Data Collection and Preprocessing Phase

Date	28 Sep 2024
Team ID	team-739715
Project Title	Real-time Bone Fracture Detection with YOLO-V8 Using X-ray Images
Maximum Marks	6 Marks

Preprocessing Template

Preprocessing involves resizing X-ray images to a consistent input size, normalizing pixel values for uniformity, and applying noise reduction techniques to enhance image clarity. Data augmentation, including rotations, flips, and brightness adjustments, is used to improve model robustness. Labels are converted into YOLO-compatible formats for efficient training.

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

```
#loading data

import cv2

import glob

import matplotlib.pyplot as plt

image_paths = glob.glob('/content/three-1')

images=[cv2.imread(img_path) for img_path in image_paths]

#Check if any images were loaded

if images:

#Display first loaded image as a sample

    plt.imshow(cv2.cvtColor(images[0], cv2.COLOR_BGR2RGB))

    plt.axis('off')

    plt.show()

else:

    print("No images found in the specified directory.")
```

```
import glob

image_paths = glob.glob('/content/three-1/*.*') # Added /*.* to find files within the directory

print(image_paths)
```

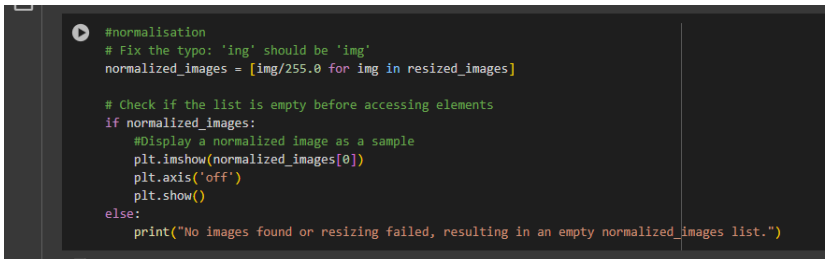
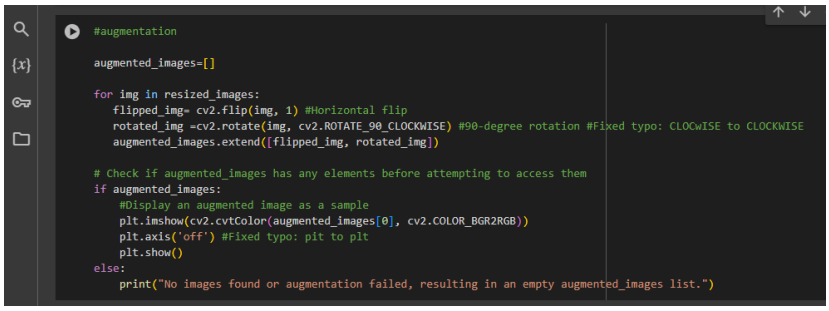
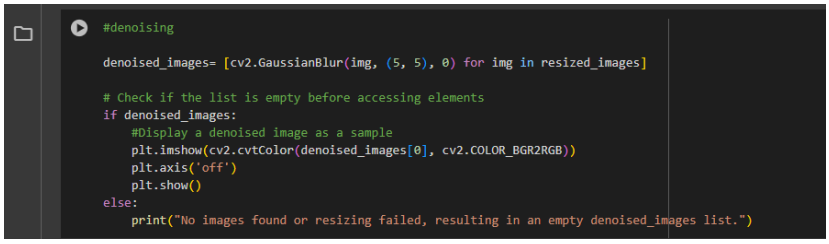
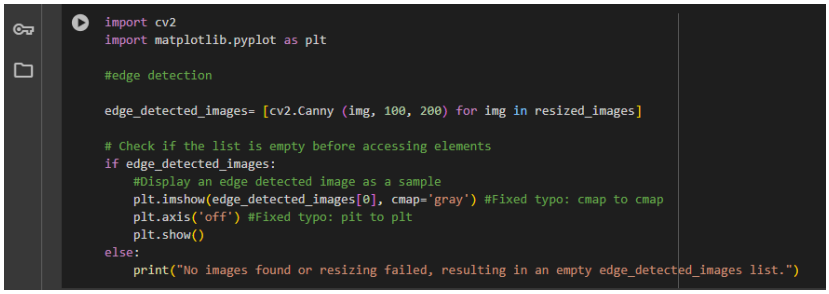
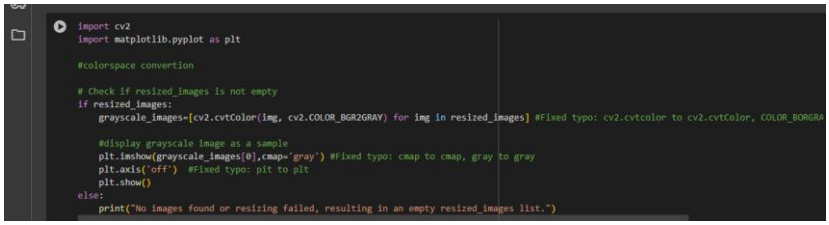
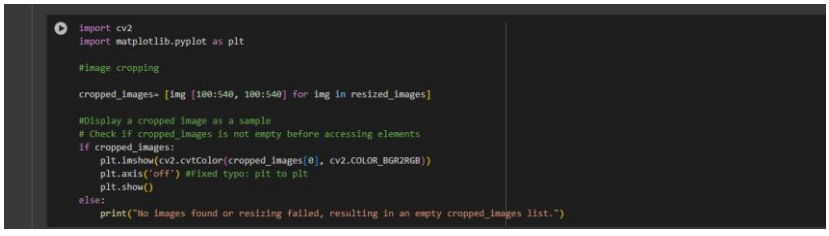
Resizing

```
import cv2
import glob
import matplotlib.pyplot as plt

image_paths = glob.glob('/content/three-1/*.*')  # Updated to find files within the directory
images = [cv2.imread(img_path) for img_path in image_paths]

if not images:
      print("No images found in the specified directory. Check the path and file existence.")
else:
      resized_images = [cv2.resize(img, (640, 640)) for img in images]

      if resized_images: #Check if resized_images has any elements before attempting to access them
          plt.imshow(cv2.cvtColor(resized_images[0], cv2.COLOR_BGR2RGB))
          plt.axis('off')
          plt.show()
      else:
          print("Resizing failed. Check if images were loaded correctly.")
```

