Task 1: Data Understanding and Visualization:

1. Load and visualize images from a dataset stored in directories, where each subdirec-

tory represents a class.

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from PIL import Image

# Training and testing directory
train_dir = "/content/drive/MyDrive/Level 6/Artificial_Intelligence/Week5/FruitinAmazon/train"
test_dir = "/content/drive/MyDrive/Level 6/Artificial_Intelligence/Week5/FruitinAmazon/test"

img_height, img_width = 128, 128  # Increased resolution

def load_images_from_directory(directory):
    images = []
    labels = []
    class_names = sorted(os.listdir(directory))  # Ensure consistent label order
    class_dict = {class_name: idx for idx, class_name in enumerate(class_names)}
    
    for class_name in class_names:
        class_path = os.path.join(directory, class_name)
        if not os.path.isdir(class_path):
            continue
        
        for img_name in os.listdir(class_path):
            img_path = os.path.join(class_path, img_name)
            try:
                img = Image.open(img_path)
                img = img.resize((img_width, img_height), Image.LANCZOS)  # LANCZOS for sharper resizing
                images.append(np.array(img))
                labels.append(class_dict[class_name])
            except Exception as e:
```

```
                print(f"Error loading image {img_path}: {e}")
    
    return np.array(images), np.array(labels), class_names

# Load training images
X, y, class_names = load_images_from_directory(train_dir)

# Normalize pixel values to [0,1]
X = X / 255.0

# Convert labels to categorical
y = to_categorical(y, num_classes=len(class_names))

# Split data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Display some sample images
def display_sample_images(X, y, class_names, rows=2, cols=5):
    fig, axes = plt.subplots(rows, cols, figsize=(10, 5))
    axes = axes.flatten()
    
    for i in range(rows * cols):
        idx = np.random.randint(len(X))
        axes[i].imshow(X[idx], interpolation='nearest')  # Ensure sharp display
        axes[i].set_title(class_names[np.argmax(y[idx])])
        axes[i].axis('off')
    
    plt.tight_layout()
    plt.show()

# Display sample images from training set
display_sample_images(X_train, y_train, class_names)
```

2. Check for Corrupted Image:

```
import os
from PIL import Image

# Training directory
train_dir = "/content/drive/MyDrive/Level 6/Artificial_Intelligence/Week5/FruitinAmazon/train"

def remove_corrupted_images(directory):
    corrupted_images = []
    
    # Iterate through each class subdirectory
    for class_name in os.listdir(directory):
        class_path = os.path.join(directory, class_name)
        if not os.path.isdir(class_path):
            continue
        
```

```
        # Iterate through each image in the class subdirectory
        for img_name in os.listdir(class_path):
            img_path = os.path.join(class_path, img_name)
            
            try:
                # Attempt to open the image
                img = Image.open(img_path)
                img.verify()  # Verify the image is valid
            except (IOError, SyntaxError) as e:
                # If an error occurs, it's a corrupted image
                corrupted_images.append(img_path)
                os.remove(img_path)  # Remove corrupted image
                print(f"Removed corrupted image: {img_path}")
    
    # Report if no corrupted images were found
    if not corrupted_images:
        print("No corrupted images found.")

# Call the function to check and remove corrupted images
remove_corrupted_images(train_dir)
```

```
No corrupted images found.
```

Task 2: Loading and Preprocessing Image Data in keras:

```
# Define image size and batch size
img_height = 128
img_width = 128
batch_size = 32
validation_split=0.2 #80% training , 20% validation
# Create preprocessing layer for normalization
rescale = tf.keras.layers.Rescaling(1./255) # Normalize pixel values to [0,1]

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    train_dir,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
        )

    # Apply the normalization (Rescaling) to the dataset
    train_ds = train_ds.map(lambda x, y: (rescale(x), y))

    # Create validation dataset with normalization
    val_ds = tf.keras.preprocessing.image_dataset_from_directory(
        train_dir,
        labels='inferred',
        label_mode='int',
        image_size=(img_height, img_width),
        interpolation='nearest',
        batch_size=batch_size,
        shuffle=False,
        validation_split=validation_split,
        subset='validation',
        seed=123
    )
    # Apply the normalization (Rescaling) to the validation dataset
    val_ds = val_ds.map(lambda x, y: (rescale(x), y))
```

