Data Collection and Preprocessing Phase

Date	20 July 2024
Team ID	SWTID1720109344
Project Title	Rice Type Classification Using Cnn
Maximum Marks	6 Marks

Preprocessing Template

The images will be preprocessed by resizing, normalizing, augmenting, denoising, adjusting contrast, detecting edges, converting color 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.

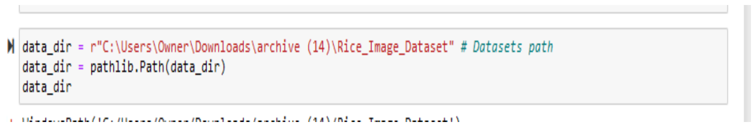
Section	**Description**
Data Overview	The dataset for this project consists of images of five rice varieties: Arborio, Basmati, Ipsala, Jasmine, and Karacadag, with 600 images per category. Each image is preprocessed by resizing to 224x224 pixels and normalizing pixel values. This dataset is used to train a deep learning model for rice type classification. The images are divided into training, validation, and test sets to evaluate the model's performance.
Resizing	To resize images to a specified target size of 224x224 pixels, OpenCV's cv2.resize function is used. Each image in the dataset is loaded, resized, and then stored back as an array. This step ensures uniform input dimensions, crucial for feeding images into the MobileNetV2 model. By standardizing the image size, the model can process and learn from the data more effectively.
Normalization	To normalize pixel values to a specific range, the pixel values of each image are divided by 255, scaling them from the range [0, 255] to [0, 1]. This normalization ensures that the data is on a consistent scale, which helps in stabilizing and speeding up the training process of the neural network. Normalized pixel values improve the model's performance and convergence.

Data Augmentation	Data augmentation techniques such as flipping, rotation, shifting, zooming, and shearing are applied to increase the diversity of the training dataset. These techniques help prevent overfitting and improve the model's generalization by creating variations of the training images. Tools like TensorFlow's ImageDataGenerator are used to apply these transformations in real-time during the training process, enhancing the robustness of the model.
Denoising	To reduce noise in the images, denoising filters such as OpenCV's cv2.fastNlMeansDenoisingColored are applied. This step helps to remove unwanted noise and artifacts from the images, improving the quality and clarity of the data. Cleaner images contribute to more accurate feature extraction and better model performance. By applying denoising, the dataset becomes more suitable for training the neural network, leading to enhanced classification results.
Edge Detection	To highlight prominent edges in the images, edge detection algorithms like Canny Edge Detection are applied using OpenCV's cv2.Canny function. This technique enhances the outlines of objects within the images, making important features more distinguishable for the model. By emphasizing these edges, the model can better learn to differentiate between the different rice varieties. This preprocessing step aids in improving the accuracy and robustness of the rice type classification.
Color Space Conversion	To convert images from one color space to another, OpenCV's cv2.cvtColor function is used. For instance, images can be converted from RGB to grayscale, which reduces computational complexity while retaining essential features. This step helps in simplifying the data and can be beneficial for certain image analysis tasks. By transforming the color space, the model can focus on intensity variations, which can enhance the classification of rice varieties.
Image Cropping	To crop images and focus on regions containing objects of interest, the images are manually or programmatically cropped to highlight key areas, such as the rice grains. This step eliminates irrelevant background and emphasizes the features crucial for classification. By focusing on the rice grains, the model can learn more effectively and improve accuracy in identifying different rice varieties. Cropping ensures that the model is trained on the most relevant parts of the images.

Batch Normalization	Batch normalization is applied to the input of each layer in the neural network to stabilize and accelerate training. By normalizing the inputs for each layer, it reduces internal covariate shift, which helps in faster convergence and improved model performance. This preprocessing step standardizes the activations, making the training process more robust and reducing the likelihood of vanishing or exploding gradients. Batch normalization enhances the overall efficiency and accuracy of the rice classification model.

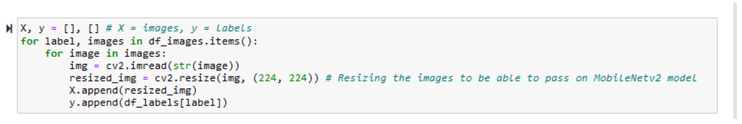
Data Preprocessing Code Screenshots

Loading Data



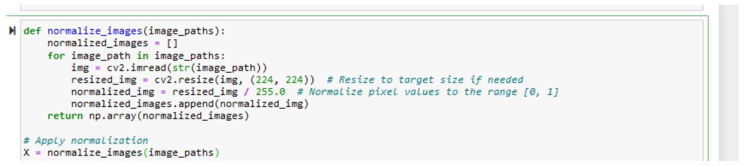
```python
data_dir = r"C:\Users\Owner\Downloads\archive (14)\Rice_Image_Dataset" # Datasets path
data_dir = pathlib.Path(data_dir)
data_dir
```

