Data Collection and Preprocessing Phase

Date	4 july 2024
Team ID	SWTID1720093035
Project Title	**TechPart Vision: Personal Computer Parts Image Classification Using EfficientNet Transfer Learning**
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	Give an overview of the data, which you're going to use in your project.
Resizing	Resize images to a specified target size.
Normalization	Normalize pixel values to a specific range.
Data Augmentation	Apply augmentation techniques such as flipping, rotation, shifting, zooming, or shearing
Denoising	Apply denoising filters to reduce noise in the images.
Edge Detection	Apply edge detection algorithms to highlight prominent edges in the images.

Color Space Conversion	Convert images from one color space to another.
Image Cropping	Crop images to focus on the regions containing objects of interest.
Batch Normalization	Apply batch normalization to the input of each layer in the neural network.

Data Preprocessing Code Screenshots

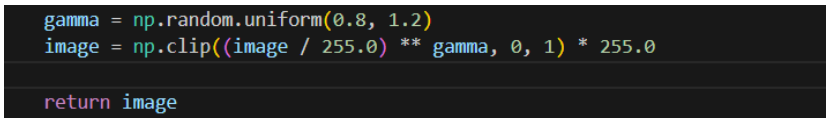
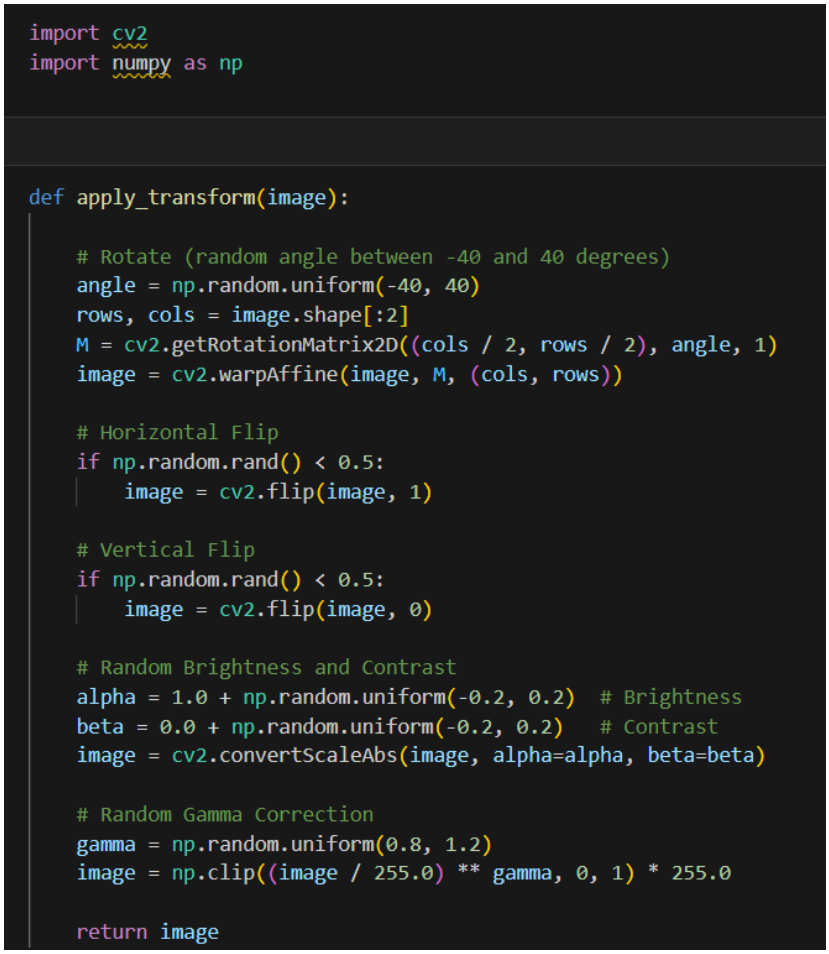
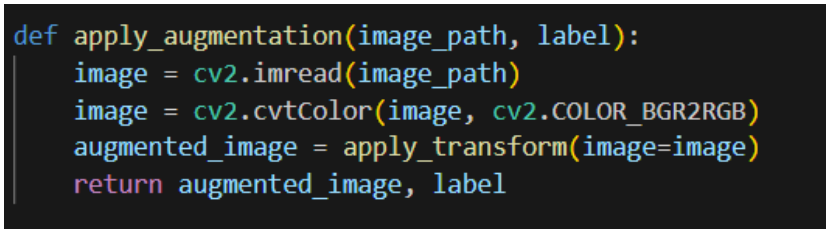
Loading Data

```
import os
import cv2
import pandas as pd
import matplotlib.pyplot as plt

directory = "/content/pc_parts"

labels = os.listdir(directory)
print("Labels:", labels)

def denoise_image(image):
    denoised_image = cv2.fastNlMeansDenoisingColored(image, None, 10, 10, 7, 21)
    return denoised_image

def read_data(folder):
    data, label, paths = [], [], []
    for l in labels:
        path = f"{folder}/{l}/"
        folder_data = os.listdir(path)
        for image_path in folder_data:
            img = cv2.imread(path + image_path)
            if img is not None:
                denoised_img = denoise_image(img)
                data.append(denoised_img)
                label.append(l)
                paths.append(os.path.join(directory, l, image_path))
    return data, label, paths

all_data, all_labels, all_paths = read_data(directory)

df = pd.DataFrame({
    'image': all_data,
    'path': all_paths,
    'label': all_labels
})
```