```
Found 90 files belonging to 6 classes.
Using 72 files for training.
Found 90 files belonging to 6 classes.
Using 18 files for validation.
```

Task 3 - Implement a CNN with

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.optimizers import Adam

# Define the CNN + Fully Connected Network model
model = Sequential()

# Convolutional Layer 1
model.add(Conv2D(32, (3, 3), padding='same', strides=1, activation='relu', input_shape=(128, 128, 3)))
```

```
# Max Pooling Layer 1
model.add(MaxPooling2D(pool_size=(2, 2), strides=2))

# Convolutional Layer 2
model.add(Conv2D(32, (3, 3), padding='same', strides=1, activation='relu'))

# Max Pooling Layer 2
model.add(MaxPooling2D(pool_size=(2, 2), strides=2))

# Flatten the output from the convolutional layers
model.add(Flatten())

# Hidden Layer 1 - 64 neurons
model.add(Dense(64, activation='relu'))

# Hidden Layer 2 - 128 neurons
model.add(Dense(128, activation='relu'))

# Output Layer (Number of classes = len(class_names))
model.add(Dense(len(class_names), activation='softmax'))

# Compile the model
model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])

# Model Summary
model.summary()
```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 128, 128, 32)	896
max_pooling2d_6 (MaxPooling2D)	(None, 64, 64, 32)	0
conv2d_7 (Conv2D)	(None, 64, 64, 32)	9,248
max_pooling2d_7 (MaxPooling2D)	(None, 32, 32, 32)	0
flatten_3 (Flatten)	(None, 32768)	0
dense_8 (Dense)	(None, 64)	2,097,216
dense_9 (Dense)	(None, 128)	8,320
dense_10 (Dense)	(None, 6)	774

Total params: 2,116,454 (8.07 MB)
Trainable params: 2,116,454 (8.07 MB)

Explanation of the Layers: Convolutional Layers (Conv2D) and Max Pooling Layers (MaxPooling2D): These layers are the same as in the previous CNN model. They extract features from the image and reduce spatial dimensions.

Flatten Layer:

The Flatten() layer reshapes the output from the convolutional layers into a 1D vector that can be passed to the fully connected layers.

Hidden Layers:

Dense Layer 1: Has 64 neurons, with ReLU activation. This layer learns the relationships between the features extracted by the convolutional layers.

Dense Layer 2: Has 128 neurons, also with ReLU activation. This further processes the features learned in the first hidden layer.

Output Layer:

The number of neurons is equal to the number of classes (i.e., len(class_names)).

Softmax activation is used for multi-class classification, where the model outputs probabilities for each class.

Model Compilation: Optimizer: Adam optimizer is used for gradient descent.

Loss function: categorical_crossentropy is used for multi-class classification.

Metrics: Accuracy is used to evaluate the model's performance.

Task 4: Compile the Model

```
# Compile the model
model.compile(
    optimizer='adam',  # Adam optimizer
    loss='sparse_categorical_crossentropy',  # Use 'categorical_crossentropy' if labels are one-hot encoded
    metrics=['accuracy']  # Accuracy metric
)
```

Task 4: Train the Model

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

# Define callbacks
# ModelCheckpoint: Save the best model based on validation accuracy
checkpoint_callback = ModelCheckpoint(
    'best_model.h5',  # File path to save the best model
    monitor='val_loss',  # Monitor validation loss (could also use 'val_accuracy')
    save_best_only=True,  # Save only the best model
    mode='min',  # Minimize the validation loss
    verbose=1  # Print a message when the model is saved
)

# EarlyStopping: Stop training if validation loss doesn't improve for a given number of epochs
early_stopping_callback = EarlyStopping(
    monitor='val_loss',  # Monitor validation loss
    patience=10,  # Stop after 10 epochs with no improvement
    restore_best_weights=True,  # Restore the weights of the best model
    verbose=1  # Print a message when training stops
)

# Train the model using model.fit() with callbacks
history = model.fit(
    X_train,  # Training data
    y_train,  # Training labels
```

```
    y_train,  # Training labels
    epochs=250,  # Number of epochs
    batch_size=16,  # Batch size
    validation_data=(X_val, y_val),  # Validation data
    callbacks=[checkpoint_callback, early_stopping_callback]  # Callbacks for saving the best model and early stopping
)
```