Resizing



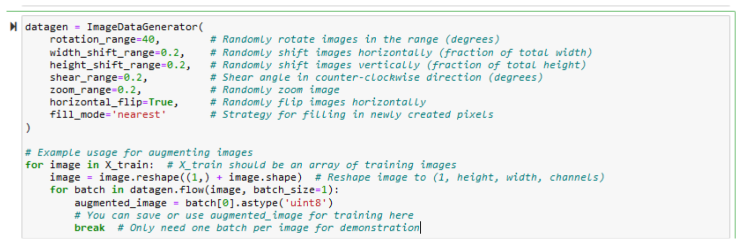
```python
X, y = [], [] # X = images, y = labels
for label, images in df_images.items():
    for image in images:
        img = cv2.imread(str(image))
        resized_img = cv2.resize(img, (224, 224)) # Resizing the images to be able to pass on MobileNetv2 model
        X.append(resized_img)
        y.append(df_labels[label])
```

Normalization



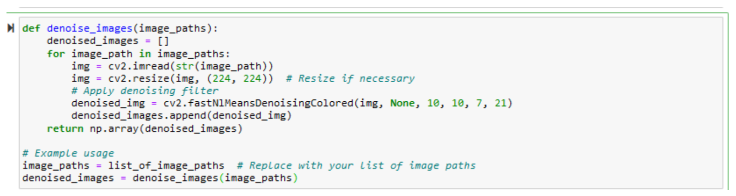
```python
def normalize_images(image_paths):
    normalized_images = []
    for image_path in image_paths:
        img = cv2.imread(str(image_path))
        resized_img = cv2.resize(img, (224, 224))  # Resize to target size if needed
        normalized_img = resized_img / 255.0  # Normalize pixel values to the range [0, 1]
        normalized_images.append(normalized_img)
    return np.array(normalized_images)

# Apply normalization
X = normalize_images(image_paths)
```

Data Augmentation



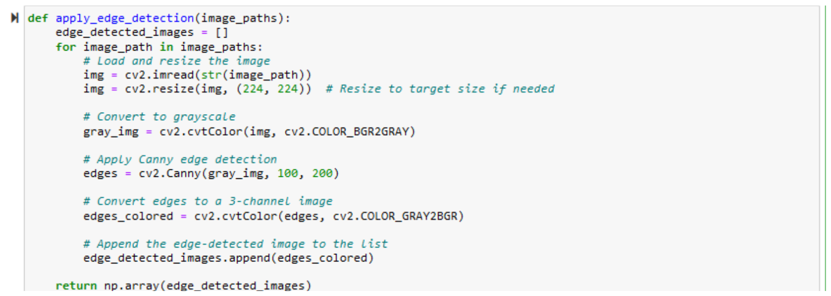
```python
datagen = ImageDataGenerator(
    rotation_range=40,        # Randomly rotate images in the range (degrees)
    width_shift_range=0.2,    # Randomly shift images horizontally (fraction of total width)
    height_shift_range=0.2,   # Randomly shift images vertically (fraction of total height)
    shear_range=0.2,          # Shear angle in counter-clockwise direction (degrees)
    zoom_range=0.2,           # Randomly zoom image
    horizontal_flip=True,     # Randomly flip images horizontally
    fill_mode='nearest'       # Strategy for filling in newly created pixels
)

# Example usage for augmenting images
for image in X_train:  # X_train should be an array of training images
    image = image.reshape((1,) + image.shape)  # Reshape image to (1, height, width, channels)
    for batch in datagen.flow(image, batch_size=1):
        augmented_image = batch[0].astype('uint8')
        # You can save or use augmented_image for training here
        break  # Only need one batch per image for demonstration
```

Denoising



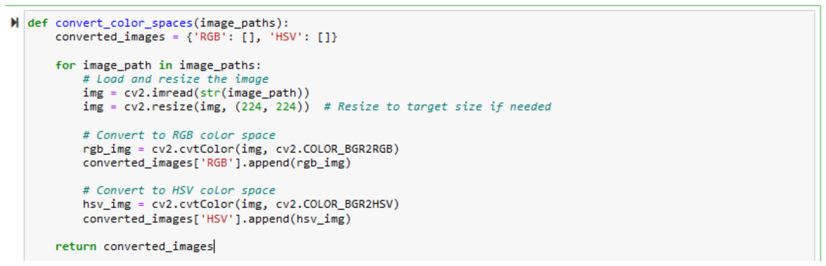
```python
def denoise_images(image_paths):
    denoised_images = []
    for image_path in image_paths:
        img = cv2.imread(str(image_path))
        img = cv2.resize(img, (224, 224))  # Resize if necessary
        # Apply denoising filter
        denoised_img = cv2.fastNlMeansDenoisingColored(img, None, 10, 10, 7, 21)
        denoised_images.append(denoised_img)
    return np.array(denoised_images)

# Example usage
image_paths = list_of_image_paths  # Replace with your list of image paths
denoised_images = denoise_images(image_paths)
```

