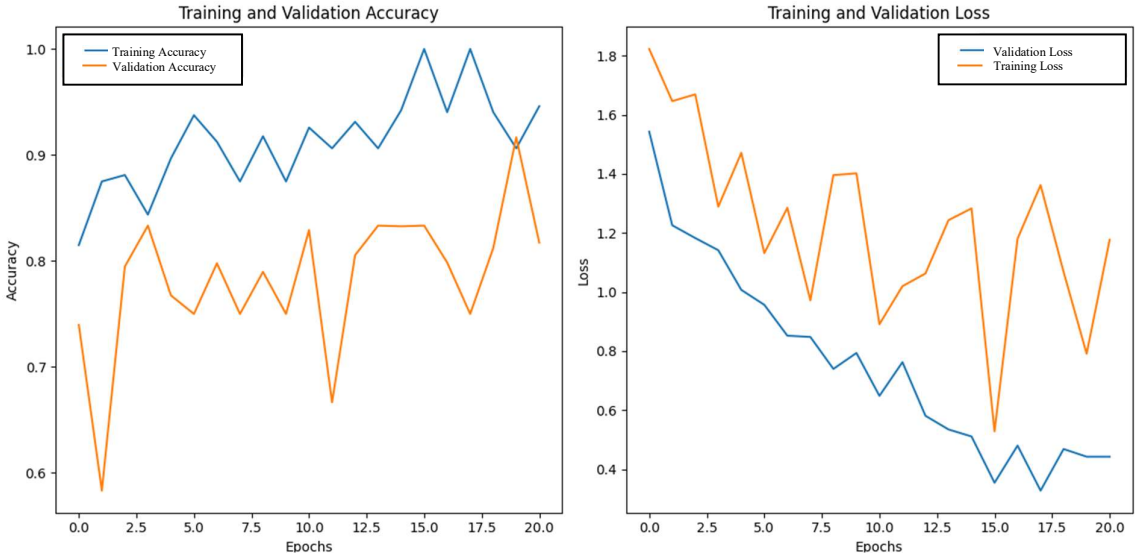


Fig2: Accuracy and Loss curve while training

In (**Fig2 Left**) Training Accuracy increases steadily and reaches above 90%, indicating that the model is effectively learning the training data. Validation Accuracy starts high but fluctuates heavily across epochs, maintaining a lower level than the Training Accuracy for most of the epochs. This suggests.

In (**Fig2 Right**) Training Loss shows a steady decrease, indicating the model is consistently learning and minimizing the error on the training set. Validation Loss fluctuates significantly and does not follow a clear downward trend, which aligns with the unstable validation accuracy. This might suggest that the model is not learning effectively on the validation data and could be overfitting

Training Loss	0.44
Validation Loss	1.18
Training Accuracy	94.60%
Validation Accuracy	81.73%

Despite the potential overfitting, the model performance is up to the mark when used in rea world scenarios and can be used to check the freshness index of produce.

9. Conclusion:

The VEG-FINDER_V1 project presents an innovative and efficient solution for automating the assessment of fruit and vegetable freshness using deep learning. By leveraging a custom convolutional neural network and a unique Freshness Index, VEG-FINDER enables precise and objective evaluations that support better quality control and waste reduction in the food supply chain. The model's high accuracy in classifying produce as fresh or rotten, along with its interpretative Freshness Index, offers a significant advancement over traditional, often subjective, methods. While some challenges remain, such as further reducing overfitting, the results demonstrate VEG-FINDER's potential to be a practical and scalable tool for retailers, suppliers, and consumers. Future developments could focus on expanding the model's applicability across different produce types and integrating it into real-world systems to help optimize shelf-life management, reduce food waste, and improve freshness for end consumers.

