Task 1: Data Understanding and Visualization:

```
!pip install gdown

!gdown 12dBHB_f5jCOcalsV8Cp38u54tgkwcVT8

!unzip "FruitinAmazon.zip"
```

```
  inflating: FruitinAmazon/train/guarana/download (6).jpeg
  inflating: FruitinAmazon/train/pupunha/images (2).jpeg
  inflating: FruitinAmazon/test/guarana/download (5).jpeg
  inflating: FruitinAmazon/train/graviola/download (6).jpeg
  inflating: FruitinAmazon/train/graviola/images (3).jpeg
  inflating: FruitinAmazon/train/graviola/images (10).jpeg
  inflating: FruitinAmazon/train/pupunha/images (12).jpeg
  inflating: FruitinAmazon/train/graviola/images (8).jpeg
  inflating: FruitinAmazon/train/guarana/download (10).jpeg
  inflating: FruitinAmazon/train/tucuma/images (2).jpeg
  inflating: FruitinAmazon/train/tucuma/images (1).jpeg
  inflating: FruitinAmazon/train/guarana/images (6).jpeg
  inflating: FruitinAmazon/train/guarana/images (5).jpeg
  inflating: FruitinAmazon/train/guarana/download (1).jpeg
  inflating: FruitinAmazon/train/guarana/images (2).jpeg
  inflating: FruitinAmazon/train/tucuma/download (9).jpeg
  inflating: FruitinAmazon/train/graviola/download (8).jpeg
  inflating: FruitinAmazon/train/pupunha/images (11).jpeg
  inflating: FruitinAmazon/train/acai/images (6).jpeg
  inflating: FruitinAmazon/test/guarana/images (4).jpeg
  inflating: FruitinAmazon/train/graviola/download (5).jpeg
  inflating: FruitinAmazon/train/tucuma/images.jpeg
  inflating: FruitinAmazon/train/tucuma/images (9).jpeg
  inflating: FruitinAmazon/train/graviola/download.jpeg
  inflating: FruitinAmazon/train/graviola/images (5).jpeg
  inflating: FruitinAmazon/train/graviola/images (2).jpeg
  inflating: FruitinAmazon/train/guarana/download (9).jpeg
  inflating: FruitinAmazon/train/acai/images (12).jpeg
  inflating: FruitinAmazon/train/tucuma/images (3).jpeg
  inflating: FruitinAmazon/train/tucuma/images (7).jpeg
  inflating: FruitinAmazon/train/graviola/images.jpeg
  inflating: FruitinAmazon/train/tucuma/images (5).jpeg
  inflating: FruitinAmazon/train/tucuma/download (3).jpeg
  inflating: FruitinAmazon/train/cupuacu/images (4).jpeg
  inflating: FruitinAmazon/train/cupuacu/images (7).jpeg
  inflating: FruitinAmazon/train/cupuacu/download.jpeg
  inflating: FruitinAmazon/test/acai/images (17).jpeg
  inflating: FruitinAmazon/train/graviola/images (1).jpeg
  inflating: FruitinAmazon/train/cupuacu/images (13).jpeg
```

```
[ ] import tensorflow as tf
    from tensorflow import keras
    from tensorflow.keras import layers
    import numpy as np
    from PIL import Image
    import os
    import glob
    import numpy as np
    import matplotlib.pyplot as plt
    import random
```

```
[ ] train_path = "FruitinAmazon/train/"
    test_path = "FruitinAmazon/test/"
```

```
os.listdir(train_path)
```

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

```python
def visualise(train_path):

  class_dirs = []
  for folder in os.listdir(train_path):
    class_dirs.append(folder)

  images = []
  labels = []

  for class_dir in class_dirs:
    class_path = os.path.join(train_path, class_dir)

    image_files = []
    for image in os.listdir(class_path):
      image_files.append(image)

    random_image_file = random.choice(image_files)
    image_path = os.path.join(class_path, random_image_file)

    images.append(plt.imread(image_path))
    labels.append(class_dir)

  fig, axes = plt.subplots(2, len(class_dirs) // 2, figsize = (12, 6))

  for i, (image, label) in enumerate(zip(images, labels)):
    row = i // (len(class_dirs) // 2)
    col = i % (len(class_dirs) // 2)

    axes[row, col].imshow(image)
    axes[row, col].set_title(label)
    axes[row, col].axis('off')

  plt.show()
```

```python
visualise(train_path)
```

```python
def check_for_corrupted(dir):
  sub_dir = []
  for folder in os.listdir(dir):
    sub_dir.append(folder)

  for dir in sub_dir:
    class_path = os.path.join(train_path, dir)

    image_files = []
    for image in os.listdir(class_path):
      image_files.append(image)

      for image in image_files:
        image = os.path.join(class_path, image)

        try:
          with Image.open(image) as img:
            img.verify()
        except Exception as e:
          print(f"Corrupted image found {image}")
  print("No corrupted image found")

check_for_corrupted(train_path)
```

```
No corrupted image found
```