Resizing

```
train_gen=gen.flow_from_dataframe(balanced_df, x_col='path', y_col='label', target_size=(255,255),seed=123,
                                  class_mode='categorical', color_mode='rgb', shuffle=True, batch_size=32)
```

```
def resize_image(image, size=(255, 255)):
    resized_image = cv2.resize(image, size)
    return resized_image
```

Normalization	
Data Augmentation	
Denoising	
Edge Detection	
Color Space Conversion	

```
    gamma = np.random.uniform(0.8, 1.2)
    image = np.clip((image / 255.0) ** gamma, 0, 1) * 255.0

    return image
```

```
import cv2
import numpy as np


def apply_transform(image):

    # Rotate (random angle between -40 and 40 degrees)
    angle = np.random.uniform(-40, 40)
    rows, cols = image.shape[:2]
    M = cv2.getRotationMatrix2D((cols / 2, rows / 2), angle, 1)
    image = cv2.warpAffine(image, M, (cols, rows))

    # Horizontal Flip
    if np.random.rand() < 0.5:
        image = cv2.flip(image, 1)

    # Vertical Flip
    if np.random.rand() < 0.5:
        image = cv2.flip(image, 0)

    # Random Brightness and Contrast
    alpha = 1.0 + np.random.uniform(-0.2, 0.2)  # Brightness
    beta = 0.0 + np.random.uniform(-0.2, 0.2)   # Contrast
    image = cv2.convertScaleAbs(image, alpha=alpha, beta=beta)

    # Random Gamma Correction
    gamma = np.random.uniform(0.8, 1.2)
    image = np.clip((image / 255.0) ** gamma, 0, 1) * 255.0

    return image
```

```
def denoise_image(image):
    denoised_image = cv2.fastNlMeansDenoisingColored(image, None, 10, 10, 7, 21)
    return denoised_image
```

```
def edge_detection(image):
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    edges = cv2.Canny(gray_image, 100, 200)
    return edges
```

```
def apply_augmentation(image_path, label):
    image = cv2.imread(image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    augmented_image = apply_transform(image=image)
    return augmented_image, label
```

Batch Normalization	EfficientNetV2B1 Model Initialisation
	`base_model=tf.keras.applications.EfficientNetV2B1(include_top=False, weights="imagenet",input_shape=(255,255,3), pooling='max')`
	`print('Created EfficientNetV2 B1 model')`
	`Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/efficientnet_v2/efficientnetv2-b1_notop.h5`
	`28456008/28456008 [==============================] - 2s 0us/step`
	`Created EfficientNetV2 B1 model`
	`base_model.trainable=True`
	`x=base_model.output`
	`x=BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001 )(x)`