10. References:

1. N. Rahman, M. M. Arefin, S. Rahman, M. S. Islam, T. Khatun, and U. Akter, "Enhancing the Accuracy of Fruit Freshness Detection by Utilizing Transfer Learning and Customizing Convolutional Neural Network (CNN)," *IEEE Conference on Multimedia Information Processing and Retrieval*, Apr. 2024, doi: 10.1109/icmi60790.2024.10585689.
2. N. Ramakrishna, "Fruit Freshness Evaluation using a Real-Time Industrial Framework for Deep Learning Ensemble Approaches," *International Journal for Science, Technology and Engineering*, vol. 11, no. 4, pp. 541–550, Jul. 2023, doi: 10.22214/ijraset.2023.54651.
3. R. Ahmed, M. M. Haque, S. Saha, C. Saha, M. Dutta, D. Z. Karim, M. Mostakim, and A. A. Rasel, "Fresh or Stale: Leveraging Deep Learning to Detect Freshness of Fruits and Vegetables," *IEEE International Conference on Electronics, Computing and Communication Technologies*, Jul. 2023, doi: 10.1109/iceccme57830.2023.10252295.
4. S. Shah, "Fruit Recognition and Freshness Detection Using Convolutional Neural Networks," in *Advances in Intelligent Systems and Computing*, Singapore: Springer, 2023, pp. 587–600, doi: 10.1007/978-981-99-0047-3_43.
5. M. A. Hamim, J. Tahseen, K. M. I. Hossain, N. Akter, and U. F. T. Asha, "Bangladeshi Fresh-Rotten Fruit & Vegetable Detection Using Deep Learning Deployment in Effective Application," *IEEE Conference on Computational Intelligence*, May 2023, doi: 10.1109/ccai57533.2023.10201244.

11. Code:

In [26]:
```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.regularizers import l2
```

In [17]:
```
# Directory for the training dataset (which has two subfolders: Fresh and Rotten)
train_dir = '/content/extracted_folder/fruits_vegetables_dataset'
```

In [19]:
```
# ImageDataGenerator with refined augmentations and a 70-30 split
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=10,     # Reduced rotation
    width_shift_range=0.1, # Reduced shift
    height_shift_range=0.1,
    zoom_range=0.1,        # Reduced zoom
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.3
)
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary',
    subset='training'
)
```

```
Found 19892 images belonging to 2 classes.
```

In [20]:
```
test_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary',
    subset='validation'
)
```

```
Found 8524 images belonging to 2 classes.
```

In [27]:
```
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    BatchNormalization(),
    MaxPooling2D(2, 2),

    Conv2D(64, (3, 3), activation='relu', kernel_regularizer=l2(0.001)),
    BatchNormalization(),
    MaxPooling2D(2, 2),

    Conv2D(128, (3, 3), activation='relu', kernel_regularizer=l2(0.001)),
    BatchNormalization(),
    MaxPooling2D(2, 2),

    Flatten(),
    Dense(512, activation='relu', kernel_regularizer=l2(0.001)),
    Dropout(0.3),  # Reduced dropout rate
    Dense(1, activation='sigmoid')  # Binary output
])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a lay
er. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [28]:
```
# Compile the model with a lower learning rate
model.compile(optimizer=Adam(learning_rate=0.0001),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

In [29]:
```
model.summary()
```

```
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
conv2d_9 (Conv2D)	(None, 148, 148, 32)	896
batch_normalization_3 (BatchNormalization)	(None, 148, 148, 32)	128
max_pooling2d_9 (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_10 (Conv2D)	(None, 72, 72, 64)	18,496
batch_normalization_4 (BatchNormalization)	(None, 72, 72, 64)	256
max_pooling2d_10 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_11 (Conv2D)	(None, 34, 34, 128)	73,856
batch_normalization_5 (BatchNormalization)	(None, 34, 34, 128)	512
max_pooling2d_11 (MaxPooling2D)	(None, 17, 17, 128)	0
flatten_3 (Flatten)	(None, 36992)	0
dense_6 (Dense)	(None, 512)	18,940,416
dropout_3 (Dropout)	(None, 512)	0
dense_7 (Dense)	(None, 1)	513

Total params: 19,035,073 (72.61 MB)
Trainable params: 19,034,625 (72.61 MB)
Non-trainable params: 448 (1.75 KB)

In [30]:
```
# Early stopping to avoid overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    validation_data=test_generator,
    validation_steps=test_generator.samples // test_generator.batch_size,
    epochs=30,  # Increased number of epochs
    callbacks=[early_stopping]
)
```

