

## WorkSheet 6 OUTPUT:

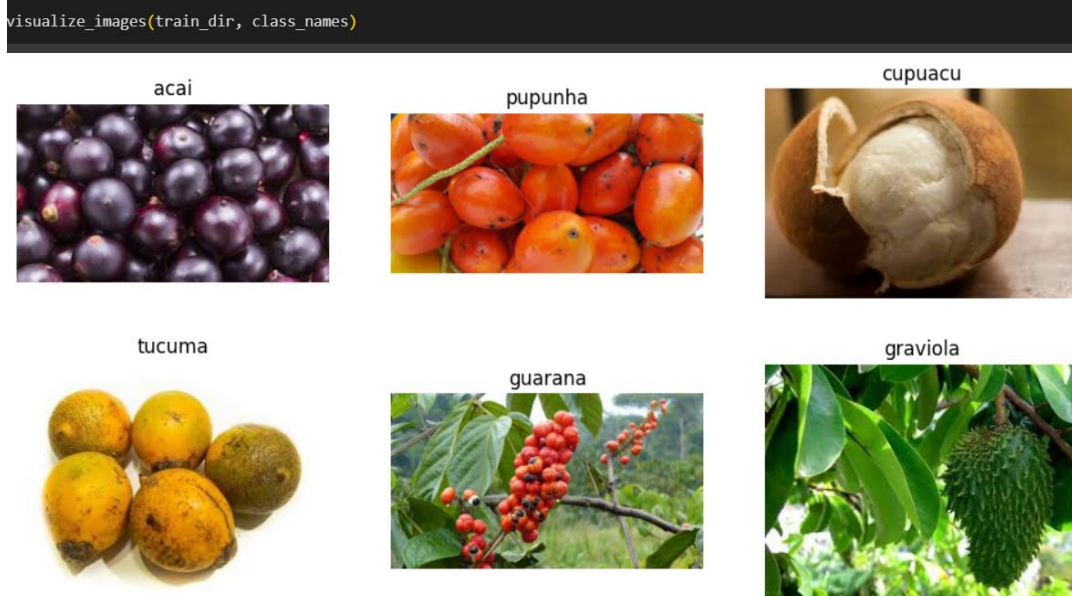
```
train_dir = "/content/drive/MyDrive/Artificial intelligence and Machine learning/week-6/FruitinAmazon/train"
test_dir = "/content/drive/MyDrive/Artificial intelligence and Machine learning/week-6/FruitinAmazon/test"
```

```
class_names = os.listdir(train_dir)
print(f"Classes: {class_names}")
```

```
Classes: ['acai', 'pupunha', 'cupuacu', 'tucuma', 'guarana', 'graviola']
```

```
def visualize_images(train_dir, class_names):
    fig, axes = plt.subplots(2, len(class_names) // 2, figsize=(12, 6))
    axes = axes.flatten()
    for i, class_name in enumerate(class_names):
        class_path = os.path.join(train_dir, class_name)
        img_name = random.choice(os.listdir(class_path))
        img_path = os.path.join(class_path, img_name)
        img = load_img(img_path)
        axes[i].imshow(img)
        axes[i].set_title(class_name)
        axes[i].axis("off")
    plt.show()
```

```
visualize_images(train_dir, class_names)
```



```

damagedImages = []
for class_name in class_names:
    class_path = os.path.join(train_dir, class_name)
    for img_name in os.listdir(class_path):
        img_path = os.path.join(class_path, img_name)
        try:
            img = load_img(img_path) # Try opening the image
        except (IOError, SyntaxError):
            damagedImages.append(img_path)
            os.remove(img_path)
            print(f"Damaged image removed: {img_path}")

if not damagedImages:
    print("No Damaged Images Found.")

```

No Damaged Images Found.

```

train_ds = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='sparse',
    subset='training',
    shuffle=True,
    seed=123
)

```

Found 72 images belonging to 6 classes.

```

val_ds = val_datagen.flow_from_directory(
    train_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='sparse',
    subset='validation',
    shuffle=False,
    seed=123
)

```

Found 18 images belonging to 6 classes.