Edge Detection



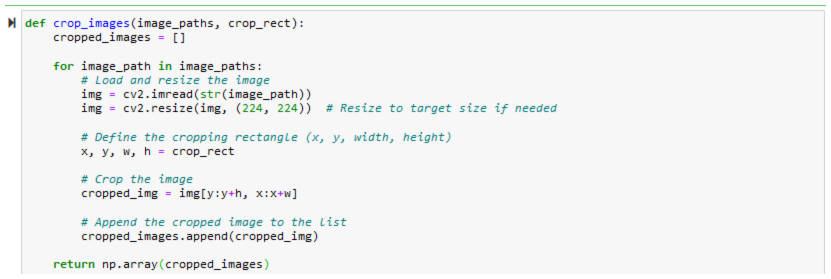
```
def apply_edge_detection(image_paths):
    edge_detected_images = []
    for image_path in image_paths:
        # Load and resize the image
        img = cv2.imread(str(image_path))
        img = cv2.resize(img, (224, 224))  # Resize to target size if needed

        # Convert to grayscale
        gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

        # Apply Canny edge detection
        edges = cv2.Canny(gray_img, 100, 200)

        # Convert edges to a 3-channel image
        edges_colored = cv2.cvtColor(edges, cv2.COLOR_GRAY2BGR)

        # Append the edge-detected image to the list
        edge_detected_images.append(edges_colored)

    return np.array(edge_detected_images)
```

Color Space Conversion



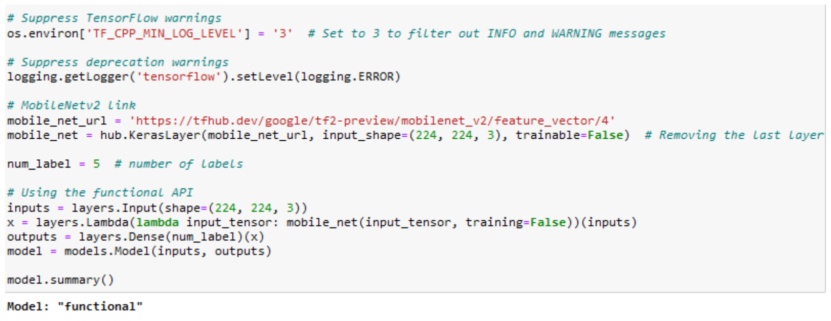
```
def convert_color_spaces(image_paths):
    converted_images = {'RGB': [], 'HSV': []}

    for image_path in image_paths:
        # Load and resize the image
        img = cv2.imread(str(image_path))
        img = cv2.resize(img, (224, 224))  # Resize to target size if needed

        # Convert to RGB color space
        rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        converted_images['RGB'].append(rgb_img)

        # Convert to HSV color space
        hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
        converted_images['HSV'].append(hsv_img)

    return converted_images
```

Image Cropping



```
def crop_images(image_paths, crop_rect):
    cropped_images = []

    for image_path in image_paths:
        # Load and resize the image
        img = cv2.imread(str(image_path))
        img = cv2.resize(img, (224, 224))  # Resize to target size if needed

        # Define the cropping rectangle (x, y, width, height)
        x, y, w, h = crop_rect

        # Crop the image
        cropped_img = img[y:y+h, x:x+w]

        # Append the cropped image to the list
        cropped_images.append(cropped_img)

    return np.array(cropped_images)
```

Batch Normalization



```
# Suppress TensorFlow warnings
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'  # Set to 3 to filter out INFO and WARNING messages

# Suppress deprecation warnings
logging.getLogger('tensorflow').setLevel(logging.ERROR)

# MobileNetv2 Link
mobile_net_url = 'https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4'
mobile_net = hub.KerasLayer(mobile_net_url, input_shape=(224, 224, 3), trainable=False)  # Removing the last layer

num_label = 5  # number of labels

# Using the functional API
inputs = layers.Input(shape=(224, 224, 3))
x = layers.Lambda(lambda input_tensor: mobile_net(input_tensor, training=False))(inputs)
outputs = layers.Dense(num_label)(x)
model = models.Model(inputs, outputs)

model.summary()
```

```
Model: "functional"
```