```
Epoch 1/30
621/621 ━━━━━━━━━━━━━━━━━━━━ 215s 335ms/step - accuracy: 0.7574 - loss: 1.8527 - val_accuracy: 0.7397 - val_loss: 1.8225
Epoch 2/30
621/621 ━━━━━━━━━━━━━━━━━━━━ 1s 825us/step - accuracy: 0.8750 - loss: 1.2257 - val_accuracy: 0.5833 - val_loss: 1.6460
Epoch 3/30
621/621 ━━━━━━━━━━━━━━━━━━━━ 205s 327ms/step - accuracy: 0.8768 - loss: 1.2277 - val_accuracy: 0.7948 - val_loss: 1.6689
Epoch 4/30
621/621 ━━━━━━━━━━━━━━━━━━━━ 0s 184us/step - accuracy: 0.8438 - loss: 1.1413 - val_accuracy: 0.8333 - val_loss: 1.2887
Epoch 5/30
621/621 ━━━━━━━━━━━━━━━━━━━━ 204s 327ms/step - accuracy: 0.8873 - loss: 1.0564 - val_accuracy: 0.7675 - val_loss: 1.4710
Epoch 6/30
621/621 ━━━━━━━━━━━━━━━━━━━━ 0s 199us/step - accuracy: 0.9375 - loss: 0.9565 - val_accuracy: 0.7500 - val_loss: 1.1316
Epoch 7/30
621/621 ━━━━━━━━━━━━━━━━━━━━ 205s 328ms/step - accuracy: 0.9082 - loss: 0.8839 - val_accuracy: 0.7978 - val_loss: 1.2849
Epoch 8/30
621/621 ━━━━━━━━━━━━━━━━━━━━ 0s 211us/step - accuracy: 0.8750 - loss: 0.8478 - val_accuracy: 0.7500 - val_loss: 0.9715

