Data Collection and Preprocessing Phase

Date	15 March 2024
Team ID	SWTID1720033149
Project Title	Visual Diagnostics: Detecting Tomato Plant Diseases With Leaf Image Analysis
Maximum Marks	6 Marks

Preprocessing Template

The images will be preprocessed by resizing, normalizing, augmenting, denoising, adjusting contrast, detecting edges, converting colour space, cropping, batch normalizing, and whitening data. These steps will enhance data quality, promote model generalization, and improve convergence during neural network training, ensuring robust and efficient performance across various computer vision tasks.

Section	**Description**
Data Overview	The dataset consists of images of tomato leaves categorized into various classes. The data is divided into training and validation sets, with an additional test set used for evaluation. Each image belongs to one of the predefined classes, and the directory structure organizes these images accordingly.
Importing Libraries and Setting Up	This part sets up the necessary libraries and configurations. It imports essential libraries for data handling, visualization, and model building.
Resizing	The images are resized to a target size of 256x256 pixels using the `input_shape` parameter of the `ResNet152V2` model.
Data Visualization	This part visualizes a sample of the training data. It loads images from the training directory and displays them using `matplotlib`
Normalization	The pixel values of the images are normalized to a range of [0, 1] using the `rescale` parameter in the `ImageDataGenerator`.

Data Augmentation	Data augmentation techniques such as rotation, width and height shifts, shearing, zooming, and horizontal flipping are applied to increase the diversity of the training data. This helps in making the model more robust and reduces the chances of overfitting.
Model Setup and Compilation	This part sets up the ResNet152V2 model, freezes some layers, and compiles the model. It customizes the model to fit the specific problem by adding a classification head.
Denoising	Denoising filters are applied to the images to reduce noise and enhance the quality of the images. This preprocessing step helps improve the accuracy of the model by removing unwanted artifacts from the images.
Edge Detection	Edge detection algorithms are used to highlight the prominent edges in the images. This helps in emphasizing the structural features of the tomato leaves, which can be useful for distinguishing between different classes.
Model Summary	This part prints the summary of the model architecture, providing an overview of the layers and parameters.
Colour Space Conversion	Colour space conversions are used to transform the images from one colour space to another. For instance, converting RGB images to grayscale or other colour spaces can help in focusing on specific features of the images.
Model Training	This part trains the model using the training data. It uses checkpoints to save the best model based on validation loss and splits the training into multiple sessions.
Image Cropping	Image cropping involves trimming the images to focus on regions containing objects of interest. This helps in removing irrelevant background information and concentrating on the key features of the tomato leaves.
Predictions and Visualization	This part makes predictions on test data and visualizes the results. It displays the actual and predicted classes for a set of validation images.
Model Saving	This part saves the trained model to a file for later use.
Batch Normalization	Batch normalization is applied to the input of each layer in the neural network. This technique helps in stabilizing and accelerating the training process by normalizing the inputs of each layer, thereby reducing internal covariate shifts.

<table>
<tr><th colspan="2">Data Preprocessing Code Screenshots</th></tr>
<tr><td>Loading Data</td><td>

```
import numpy as np
import pandas as pd
import os
os.chdir('/kaggle/input/tomatoleaf/tomato/')
os.listdir()
```

</td></tr>
<tr><td>Resizing</td><td>

```
base_model = ResNet152V2(input_shape=(256,256,3), include_top=False)
```

</td></tr>
<tr><td>Normalization</td><td>

```
datagen = keras.preprocessing.image.ImageDataGenerator(rescale=1/255, validation_split=0.3)
datagen2 = keras.preprocessing.image.ImageDataGenerator(rescale=1/255)
```

</td></tr>
<tr><td>Data Augmentation</td><td>

```
#Training and validation dataset
train = datagen.flow_from_directory('./train', seed=123, subset='training')

val = datagen.flow_from_directory('./train', seed=123, subset='validation')

#Test dataset for evaluation
test = datagen2.flow_from_directory('./val')
```

