CHAPTER 1

INTRODUCTION

Human body consists of 206 bones with different size, shape and complexity. In humans, lower-leg bone breaks are a commonly occurred. The bone fractured risk are high in aged people due to the weak bones. The small and tiny bone fractures in the bones are difficult to analyze by the doctor. The manual process to find the fracture in the bone is time consuming and also we may have errors. So, it is necessary to develop a computer based system to reduce the time and the errors. We have many machine learning technology which are used in medical imaging as well as in the power electronics fields.

Wilhelm Roentgen discovered X-ray in 1895. X-ray is an electromagnetic radiation. The wavelength of X-ray ranges from 0.01 to 10 Nm and its frequency ranges from 30 Petahertz and 30 Exahertz. And the energy range of X-ray is from 100 Ev and 100 Kev. Fractures can be detected by X-ray.

Image processing is a powerful tool used to detect the image details with high accuracy. The bone images contain noise and edges. So, a suitable preprocessing algorithm is used to remove the noise and edges. Finally, the system is trained with the features and classification is performed by the ML algorithms.

Dimililer and Kamil have used ANN (Artificial Neural Network) to classify fracture bone but he could not classify the bone into the healthy and the fracture. Yang et al have used a contour feature of the X-ray image to detect the fracture bone. The accuracy of the system is 85% which need to be improved. To analyze and locate abnormalities in medical images of the human skeleton Sophisticated algorithm are needed.

Traditional machine learning approaches including pre-processing, feature extraction and classification have been widely applied in previous studies on fracture detection and classification. The first stage is Noise Reduction and image processing. The second stage of the project involves the process extracting unique properties from the images.

And the last step is we classify and test and train the model. The development of a system with computer-assisted automated fracture identification employing image processing techniques this efforts seeks to determine whether there is a broken bone. In this model the X-ray image is given, then by applying filter the noise is removed, then edge detection occurs, after feature extraction is used then classification method is used to classify images to detect fracture and non-fracturing images.

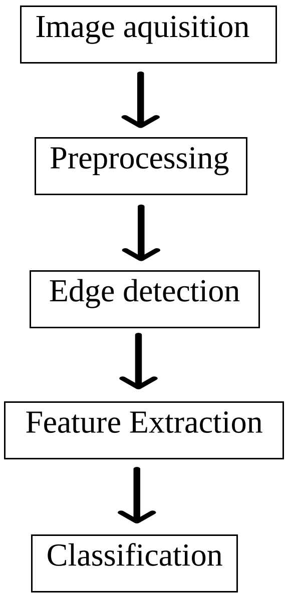


Figure 1: Image processing Steps

Description: The above image shows how dataset is loaded and processed through steps to classify the types fractures.

CT/X-ray images are taken from a Kaggle that has fractured images along with normal bone images.

LITERATURE SURVEY

Deep learning approaches for bone fracture diagnosis that have attracted a lot of study since several models and approaches are investigated to raise the diagnostic accuracy. Using X-ray images it was used in one study to identify the bone fractures that have chosen for its ability to lower computing complexity while preserving the high accuracy, hence obtaining an outstanding 98% of the accuracy [1].

CNN architecture on a large number of annotate X-ray images including various types of the fracture. Using many datasets from different medical institutions, the system's assessment of accuracy, sensitivity, specificity and F1 score unequivocally indicates CNNS tremendous strength in medical image processing [2]. CNN's natural ability is to extract complex information from the medical images was used to identify the minute elements in fracture patterns [4].

Transfer of learning and employing pre-trained models such as ResNet and VGG expedited training and improved performance. The data augmentation technique were used to improve the model generalization [2]. At last, a proposal integrating ResNet50 in a machine learning approach aimed to lower reliance on human diagnosis and improve the diagnosis accuracy [5].

RCNN algorithm regularly showed the better performance in the study over several trial environments [3]. This paper that describes the need of choosing the suitable optimization method and network topologies to raise the model performance.

Support Vector Machine (SVM), is one of the best machine learning algorithms, it was created in the 1990s and it is largely employed to pattern recognition [6]. SVM is a type of supervised machine learning in which an SVM training algorithm develops a model that predicts the category of the new example that has given as a set of training samples [7].

One of the simplest machine learning algorithms, K- Nearest Neighbour (KNN) so it is based on the supervised learning approach. The KNN algorithm categorizes new data points based on the similarity and records all available information [8].