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 128, 128, 32)	896
batch_normalization (BatchNormalization)	(None, 128, 128, 32)	128
conv2d_1 (Conv2D)	(None, 128, 128, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 128, 128, 64)	256
max_pooling2d (MaxPooling2D)	(None, 64, 64, 64)	0
dropout (Dropout)	(None, 64, 64, 64)	0
conv2d_2 (Conv2D)	(None, 64, 64, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 64, 64, 128)	512
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 128)	0
dropout_1 (Dropout)	(None, 32, 32, 128)	0
flatten (Flatten)	(None, 131072)	0
dense (Dense)	(None, 256)	33,554,688
batch_normalization_3 (BatchNormalization)	(None, 256)	1,024
dropout_2 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 6)	1,542

dense_1 (Dense)	(None, 6)	1,542
-----------------	-----------	-------

Total params: 33,651,398 (128.37 MB)  
Trainable params: 33,650,438 (128.37 MB)  
Non-trainable params: 960 (3.75 KB)

```
Epoch 1/30
3/3 — 0s 7s/step - accuracy: 0.1324 - loss: 4.1515WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.sav
Epoch 2/30
3/3 — 0s 6s/step - accuracy: 0.3944 - loss: 2.7233WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.sav
3/3 — 20s 8s/step - accuracy: 0.4104 - loss: 2.7072 - val_accuracy: 0.3333 - val_loss: 2.9193 - learning_rate: 0.0010
Epoch 3/30
3/3 — 16s 7s/step - accuracy: 0.5132 - loss: 2.6731 - val_accuracy: 0.2222 - val_loss: 3.2767 - learning_rate: 0.0010
Epoch 4/30
3/3 — 16s 5s/step - accuracy: 0.4931 - loss: 2.6378 - val_accuracy: 0.2778 - val_loss: 4.7318 - learning_rate: 0.0010
Epoch 5/30
3/3 — 16s 7s/step - accuracy: 0.5465 - loss: 2.7442 - val_accuracy: 0.1667 - val_loss: 7.1288 - learning_rate: 0.0010
Epoch 6/30
3/3 — 16s 5s/step - accuracy: 0.5993 - loss: 2.6190 - val_accuracy: 0.1667 - val_loss: 9.5762 - learning_rate: 5.0000e-04
Epoch 7/30
3/3 — 16s 4s/step - accuracy: 0.5451 - loss: 2.5632 - val_accuracy: 0.1667 - val_loss: 11.0435 - learning_rate: 5.0000e-04
```

```
test_datagen = ImageDataGenerator(rescale=1./255)
```

```

test_datagen = ImageDataGenerator(rescale=1./255)
test_ds = test_datagen.flow_from_directory(
    test_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='sparse',
    shuffle=False
)
test_loss, test_accuracy = model.evaluate(test_ds)
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")

Found 30 images belonging to 6 classes.
1/1 ----- 2s 2s/step - accuracy: 0.3333 - loss: 2.8726
Test Accuracy: 33.33%

model.save("final_model.h5")
loaded_model = tf.keras.models.load_model("final_model.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 recommend using instead the SavedModel format via `model.save(filepath, save_format='tf')`.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.

```

```

y_true = test_ds.classes
y_pred = np.argmax(loaded_model.predict(test_ds), axis=1)

print(classification_report(y_true, y_pred, target_names=class_names))

```

```

1/1 ----- 1s 1s/step
precision recall f1-score support
acai 0.00 0.00 0.00 5
pupunha 0.25 1.00 0.40 5
cupuacu 0.00 0.00 0.00 5
tucuma 0.00 0.00 0.00 5
guarana 0.62 1.00 0.77 5
graviola 0.00 0.00 0.00 5

```

```

print(classification_report(y_true, y_pred, target_names=class_names))

```

```

1/1 ----- 1s 1s/step
precision recall f1-score support
acai 0.00 0.00 0.00 5
pupunha 0.25 1.00 0.40 5
cupuacu 0.00 0.00 0.00 5
tucuma 0.00 0.00 0.00 5
guarana 0.62 1.00 0.77 5
graviola 0.00 0.00 0.00 5
accuracy 0.33 0.33 0.33 30
macro avg 0.15 0.33 0.10 30
weighted avg 0.15 0.33 0.19 30

```

```

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples
warn_prf(average, modifier, f'{metric.capitalize()} is', len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples
warn_prf(average, modifier, f'{metric.capitalize()} is', len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples
warn_prf(average, modifier, f'{metric.capitalize()} is', len(result))