Task 2: Loading and Preprocessing Image Data in keras:

```
img_height = 128
img_width = 128
batch_size = 32
validation_split = 0.2

rescale = tf.keras.layers.Rescaling(1./255)

train_ds = tf.keras.utils.image_dataset_from_directory(
    train_path,
    labels='inferred',
    label_mode='int',
    image_size=(img_height, img_width),
    interpolation='nearest',
    batch_size=batch_size,
    shuffle=True,
    validation_split=validation_split,
    subset='training',
    seed=123
)

train_ds = train_ds.map(lambda x, y: (rescale(x), y))

val_ds = tf.keras.utils.image_dataset_from_directory(
    test_path,
    labels='inferred',
    label_mode='int',
    image_size=(img_height, img_width),
    interpolation='nearest',
    batch_size=batch_size,
    shuffle=False,
    seed=123
)

val_ds = val_ds.map(lambda x, y: (rescale(x), y))

test_ds = tf.keras.utils.image_dataset_from_directory(
    test_path,
    labels='inferred',
    label_mode='int',
    image_size=(img_height, img_width),
    interpolation='nearest',
    batch_size=batch_size,
    shuffle=False,
    seed=123
)

test_ds = test_ds.map(lambda x, y: (rescale(x), y))
```

```
Found 90 files belonging to 6 classes.
Using 72 files for training.
Found 30 files belonging to 6 classes.
Found 30 files belonging to 6 classes.
```

Task 3 - Implement a CNN with Convolutional Architecture:

```
model = keras.Sequential([
    layers.Conv2D(32, (3, 3), padding='same', strides=1, activation='relu', input_shape=(128, 128, 3)),
    layers.MaxPooling2D((2, 2), strides=2),
    layers.Conv2D(32, (3, 3), padding='same', strides=1, activation='relu'),
    layers.MaxPooling2D((2, 2), strides=2),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(128, activation='relu'),
    layers.Dense(6, activation="softmax")
])
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. 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)
```

```
model.summary()
```

```
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 128, 128, 32)	896
max_pooling2d_4 (MaxPooling2D)	(None, 64, 64, 32)	0
conv2d_5 (Conv2D)	(None, 64, 64, 32)	9,248
max_pooling2d_5 (MaxPooling2D)	(None, 32, 32, 32)	0
flatten_2 (Flatten)	(None, 32768)	0
dense_6 (Dense)	(None, 64)	2,097,216
dense_7 (Dense)	(None, 128)	8,320
dense_8 (Dense)	(None, 6)	774

```
Total params: 2,116,454 (8.07 MB)
Trainable params: 2,116,454 (8.07 MB)
Non-trainable params: 0 (0.00 B)
```

Task 4: Compile the Model

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

Task 4: Compile the Model

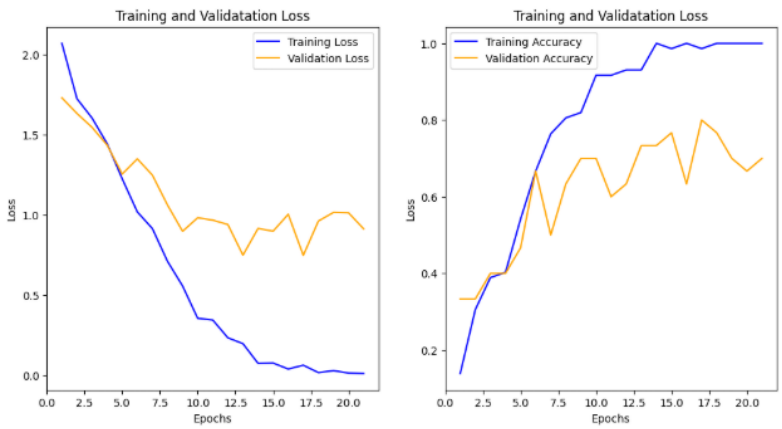
```
[ ] model.compile(
        optimizer='adam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )
```

Task 4: Train the Model

```
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

callbacks = [
    keras.callbacks.ModelCheckpoint("best_model.keras"),
    keras.callbacks.EarlyStopping(monitor="val_loss", patience=4, restore_best_weights=True),
]

history = model.fit(
    train_ds,
    epochs=250,
    batch_size=16,
    validation_data=val_ds,
    callbacks=callbacks
)
```