ALGORITHM	**ACCURACY**	**ISSUES YET TO BE ADDRESSED**	**UNADDRESSED PROBLEM**
DEEP LEARNING	Up to 98% (using x-ray images)	Computational complexity reduction while maintaining the accuracy.	Dependence on large annotated datasets.
CNN ARCHITECTURE	High accuracy, sensitivity, specificity, and F1 score.	Requires extensive datasets and computational resources.	Limited generalization across diverse medical scenarios.
Transfer Learning (ResNet, VGG)	Improved performance and expedited training.	Dependence on pre - trained models from non – medical datasets.	Optimization for medical specific challenges.
ResNet50 INTEGRATION	Aimed to enhance diagnostic accuracy.	Reducing human reliance and improving automation.	Comprehensive validation across different factures types.
RCNN ALGORITM	Better performance in trails.	Selectable of suitable optimization methods and network topologies.	Scalability real world medical environments.
SUPPORT VECTOR MACHINE(SVM)	Effective for pattern recognition.	Requires optimal feature selection.	Limited to linear separability without kernel tricks.
K-NEAREST NEIGHBOUR(KNN)	Simple and effective for classification.	Computational costs increases with large datasets.	Limited by similarity- based classification.

Table 3: Comparison table

Description: The above table shows the differences between CNN, ResNet, RCNN, SVM and KNN algorithm . It defines how they differ in our training model. This also shows the issues of Existing model and issues to be resolved on Proposed model.

PROJECT REQUIREMENT SPECIFICATION

1.Data Requirements:

The first step is to gather a lot of X-ray images. These should include images with fractures (like simple, complex or tiny hairline fractures) and images without fractures. To make the system useful for everyone, the data should cover different body parts (like hands, legs and wrists) and include people of all ages and backgrounds.

Radiologists will help by marking exactly where the fractures are and what kind of fractures they are. We'll clean up the images to make them look consistent by adjusting brightness and size. To make the system smarter, we'll create more examples by flipping or rotating the images. A separate set of images will be kept aside to check and improve the system as it's developed.

2.Model Requirements:

We'll start with an existing AI model (like ResNet or EfficientNet) and train it to detect fractures. The system will:

1.Tell whether there's a fracture or not.

2.Identify what type of fracture it is.

3.Highlight the exact area on the X-ray where the fracture is.

The system should also show how it made its decision so doctors can trust it. For example, it can point out which part of the X-ray it focused on.

3.Software and Tools:

We'll use the Visual Studio (VS code) tool to build the system and the software which has been used is Ultralytics in machine learning. VS code is the powerful tool and also it is lightweight code editor that is ideal for machine learning. These both tools complement each other to streamline the development and implementation of machine learning projects.

4.Deployment Requirements:

The system should be simple to use. The doctor or technician should be able to upload an X-ray and get the results quickly. It should work well with hospital systems and give results fast.

5.Regulatory and Ethical Considerations:

Because this involves medical data, we'll ensure patient privacy by following rules like HIPAA or GDPR. This means removing personal details from the X-rays.

The system will also be tested on a wide variety of data to make sure it works well for everyone, no matter their age, gender, or background. Before it's used in real life, doctors will test it thoroughly to confirm it's reliable.

PROBLEM DEFINITION:

PROBLEM STATEMENT

Detecting fractures in X-ray images is an important but often it is a difficult task in healthcare. The goal of this project is to create smart system using machine learning that can automatically look at X-ray images and it can figure out if there is a fracture (a broken bone). The system will not only detect if there is a fracture but also point out exactly where it is in the X-ray image. This system is meant to help doctors and medical professionals by acting as an extra set of "eyes". While doctors are very skilled, sometimes the fractures can be a tricky task for them because of the complexity of X-r=`ay images. This can lead to mistakes or delays in the diagnosis, which could affect the treatment of the patient.

Solutions:

YOLOv8 is a popular algorithm that is used in machine learning, especially for the object detection and image recognition tasks.

YOLOv8 it improves on the previous versions by using more efficient techniques to detect the objects with the higher precision, even in the complex environments.

It is very easy to use, it also requires less computational power compared to the other advanced models and it can handle the multiple tasks like object classification and segmentation also it's in a single framework.