```

```

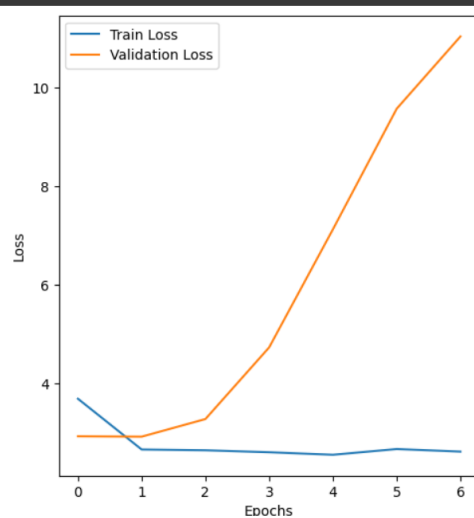
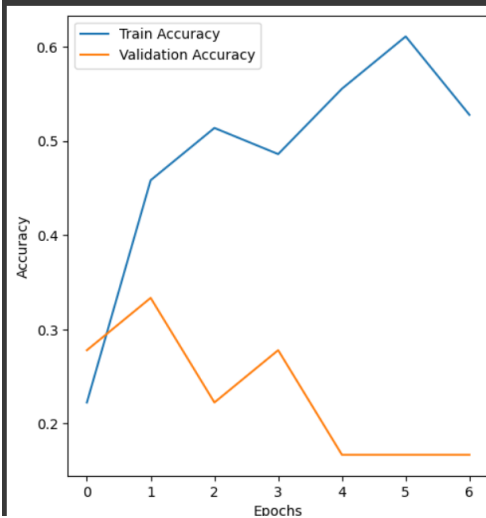
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)

```

```

plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

```



Downloading data from [https://storage.googleapis.com/tensorflow/keras-applications/mobilenet\\_v2/mobilenet\\_v2\\_weights\\_tf\\_dim\\_ordering\\_tf\\_kernels\\_1.0\\_128\\_no\\_top.h5](https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_128_no_top.h5)  
9406464/9406464 — 0s 0us/step

```
inputs = Input(shape=(img_height, img_width, 3))
x = base_model(inputs, training=False)
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.4)(x)
outputs = Dense(num_classes, activation='softmax')(x)
model = Model(inputs, outputs)
```

model.summary()

Model: "functional\_16"

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 128, 128, 3)	0
mobilenetv2_1.0_128 (Functional)	(None, 8, 8, 1280)	2,257,984
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0
dense_2 (Dense)	(None, 128)	163,968
dropout_3 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 4)	172

(Functional)	(None, 4)	172
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0
dense_2 (Dense)	(None, 128)	163,968
dropout_3 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 4)	172

Total params: 2,422,126 (9.24 MB)  
Trainable params: 164,140 (643.52 KB)  
Non-trainable params: 2,257,984 (8.61 MB)

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
callbacks = [
    ModelCheckpoint("best_model_t1.h5", save_best_only=True, monitor="val_accuracy", mode="max"),
    EarlyStopping(monitor="val_loss", patience=5, restore_best_weights=True),
    ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=3, min_lr=1e-6)
]
```