Epoch 9/30
621/621 ━━━━━━━━━━━━━━━━━━━━ 205s 328ms/step - accuracy: 0.9187 - loss: 0.7588 - val_accuracy: 0.7897 - val_loss: 1.3957
Epoch 10/30
621/621 ━━━━━━━━━━━━━━━━━━━━ 0s 117us/step - accuracy: 0.8750 - loss: 0.7935 - val_accuracy: 0.7500 - val_loss: 1.4015
Epoch 11/30
621/621 ━━━━━━━━━━━━━━━━━━━━ 205s 327ms/step - accuracy: 0.9191 - loss: 0.6749 - val_accuracy: 0.8292 - val_loss: 0.8909
Epoch 12/30
621/621 ━━━━━━━━━━━━━━━━━━━━ 0s 40us/step - accuracy: 0.9062 - loss: 0.7624 - val_accuracy: 0.6667 - val_loss: 1.0196
Epoch 13/30
621/621 ━━━━━━━━━━━━━━━━━━━━ 205s 327ms/step - accuracy: 0.9325 - loss: 0.5924 - val_accuracy: 0.8055 - val_loss: 1.0626
Epoch 14/30
621/621 ━━━━━━━━━━━━━━━━━━━━ 0s 125us/step - accuracy: 0.9062 - loss: 0.5347 - val_accuracy: 0.8333 - val_loss: 1.2431
Epoch 15/30
621/621 ━━━━━━━━━━━━━━━━━━━━ 205s 327ms/step - accuracy: 0.9414 - loss: 0.5254 - val_accuracy: 0.8327 - val_loss: 1.2828
Epoch 16/30
621/621 ━━━━━━━━━━━━━━━━━━━━ 0s 192us/step - accuracy: 1.0000 - loss: 0.3543 - val_accuracy: 0.8333 - val_loss: 0.5282
Epoch 17/30
621/621 ━━━━━━━━━━━━━━━━━━━━ 205s 327ms/step - accuracy: 0.9395 - loss: 0.4829 - val_accuracy: 0.7985 - val_loss: 1.1804
Epoch 18/30
621/621 ━━━━━━━━━━━━━━━━━━━━ 0s 105us/step - accuracy: 1.0000 - loss: 0.3277 - val_accuracy: 0.7500 - val_loss: 1.3618
Epoch 19/30
621/621 ━━━━━━━━━━━━━━━━━━━━ 204s 327ms/step - accuracy: 0.9405 - loss: 0.4734 - val_accuracy: 0.8120 - val_loss: 1.0679
Epoch 20/30
621/621 ━━━━━━━━━━━━━━━━━━━━ 0s 107us/step - accuracy: 0.9062 - loss: 0.4425 - val_accuracy: 0.9167 - val_loss: 0.7909
Epoch 21/30
621/621 ━━━━━━━━━━━━━━━━━━━━ 205s 328ms/step - accuracy: 0.9449 - loss: 0.4474 - val_accuracy: 0.8173 - val_loss: 1.1766
```

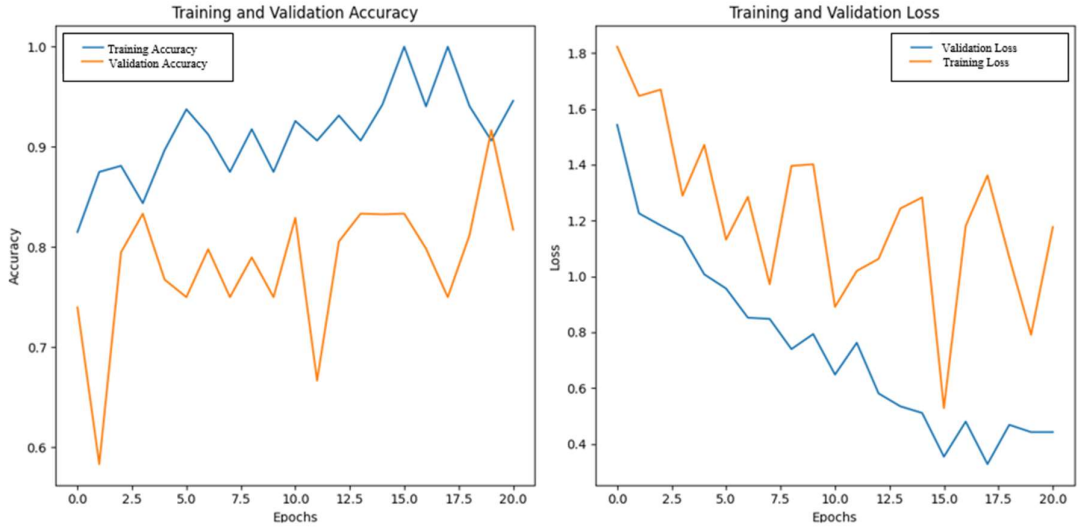
In [36]:
```
import matplotlib.pyplot as plt

# Plot training & validation accuracy
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plot training & validation loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Show plots
plt.tight_layout()
plt.show()

# Print the final loss and accuracy
final_loss = history.history['loss'][-1]
final_val_loss = history.history['val_loss'][-1]
final_accuracy = history.history['accuracy'][-1]
final_val_accuracy = history.history['val_accuracy'][-1]
```




```
Final Training Loss: 0.44
Final Validation Loss: 1.18
Final Training Accuracy: 94.60%
Final Validation Accuracy: 81.73%
```

In [37]:
```
# Save the model
model.save('fruit_veg_classifier.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recomme
nd using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
```

GitHub Link for code :-https://github.com/Achintya019/VEG-FINDER_V1-Vegetable-and-Edible-Goods-Freshness-Index-for-Neural-Detection-and-Evaluation-Recogn