RELEVANCE OF THE PROBLEM

Detecting fractures through X-ray images is a critical issue in healthcare, with a direct impact on patient outcomes and recovery. Fracture detection is a very big critical aspect of the medical, especially in the emergency and during the orthopedic care (It is treatment for a bone, joint and muscle disorders). Misdiagnosed or undiagnosed fractures can lead to severe complications such as improper healing, chronic pain and long-term disability. The development of a machine learning- based fracture detection system addresses several pressing challenges in healthcare:

1. Reducing Diagnostic Errors: Radiologists and medical professionals often they face a highworkloads, leading to the potential oversight in interpreting X-ray images. A machine learning system can serve as a second opinion, reducing errors and ensuring to accurate diagnosis.
2. Improving the accessibility: In many of the regions, especially in rural and undeserved areas, there is a shortage of skilled radiologists. An automated fracture detection system can bridge this way of gap, enabling the faster more reliable diagnoses.
3. Enhancing efficiency: The manual analysis of X-rays is time-consuming. Automating this process can speed up the diagnosis, allowing patients to receive timely treatment and improving the overall workflow in the medical facilities.
4. Supporting complex diagnosis: Differentiating between the various types of fractures requires expertise. A robust system that identifies the type and location of the fracture with the high sensitivity that can assist doctors in planning precise treatments.
5. Improving Patient Outcomes: Early and accurate fracture detection ensures the proper intervention, reducing the risk of complications, improving recovery times and enhancing the quality of life for patients.

SYSTEM ARCHITECTURE

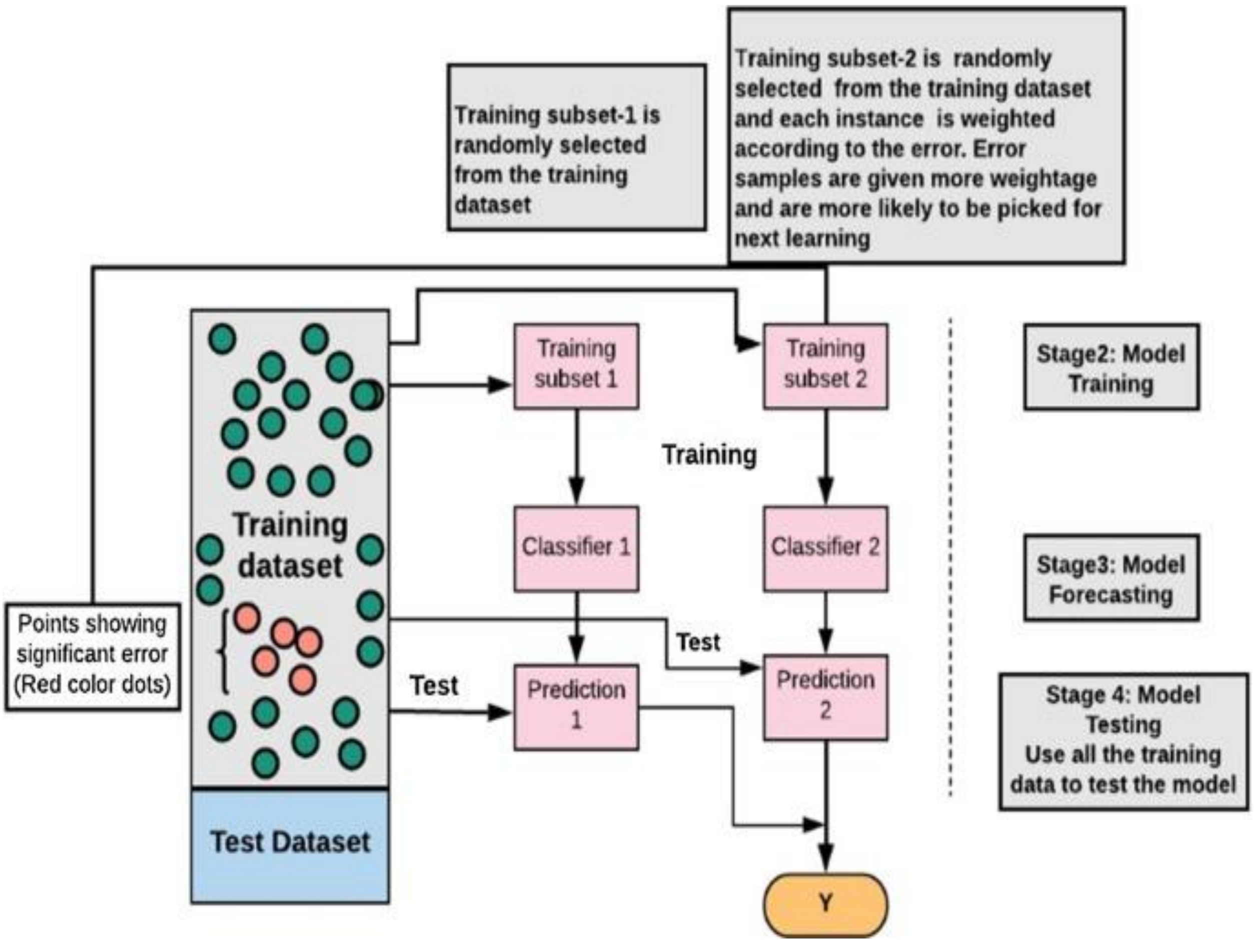
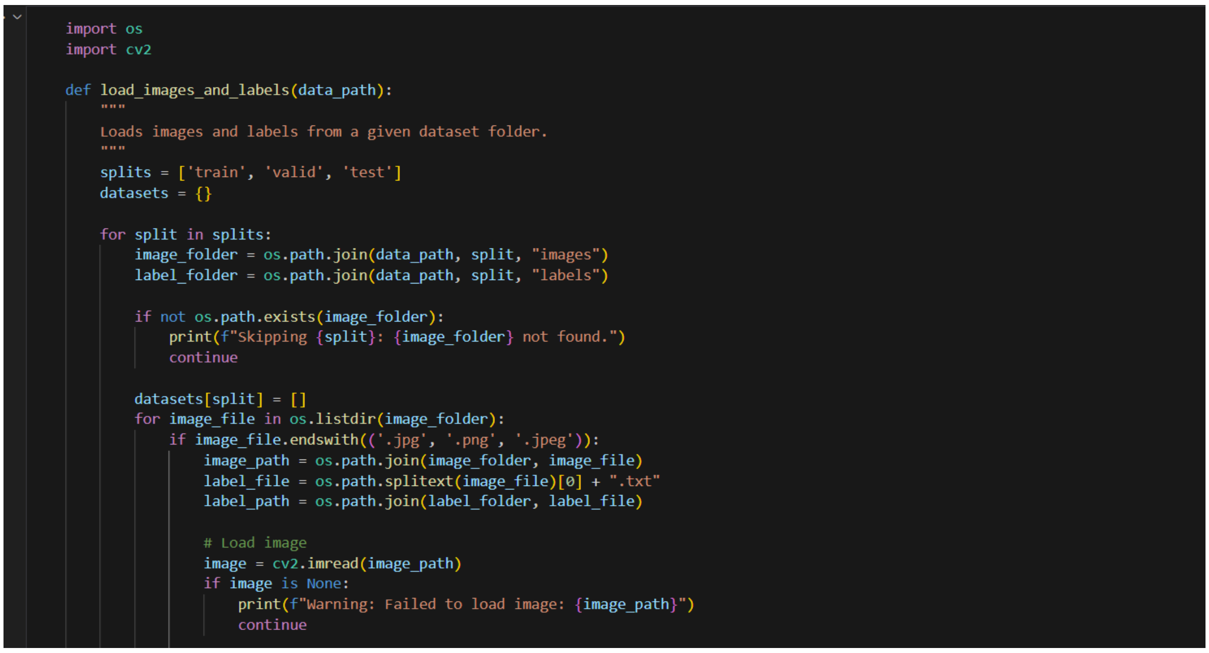