```
Epoch 1/30
3/3 — 0s 648ms/step - accuracy: 0.2657 - loss: 2.3201WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. 1
3/3 — 12s 2s/step - accuracy: 0.2549 - loss: 2.3198 - val_accuracy: 0.2222 - val_loss: 1.9954 - learning_rate: 0.0010
Epoch 2/30
3/3 — 0s 666ms/step - accuracy: 0.4155 - loss: 1.6917WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. 1
3/3 — 6s 2s/step - accuracy: 0.4123 - loss: 1.7130 - val_accuracy: 0.4444 - val_loss: 1.4282 - learning_rate: 0.0010
Epoch 3/30
3/3 — 0s 378ms/step - accuracy: 0.3944 - loss: 1.4936WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. 1
3/3 — 3s 686ms/step - accuracy: 0.4104 - loss: 1.4563 - val_accuracy: 0.5556 - val_loss: 1.2327 - learning_rate: 0.0010
Epoch 4/30
3/3 — 2s 408ms/step - accuracy: 0.5286 - loss: 0.9507 - val_accuracy: 0.5000 - val_loss: 1.1580 - learning_rate: 0.0010
Epoch 5/30
3/3 — 0s 568ms/step - accuracy: 0.7241 - loss: 0.7472WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. 1
3/3 — 2s 841ms/step - accuracy: 0.7236 - loss: 0.7553 - val_accuracy: 0.7222 - val_loss: 0.9767 - learning_rate: 0.0010
Epoch 6/30
3/3 — 2s 770ms/step - accuracy: 0.8531 - loss: 0.5173 - val_accuracy: 0.7222 - val_loss: 0.8294 - learning_rate: 0.0010
Epoch 7/30
3/3 — 3s 898ms/step - accuracy: 0.8345 - loss: 0.5103 - val_accuracy: 0.7222 - val_loss: 0.6791 - learning_rate: 0.0010
Epoch 8/30
3/3 — 0s 761ms/step - accuracy: 0.8657 - loss: 0.4015WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. 1
3/3 — 4s 1s/step - accuracy: 0.8611 - loss: 0.3997 - val_accuracy: 0.8333 - val_loss: 0.5864 - learning_rate: 0.0010
Epoch 9/30
3/3 — 2s 717ms/step - accuracy: 0.8111 - loss: 0.4936 - val_accuracy: 0.7778 - val_loss: 0.5607 - learning_rate: 0.0010
Epoch 10/30
3/3 — 2s 1s/step - accuracy: 0.9458 - loss: 0.2534 - val_accuracy: 0.7778 - val_loss: 0.5827 - learning_rate: 0.0010
Epoch 11/30
3/3 — 3s 937ms/step - accuracy: 0.9247 - loss: 0.2861 - val_accuracy: 0.8333 - val_loss: 0.6355 - learning_rate: 0.0010
Epoch 12/30
3/3 — 2s 733ms/step - accuracy: 0.9333 - loss: 0.2065 - val_accuracy: 0.8333 - val_loss: 0.6742 - learning_rate: 0.0010
Epoch 13/30
3/3 — 2s 846ms/step - accuracy: 0.9582 - loss: 0.1814 - val_accuracy: 0.8333 - val_loss: 0.6596 - learning_rate: 5.0000e-04
Epoch 14/30
```

```

test_loss, test_accuracy = model.evaluate(test_ds)
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")

1/1 ----- 1s 673ms/step - accuracy: 0.9667 - loss: 0.3166
Test Accuracy: 96.67%

model.save("final_model_t1.h5")

loaded_model = tf.keras.models.load_model("final_model_t1.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 recommend using instead the nat
WARNING:absl:compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.

y_true = test_ds.classes
y_pred_probs = loaded_model.predict(test_ds)
y_pred = np.argmax(y_pred_probs, axis=1)

print("Inference Output: First 20 Samples:")
for i in range(20):
    true_label = class_names[int(y_true[i])]
    pred_label = class_names[int(y_pred[i])]

```

```

1/1 ----- 2s 2s/step
Inference Output: First 20 Samples:
1. True: acai - Predicted: acai
2. True: acai - Predicted: acai
3. True: acai - Predicted: acai
4. True: acai - Predicted: acai
5. True: acai - Predicted: acai
6. True: pupunha - Predicted: pupunha
7. True: pupunha - Predicted: pupunha
8. True: pupunha - Predicted: pupunha
9. True: pupunha - Predicted: pupunha
10. True: pupunha - Predicted: pupunha
11. True: cupuacu - Predicted: cupuacu
12. True: cupuacu - Predicted: cupuacu
13. True: cupuacu - Predicted: cupuacu
14. True: cupuacu - Predicted: cupuacu
15. True: cupuacu - Predicted: cupuacu
16. True: tucuma - Predicted: tucuma
17. True: tucuma - Predicted: tucuma
18. True: tucuma - Predicted: tucuma
19. True: tucuma - Predicted: tucuma
20. True: tucuma - Predicted: tucuma

```

```

print(classification_report(y_true, y_pred, target_names=class_names))

```

```

Classification Report:

```

	precision	recall	f1-score	support
acai	1.00	1.00	1.00	5
pupunha	1.00	1.00	1.00	5
cupuacu	1.00	1.00	1.00	5
tucuma	0.83	1.00	0.91	5
guarana	1.00	1.00	1.00	5
graviola	1.00	0.80	0.89	5
accuracy			0.97	30
macro avg	0.97	0.97	0.97	30
weighted avg	0.97	0.97	0.97	30

```
plt.legend()  
plt.show()
```