</td></tr>
<tr><td>Denoising</td><td>NA - Not Applicable</td></tr>
<tr><td>Data Visualization</td><td>

```
# Training data visualization

classes = os.listdir('./train')

plt.figure(figsize=(25, 10))

for i in enumerate (classes) :
    pic = os.listdir ('./train/'+i[1])[0]
    image = Image.open('./train/'+i[1]+'/'+pic)
    image = np.asarray(image)
    plt. subplot (2,5,i[0]+1)
    plt. title(' {0} / Shape = {1}'. format (i[1], image.shape))
    plt. imshow (image)
plt.show()
```

</td></tr>
</table>

<table>
<tr><td>Model Setup and Compilation</td><td>

```
from tensorflow.keras.applications import ResNet152V2

# Define the base model with pre-trained weights, excluding the top layers
base_model = ResNet152V2(input_shape=(256,256,3), include_top=False)

# Freeze the layers up to the 140th layer
for layer in base_model.layers[:140]:
    layer.trainable = False

# Unfreeze the layers from the 140th layer onwards
for layer in base_model.layers[140:]:
    layer.trainable = True
```

```
# Add your custom classification head
x = GlobalAveragePooling2D()(base_model.output)
x = Dense(1000, activation='relu')(x)  # Adjust units as needed
predictions = Dense(10, activation='softmax')(x)  # Adjust number of classes accordingly

# Create the final model
model = Model(inputs=base_model.input, outputs=predictions)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

</td></tr>
<tr><td>Edge Detection</td><td>NA - Not Applicable</td></tr>
<tr><td>Model Summary</td><td>

```
model.summary()
```

</td></tr>
<tr><td>Colour Space Conversion</td><td>NA - Not Applicable</td></tr>
<tr><td>Model Training</td><td>

```
#Model Training
model.fit(train, batch_size= 80, epochs= 15, validation_data=val )
```

</td></tr>
<tr><td>Image Cropping</td><td>Give the code snippet as an image (copy and paste the picture in this block).</td></tr>
<tr><td>Batch Normalization</td><td>

```
# Add your custom classification head
x = GlobalAveragePooling2D()(base_model.output)
x = Dense(1000, activation='relu')(x)  # Adjust units as needed
predictions = Dense(10, activation='softmax')(x)  # Adjust number of classes accordingly

# Create the final model
model = Model(inputs=base_model.input, outputs=predictions)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

</td></tr>
</table>

Predictions and Visualization	```# Prediction and visualizations
	classes = os.listdir('./val')
	plt.figure(figsize=(18, 28))
	
	for i, cls in enumerate(classes):
	    pic_list = os.listdir('./val/' + cls)
	    pic = pic_list[np.random.randint(len(pic_list) - 1)]
	    image = Image.open('./val/' + cls + '/' + pic)
	    image = np.asarray(image)
	    pred = np.argmax(model.predict(image.reshape(-1, 256, 256, 3) / 255))
	
	    for j, key in enumerate(list(test.class_indices.keys())):
	        if pred == j:
	            prediction = key
	
	    plt.subplot(5, 2, i + 1) # Corrected subplot indexing
	    plt.title('Actual: {0} / Predicted: {1}'.format(cls, prediction))
	    plt.imshow(image)
	
	plt.show()```

Predictions and Visualization	```# Prediction and visualizations
	classes = os.listdir('./val')
	plt.figure(figsize=(18, 28))
	
	for i, cls in enumerate(classes):
	    pic_list = os.listdir('./val/' + cls)
	    pic = pic_list[np.random.randint(len(pic_list) - 1)]
	    image = Image.open('./val/' + cls + '/' + pic)
	    image = np.asarray(image)
	    pred = np.argmax(model.predict(image.reshape(-1, 256, 256, 3) / 255))
	
	    for j, key in enumerate(list(test.class_indices.keys())):
	        if pred == j:
	            prediction = key
	
	    plt.subplot(5, 2, i + 1) # Corrected subplot indexing
	    plt.title('Actual: {0} / Predicted: {1}'.format(cls, prediction))
	    plt.imshow(image)
	
	plt.show()```