Training subset-1 is randomly selected from the training dataset
Training subset-2 is randomly selected from the training dataset and each instance is weighted according to the error. Error samples are given more weightage and are more likely to be picked for next learning
Training dataset
Test Dataset
Points showing significant error (Red color dots)
Training subset 1
Training subset 2
Training
Classifier 1
Classifier 2
Test
Test
Prediction 1
Prediction 2
Y
Stage2: Model Training
Stage3: Model Forecasting
Stage 4: Model Testing Use all the training data to test the model

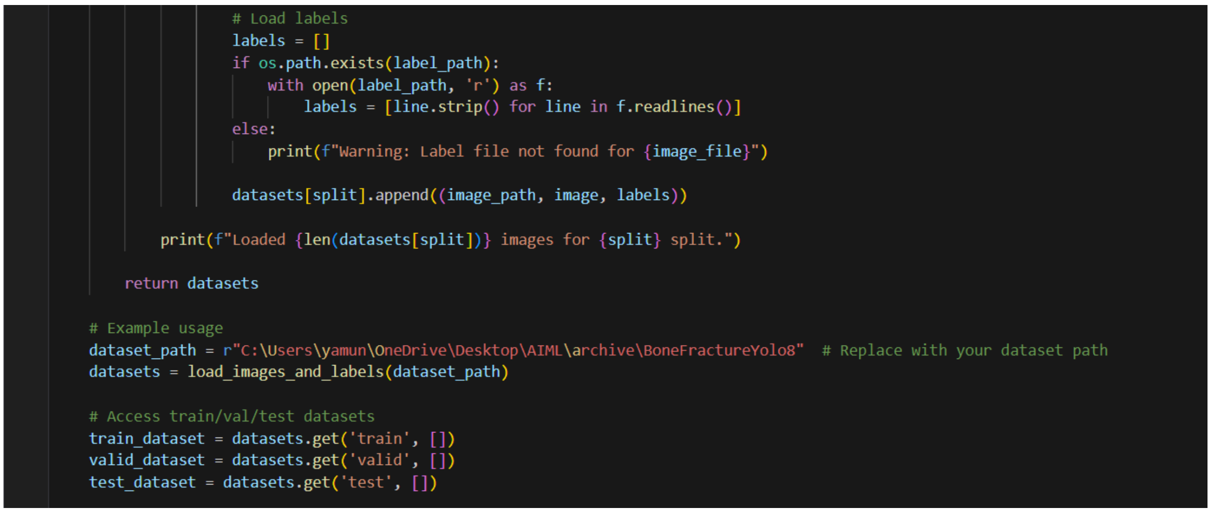
Figure 2: System Architecture

Description: The image illustrates a boosting process where a training dataset is split into subsets, with misclassified samples given higher weights in subsequent iterations. Multiple classifiers are trained sequentially, and their predictions are combined to improve overall model accuracy. The process involves stages for training, forecasting, and final testing.

IMPLEMENTATION



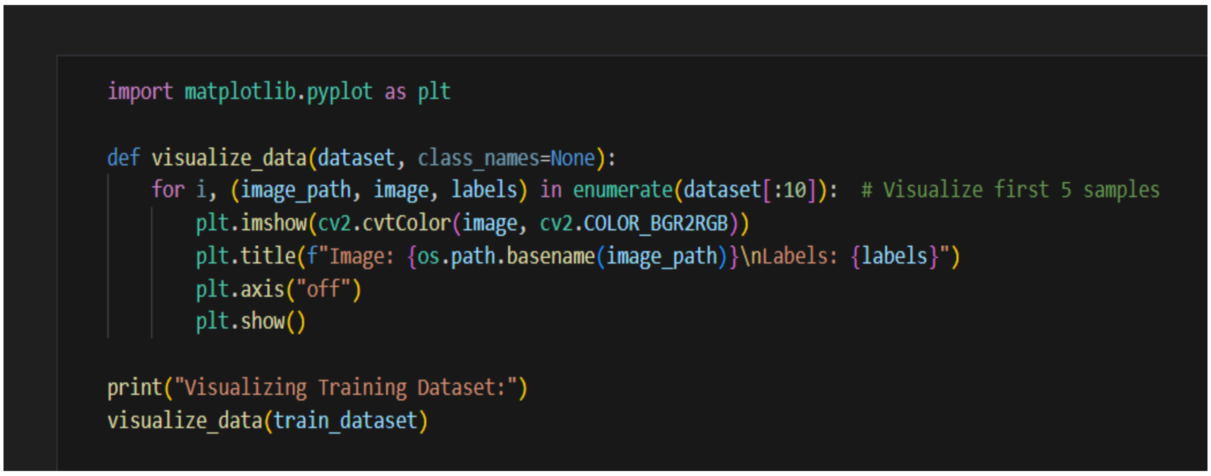
```python
import os
import cv2

def load_images_and_labels(data_path):
    """
    Loads images and labels from a given dataset folder.
    """
    splits = ['train', 'valid', 'test']
    datasets = {}

    for split in splits:
        image_folder = os.path.join(data_path, split, "images")
        label_folder = os.path.join(data_path, split, "labels")

        if not os.path.exists(image_folder):
            print(f"Skipping {split}: {image_folder} not found.")
            continue

        datasets[split] = []
        for image_file in os.listdir(image_folder):
            if image_file.endswith(('.jpg', '.png', '.jpeg')):
                image_path = os.path.join(image_folder, image_file)
                label_file = os.path.splitext(image_file)[0] + ".txt"
                label_path = os.path.join(label_folder, label_file)

                # Load image
                image = cv2.imread(image_path)
                if image is None:
                    print(f"Warning: Failed to load image: {image_path}")
                    continue
```



```python
                # Load labels
                labels = []
                if os.path.exists(label_path):
                    with open(label_path, 'r') as f:
                        labels = [line.strip() for line in f.readlines()]
                else:
                    print(f"Warning: Label file not found for {image_file}")

                datasets[split].append((image_path, image, labels))

        print(f"Loaded {len(datasets[split])} images for {split} split.")

    return datasets

# Example usage
dataset_path = r"C:\Users\yamun\OneDrive\Desktop\AIML\archive\BoneFractureYolo8"  # Replace with your dataset path
datasets = load_images_and_labels(dataset_path)

# Access train/val/test datasets
train_dataset = datasets.get('train', [])
valid_dataset = datasets.get('valid', [])
test_dataset = datasets.get('test', [])
```