Normalization	
Data Augmentation	
Denoising	
Edge Detection	
Color Space Conversion	
Image Cropping	

```
#normalisation
# Fix the typo: 'ing' should be 'img'
normalized_images = [img/255.0 for img in resized_images]

# Check if the list is empty before accessing elements
if normalized_images:
    #Display a normalized image as a sample
    plt.imshow(normalized_images[0])
    plt.axis('off')
    plt.show()
else:
    print("No images found or resizing failed, resulting in an empty normalized_images list.")
```

```
#augmentation

augmented_images=[]

for img in resized_images:
   flipped_img= cv2.flip(img, 1) #Horizontal flip
   rotated_img =cv2.rotate(img, cv2.ROTATE_90_CLOCKWISE) #90-degree rotation #Fixed typo: CLOCwISE to CLOCKWISE
   augmented_images.extend([flipped_img, rotated_img])

# Check if augmented_images has any elements before attempting to access them
if augmented_images:
    #Display an augmented image as a sample
    plt.imshow(cv2.cvtColor(augmented_images[0], cv2.COLOR_BGR2RGB))
    plt.axis('off') #Fixed typo: pit to plt
    plt.show()
else:
    print("No images found or augmentation failed, resulting in an empty augmented_images list.")
```

```
#denoising

denoised_images= [cv2.GaussianBlur(img, (5, 5), 0) for img in resized_images]

# Check if the list is empty before accessing elements
if denoised_images:
    #Display a denoised image as a sample
    plt.imshow(cv2.cvtColor(denoised_images[0], cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.show()
else:
    print("No images found or resizing failed, resulting in an empty denoised_images list.")
```

```
import cv2
import matplotlib.pyplot as plt

#edge detection

edge_detected_images= [cv2.Canny (img, 100, 200) for img in resized_images]

# Check if the list is empty before accessing elements
if edge_detected_images:
    #Display an edge detected image as a sample
    plt.imshow(edge_detected_images[0], cmap='gray') #Fixed typo: cmap to cmap
    plt.axis('off') #Fixed typo: pit to plt
    plt.show()
else:
    print("No images found or resizing failed, resulting in an empty edge_detected_images list.")
```

```
import cv2
import matplotlib.pyplot as plt

#colorspace convertion

# Check if resized_images is not empty
if resized_images:
    grayscale_images=[cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) for img in resized_images] #Fixed typo: cv2.cvtcolor to cv2.cvtColor, COLOR_BORGRA

    #display grayscale image as a sample
    plt.imshow(grayscale_images[0],cmap='gray') #Fixed typo: cmap to cmap, gray to gray
    plt.axis('off')  #Fixed typo: pit to plt
    plt.show()
else:
    print("No images found or resizing failed, resulting in an empty resized_images list.")
```

```
import cv2
import matplotlib.pyplot as plt

#image cropping

cropped_images= [img [100:540, 100:540] for img in resized_images]

#Display a cropped image as a sample
# Check if cropped_images is not empty before accessing elements
if cropped_images:
    plt.imshow(cv2.cvtColor(cropped_images[0], cv2.COLOR_BGR2RGB))
    plt.axis('off') #Fixed typo: pit to plt
    plt.show()
else:
    print("No images found or resizing failed, resulting in an empty cropped_images list.")
```