```
Epoch 1/250
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-23-9c4d3f85f548> in <cell line: 0>()
     21 
     22 # Train the model using model.fit() with callbacks
---> 23 history = model.fit(
     24     X_train,  # Training data
     25     y_train,  # Training labels

                                  1 frames

/usr/local/lib/python3.11/dist-packages/keras/src/backend/tensorflow/nn.py in sparse_categorical_crossentropy(target, output, from_logits, axis)
    723         )
    724     if len(target.shape) != len(output.shape[:-1]):
--> 725         raise ValueError(
    726             "Argument `output` must have rank (ndim) `target.ndim - 1`. "
    727             "Received: "

ValueError: Argument `output` must have rank (ndim) `target.ndim - 1`. Received: target.shape=(None, 6), output.shape=(None, 6)
```

Next steps: Explain error

```
# Remove one-hot encoding (to_categorical)
X, y, class_names = load_images_from_directory(train_dir)

# Normalize pixel values to [0,1]
X = X / 255.0

# Split data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Model Compilation using sparse_categorical_crossentropy
model.compile(
    optimizer='adam',  # Adam optimizer
    loss='sparse_categorical_crossentropy',  # For integer labels
    metrics=['accuracy']  # Accuracy metric
```

```
)

# Define callbacks
checkpoint_callback = ModelCheckpoint(
    'best_model.h5',  # File path to save the best model
    monitor='val_loss',  # Monitor validation loss
    save_best_only=True,  # Save only the best model
    mode='min',  # Minimize the validation loss
    verbose=1  # Print a message when the model is saved
)

early_stopping_callback = EarlyStopping(
    monitor='val_loss',  # Monitor validation loss
    patience=10,  # Stop after 10 epochs with no improvement
    restore_best_weights=True,  # Restore the weights of the best model
    verbose=1  # Print a message when training stops
)

# Train the model using model.fit() with callbacks
history = model.fit(
    X_train,  # Training data
    y_train,  # Training labels
    epochs=250,  # Number of epochs
    batch_size=16,  # Batch size
    validation_data=(X_val, y_val),  # Validation data
    callbacks=[checkpoint_callback, early_stopping_callback]  # Callbacks for saving the best model and early stopping
)
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 402ms/step - accuracy: 0.9740 - loss: 0.1890
Epoch 10: val_loss did not improve from 0.90578
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 497ms/step - accuracy: 0.9714 - loss: 0.1949 - val_accuracy: 0.4444 - val_loss: 1.4012
Epoch 11/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 406ms/step - accuracy: 0.9510 - loss: 0.1365
Epoch 11: val_loss did not improve from 0.90578
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 446ms/step - accuracy: 0.9523 - loss: 0.1346 - val_accuracy: 0.5556 - val_loss: 1.1070
Epoch 12/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 250ms/step - accuracy: 1.0000 - loss: 0.0362
Epoch 12: val_loss did not improve from 0.90578
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 291ms/step - accuracy: 1.0000 - loss: 0.0351 - val_accuracy: 0.6667 - val_loss: 1.2310
Epoch 13/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 261ms/step - accuracy: 1.0000 - loss: 0.0247
Epoch 13: val_loss did not improve from 0.90578
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 310ms/step - accuracy: 1.0000 - loss: 0.0248 - val_accuracy: 0.6667 - val_loss: 1.0883
Epoch 14/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 277ms/step - accuracy: 1.0000 - loss: 0.0132
Epoch 14: val_loss did not improve from 0.90578
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 328ms/step - accuracy: 1.0000 - loss: 0.0131 - val_accuracy: 0.6111 - val_loss: 1.2333
Epoch 15/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 261ms/step - accuracy: 1.0000 - loss: 0.0063
Epoch 15: val_loss did not improve from 0.90578
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 308ms/step - accuracy: 1.0000 - loss: 0.0061 - val_accuracy: 0.5000 - val_loss: 1.5465
Epoch 16/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 416ms/step - accuracy: 1.0000 - loss: 0.0032
Epoch 16: val_loss did not improve from 0.90578
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 484ms/step - accuracy: 1.0000 - loss: 0.0033 - val_accuracy: 0.5556 - val_loss: 1.6598
Epoch 17/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 456ms/step - accuracy: 1.0000 - loss: 0.0021
Epoch 17: val_loss did not improve from 0.90578
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 526ms/step - accuracy: 1.0000 - loss: 0.0021 - val_accuracy: 0.5556 - val_loss: 1.5062
Epoch 18/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 246ms/step - accuracy: 1.0000 - loss: 0.0016
Epoch 18: val_loss did not improve from 0.90578
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 285ms/step - accuracy: 1.0000 - loss: 0.0016 - val_accuracy: 0.5556 - val_loss: 1.4697
Epoch 19/250
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 255ms/step - accuracy: 1.0000 - loss: 0.0015
Epoch 19: val_loss did not improve from 0.90578
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 295ms/step - accuracy: 1.0000 - loss: 0.0015 - val_accuracy: 0.6111 - val_loss: 1.5423
Epoch 19: early stopping
```

Task 5: Evaluate the Model