```
Epoch 1/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 13s 3s/step - accuracy: 0.1554 - loss: 1.9970 - val_accuracy: 0.3333 - val_loss: 1.7311
Epoch 2/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 755ms/step - accuracy: 0.3442 - loss: 1.7057 - val_accuracy: 0.3333 - val_loss: 1.6335
Epoch 3/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 591ms/step - accuracy: 0.3663 - loss: 1.6248 - val_accuracy: 0.4000 - val_loss: 1.5464
Epoch 4/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 578ms/step - accuracy: 0.3655 - loss: 1.4693 - val_accuracy: 0.4000 - val_loss: 1.4358
Epoch 5/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 616ms/step - accuracy: 0.5130 - loss: 1.2344 - val_accuracy: 0.4667 - val_loss: 1.2554
Epoch 6/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 824ms/step - accuracy: 0.6771 - loss: 1.0071 - val_accuracy: 0.6667 - val_loss: 1.3501
Epoch 7/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 578ms/step - accuracy: 0.7999 - loss: 0.8484 - val_accuracy: 0.5000 - val_loss: 1.2482
Epoch 8/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 583ms/step - accuracy: 0.8051 - loss: 0.7133 - val_accuracy: 0.6333 - val_loss: 1.0619
Epoch 9/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 615ms/step - accuracy: 0.8550 - loss: 0.5297 - val_accuracy: 0.7000 - val_loss: 0.8978
Epoch 10/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 589ms/step - accuracy: 0.9349 - loss: 0.3413 - val_accuracy: 0.7000 - val_loss: 0.9825
Epoch 11/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 572ms/step - accuracy: 0.9154 - loss: 0.3735 - val_accuracy: 0.6000 - val_loss: 0.9674
Epoch 12/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 889ms/step - accuracy: 0.9379 - loss: 0.2368 - val_accuracy: 0.6333 - val_loss: 0.9403
Epoch 13/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 578ms/step - accuracy: 0.9340 - loss: 0.2040 - val_accuracy: 0.7333 - val_loss: 0.7490
Epoch 14/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 522ms/step - accuracy: 1.0000 - loss: 0.0775 - val_accuracy: 0.7333 - val_loss: 0.9156
Epoch 15/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 587ms/step - accuracy: 0.9891 - loss: 0.0711 - val_accuracy: 0.7667 - val_loss: 0.8987
Epoch 16/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 581ms/step - accuracy: 1.0000 - loss: 0.0412 - val_accuracy: 0.6333 - val_loss: 1.0041
Epoch 17/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 520ms/step - accuracy: 0.9891 - loss: 0.0657 - val_accuracy: 0.8000 - val_loss: 0.7478
Epoch 18/250
3/3 ━━━━━━━━━━━━━━━━━━━━ 4s 861ms/step - accuracy: 1.0000 - loss: 0.0152 - val_accuracy: 0.7667 - val_loss: 0.9618
```

```python
import matplotlib.pyplot as plt

train_loss = history.history['loss']
val_loss = history.history['val_loss']

train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(range(1, len(train_loss) + 1), train_loss, label='Training Loss', color='blue')
plt.plot(range(1, len(val_loss) + 1), val_loss, label='Validation Loss', color='orange')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validatation Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(range(1, len(train_acc) + 1), train_acc, label='Training Accuracy', color='blue')
plt.plot(range(1, len(val_acc) + 1), val_acc, label='Validation Accuracy', color='orange')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validatation Loss')
plt.legend()

plt.show()
```



Task 5: Evaluate the Model

```python
test_loss, test_acc = model.evaluate(test_ds)
print(f"Test Accuracy: {test_acc:.4f}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 222ms/step - accuracy: 0.8000 - loss: 0.7478
Test Accuracy: 0.8000
```

```python
model.save('bestest_model.h5')

bestest_model = tf.keras.models.load_model('bestest_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 native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
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.
```

Task 7: Predictions and Classification Report

```python
import numpy as np
from sklearn.metrics import classification_report

y_pred_prob = bestest_model.predict(test_ds)
y_pred = np.argmax(y_pred_prob, axis=1)

y_true = []
for images, labels in test_ds.unbatch():
  y_true.append(labels.numpy())

y_true = np.array(y_true)
print(y_true)
print(classification_report(y_true, y_pred))
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 256ms/step
[0 0 0 0 0 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 4 4 4 4 4 5 5 5 5 5]
              precision    recall  f1-score   support

           0       0.62      1.00      0.77         5
           1       0.71      1.00      0.83         5
           2       1.00      0.80      0.89         5
           3       1.00      0.80      0.89         5
           4       1.00      0.80      0.89         5
           5       0.67      0.40      0.50         5

    accuracy                           0.80        30
   macro avg       0.83      0.80      0.79        30
weighted avg       0.83      0.80      0.79        30
```