Data Collection and Preprocessing Phase

Date	28 November 2024
Team ID	739771
Project Title	Deep fruit veg: Automated Fruit And Veg Identification
Maximum Marks	6 Marks

Preprocessing Template

The dataset consists of 30,000 images representing 30 classes of fruits and vegetables. Images are split into training (60%), validation (20%), and test (20%) sets. This diverse dataset ensures comprehensive learning, enabling accurate classification and robust generalization across various conditions.

Section	Description
Data Overview	The dataset consists of 30,000 images representing 30 classes of fruits and vegetables. Images are split into training (60%), validation (20%), and test (20%) sets. This diverse dataset ensures comprehensive learning, enabling accurate classification and robust generalization across various conditions.
Resizing	• Resize images to a uniform size (e.g., 224x224) to standardize input for the neural network. • **Code Snippet (Screenshot Placeholder):** Include Python code using cv2.resize or TensorFlow's image. Resize.
Normalization	Normalize pixel values to the range [0, 1] to ensure numerical stability during model training. **Code Snippet (Screenshot Placeholder):** Include code using libraries like NumPy or TensorFlow.
Data Augmentation	Apply augmentation techniques such as flipping, rotation, shifting, zooming, or shearing.
Denoising	Remove unwanted noise from images, which can improve feature extraction. **Implementation:** Apply filters like Gaussian Blur or Bilateral Filter using OpenCV (cv2.GaussianBlur orcv2.bilateralFilter).

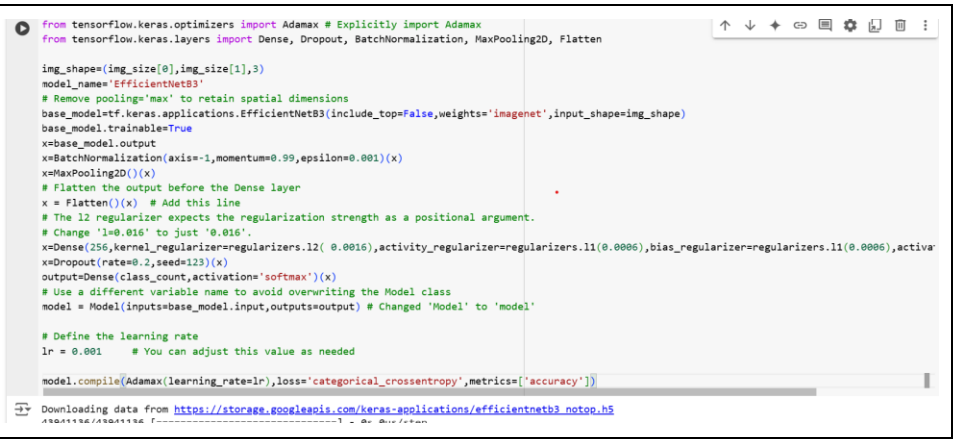
Edge Detection	Enhance object outlines to improve feature detection for classification. Use algorithms like Canny Edge Detection (cv2.Canny) or Sobel filters.
Color Space Conversion	Convert images to grayscale or alternative color spaces (e.g., HSV) to highlight specific features or reduce complexity. Use OpenCV (cv2.cvtColor) to convert between color spaces.
Image Cropping	Focus on the region of interest (fruits/vegetables) by removing unnecessary background information. Automate cropping using bounding box detection or manually crop key regions.
Batch Normalization	Normalize input activations for each layer in the neural network, improving convergence and model stability. Include batch normalization layers in the model architecture using Keras (Batch Normalization).
Data Preprocessing Code Screenshots	
Loading Data	sdir=r'/content/drive/MyDrive/ttv_plants' files=glob.glob(sdir+'/**/*.jpg',recursive=True) print(len(files))
Resizing	

```
max_value=np.max(countlist)
max_index=countlist.index(max_value)
max_class=classes[max_index]
min_value=np.min(countlist)
min_index=countlist.index(min_value)
min_class=classes[min_index]
print('the class with the maximum number of images is:',max_class)
print('the class with the minimum number of images is:',min_class)
ht=0
wt=0
train_df_sample=train_df.sample(n=100,random_state=123,axis=0)
for i in range(len(train_df_sample)):
  fpath=train_df_sample['filepaths'].iloc[i]
  img=cv2.imread(fpath)
  shape=img.shape
  ht+=shape[0]
  wt+=shape[1]
  print('average height=',ht//100,'average width=',wt//100,'aspect ratio=',ht/wt)
```

```
the class with the maximum number of images is: aloevera
the class with the minimum number of images is: aloevera
average height= 1 average width= 3 aspect ratio= 0.5225806451612903
average height= 15 average width= 23 aspect ratio= 0.6473864610111397
average height= 17 average width= 29 aspect ratio= 0.6022192333557498
```