```
from tensorflow.keras.preprocessing import image_dataset_from_directory

# Load the test data (assuming the test data is in a similar format to the training data)
test_ds = image_dataset_from_directory(
    test_dir,
    labels='inferred',
    label_mode='int',
    image_size=(img_height, img_width),  # Ensure test images are resized to match training images
    interpolation='nearest',
    batch_size=batch_size,
    shuffle=False
)

# Apply normalization to the test dataset (same as training and validation datasets)
test_ds = test_ds.map(lambda x, y: (rescale(x), y))

# Evaluate the model on the test dataset
test_loss, test_accuracy = model.evaluate(test_ds)

# Print the results
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")
```

```
Found 30 files belonging to 6 classes.
1/1 ━━━━━━━━━━━━━━━━━━━━ 7s 7s/step - accuracy: 0.6000 - loss: 1.0478
Test Loss: 1.0478088855743408
Test Accuracy: 0.6000000238418579
```

```
# Save the model to an .h5 file
model.save('my_model.keras')
```

```
from tensorflow.keras.models import load_model

# Load the model in the Keras format
loaded_model = load_model('my_model.keras')
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/saving/saving_lib.py:757: UserWarning: Skipping variable loading for optimizer 'rmspro
  saveable.load_own_variables(weights_store.get(inner_path))
```

```
# Evaluate the loaded model on the test dataset
```

```
test_loss, test_accuracy = loaded_model.evaluate(test_ds)

# Print the results
print(f"Test Loss (after reloading): {test_loss}")
print(f"Test Accuracy (after reloading): {test_accuracy}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step - accuracy: 0.6000 - loss: 1.0478
Test Loss (after reloading): 1.0478088855743408
Test Accuracy (after reloading): 0.6000000238418579
```

Task 7: Predictions and Classification Report

```
import numpy as np
from sklearn.metrics import classification_report
import tensorflow as tf
import os

# Get class names from the directory structure
class_names = sorted(os.listdir(test_dir))  # List of class names

# Get the test dataset (make sure it's in the same format as train_ds)
test_ds = tf.keras.preprocessing.image_dataset_from_directory(
    test_dir,
    labels='inferred',
    label_mode='int',
    image_size=(img_height, img_width),
    batch_size=batch_size,
    shuffle=False
)

# Get true labels from the test dataset
true_labels = np.concatenate([y.numpy() for _, y in test_ds], axis=0)

# Make predictions on the test dataset
predictions = loaded_model.predict(test_ds)

# Convert predicted probabilities to class labels
predicted_labels = np.argmax(predictions, axis=-1)

# Ensure true_labels and predicted_labels are 1D arrays
true_labels = true_labels.flatten()
predicted_labels = predicted_labels.flatten()
```

```
predicted_labels = predicted_labels.flatten()

# Generate the classification report
report = classification_report(true_labels, predicted_labels, target_names=class_names)

# Print the classification report
print(report)
```

```
Found 30 files belonging to 6 classes.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 412ms/step
              precision    recall  f1-score   support

        acai       0.50      1.00      0.67         5
     cupuacu       0.50      0.60      0.55         5
    graviola       0.75      0.60      0.67         5
     guarana       1.00      0.40      0.57         5
     pupunha       0.67      0.80      0.73         5
      tucuma       1.00      0.40      0.57         5
```