Normalization	`import pandas as pd` `from tensorflow.keras.preprocessing.image import ImageDataGenerator` `# Now, you can use flow_from_dataframe:` `img_size = (200,260)` `working_dir = r'./'` `batch_size = 20` `trgen = ImageDataGenerator(horizontal_flip=True, rotation_range=20,zoom_range=0.2)` `t_and_v_gen = ImageDataGenerator()` `msg = '{0:70s} for train generator'.format('')` `print(msg, '\r', end='')` `train_gen = trgen.flow_from_dataframe(` `train_df,` `x_col='filepaths',` `y_col='labels',` `target_size=img_size,` `batch_size=batch_size,` `shuffle=True,` `class_mode='categorical',` `color_mode='rgb',` `)` `msg = '{0:70s} for validation generator'.format('')` `print(msg, '\r', end='')` `valid_gen = t_and_v_gen.flow_from_dataframe(` `valid_df,` `x_col='filepaths',` `y_col='labels',` `target_size=img_size,`
Data Augmentation	`Found 3600 validated image filenames belonging to 30 classes.` `Found 450 validated image filenames belonging to 30 classes.` `[ ] Start coding or generate with AI.` `from tensorflow.keras.preprocessing.image import ImageDataGenerator` `# ... (your existing code) ...` `# Define t_and_v_gen as an ImageDataGenerator instance` `t_and_v_gen = ImageDataGenerator(` `# (Add any desired data augmentation parameters here)` `)` `length = len(valid_df)` `# Define test_batch_size before using it` `test_batch_size = sorted([int(length/n) for n in range(1,length+1) if length % n==0 and length/n<=80],reverse=True)[0` `test_gen = t_and_v_gen.flow_from_dataframe(` `valid_df,` `x_col='filepaths',` `y_col='labels',` `target_size=img_size,` `batch_size=test_batch_size,` `shuffle=False,` `class_mode='categorical',` `color_mode='rgb',`
Denoising	`def show_img_samples(gen):` `t_dict=gen.class_indices` `classes=list(t_dict.keys())` `images,labels=next(gen)` `fig=plt.figure(figsize=(20,20))` `length=len(labels)` `if length<25:` `r=length` `else:` `r=25` `for i in range(r):` `plt.subplot(5,5,i+1)` `image=images[i]/255` `plt.imshow(image)` `index=np.argmax(labels[i])` `class_name=classes[index]` `plt.title(class_name,color='blue',fontsize=14)` `plt.axis('off')` `plt.show()` `[ ] show_img_samples(train_gen)` 

Edge Detection	``` length=len(labels) if length<25: r=length else: r=25 for i in range(r): plt.subplot(5,5,i+1) image=images[i]/255 plt.imshow(image) index=np.argmax(labels[i]) class_name=classes[index] plt.title(class_name,color='blue',fontsize=14) plt.axis('off') plt.show() [ ] show_img_samples(train_gen) ```
Color Space Conversion	``` def show_img_samples(gen): t_dict=gen.class_indices classes=list(t_dict.keys()) images,labels=next(gen) fig=plt.figure(figsize=(20,20)) length=len(labels) if length<25: r=length else: r=25 for i in range(r): plt.subplot(5,5,i+1) image=images[i]/255 plt.imshow(image) index=np.argmax(labels[i]) class_name=classes[index] plt.title(class_name,color='blue',fontsize=14) plt.axis('off') plt.show() ```
Image Cropping	``` def show_img_samples(gen): t_dict=gen.class_indices classes=list(t_dict.keys()) images,labels=next(gen) fig=plt.figure(figsize=(20,20)) length=len(labels) ```

Batch Normalization	

```
from tensorflow.keras.optimizers import Adamax # Explicitly import Adamax
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization, MaxPooling2D, Flatten

img_shape=(img_size[0],img_size[1],3)
model_name='EfficientNetB3'
# Remove pooling='max' to retain spatial dimensions
base_model=tf.keras.applications.EfficientNetB3(include_top=False,weights='imagenet',input_shape=img_shape)
base_model.trainable=True
x=base_model.output
x=BatchNormalization(axis=-1,momentum=0.99,epsilon=0.001)(x)
x=MaxPooling2D()(x)
# Flatten the output before the Dense layer
x = Flatten()(x)  # Add this line
# The l2 regularizer expects the regularization strength as a positional argument.
# Change 'l=0.016' to just '0.016'.
x=Dense(256,kernel_regularizer=regularizers.l2( 0.0016),activity_regularizer=regularizers.l1(0.0006),bias_regularizer=regularizers.l1(0.0006),activa
x=Dropout(rate=0.2,seed=123)(x)
output=Dense(class_count,activation='softmax')(x)
# Use a different variable name to avoid overwriting the Model class
model = Model(inputs=base_model.input,outputs=output) # Changed 'Model' to 'model'

# Define the learning rate
lr = 0.001     # You can adjust this value as needed

model.compile(Adamax(learning_rate=lr),loss='categorical_crossentropy',metrics=['accuracy'])
```

```
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb3_notop.h5
```