Figure 3.1: Loading images and labels

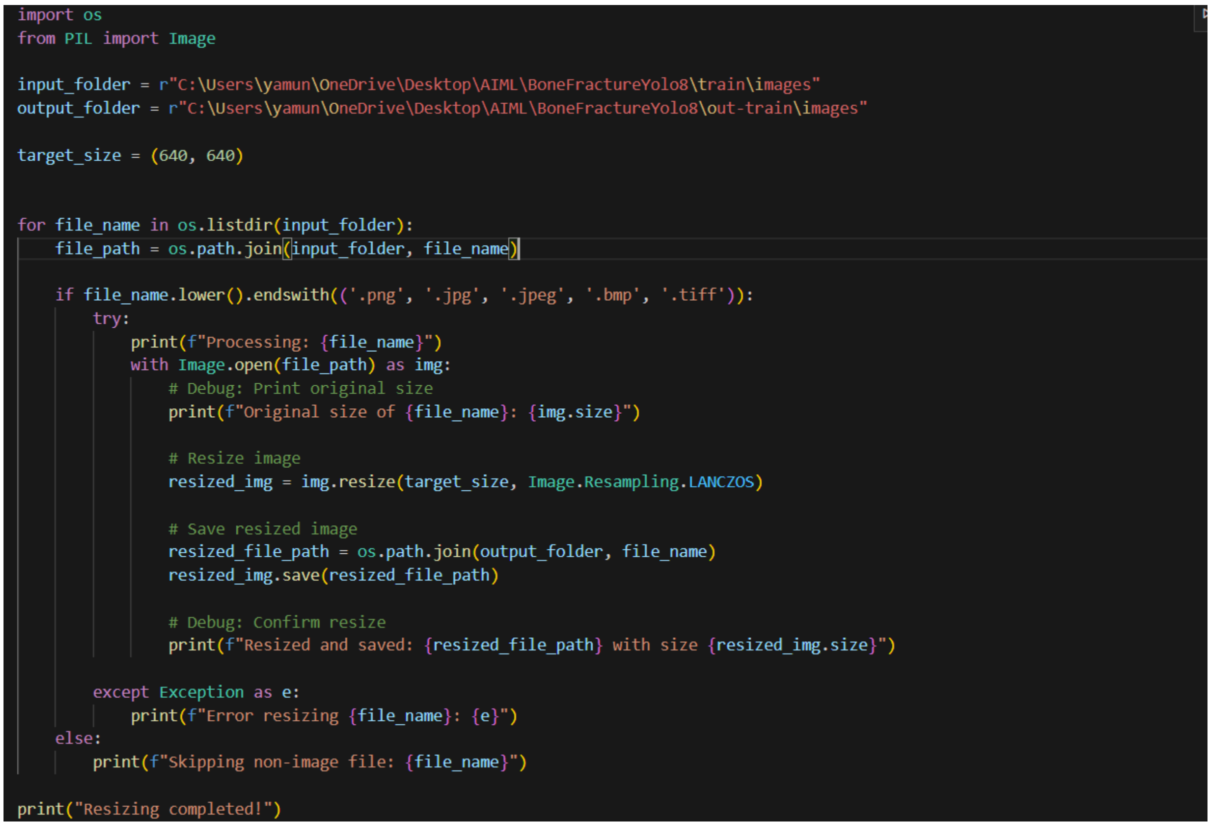
Description: The above code shows how to load our train, test and valid dataset into our project environment.



```python
import matplotlib.pyplot as plt

def visualize_data(dataset, class_names=None):
    for i, (image_path, image, labels) in enumerate(dataset[:10]):  # Visualize first 5 samples
        plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
        plt.title(f"Image: {os.path.basename(image_path)}\nLabels: {labels}")
        plt.axis("off")
        plt.show()

print("Visualizing Training Dataset:")
visualize_data(train_dataset)
```

Figure 3.2: Visualizing data

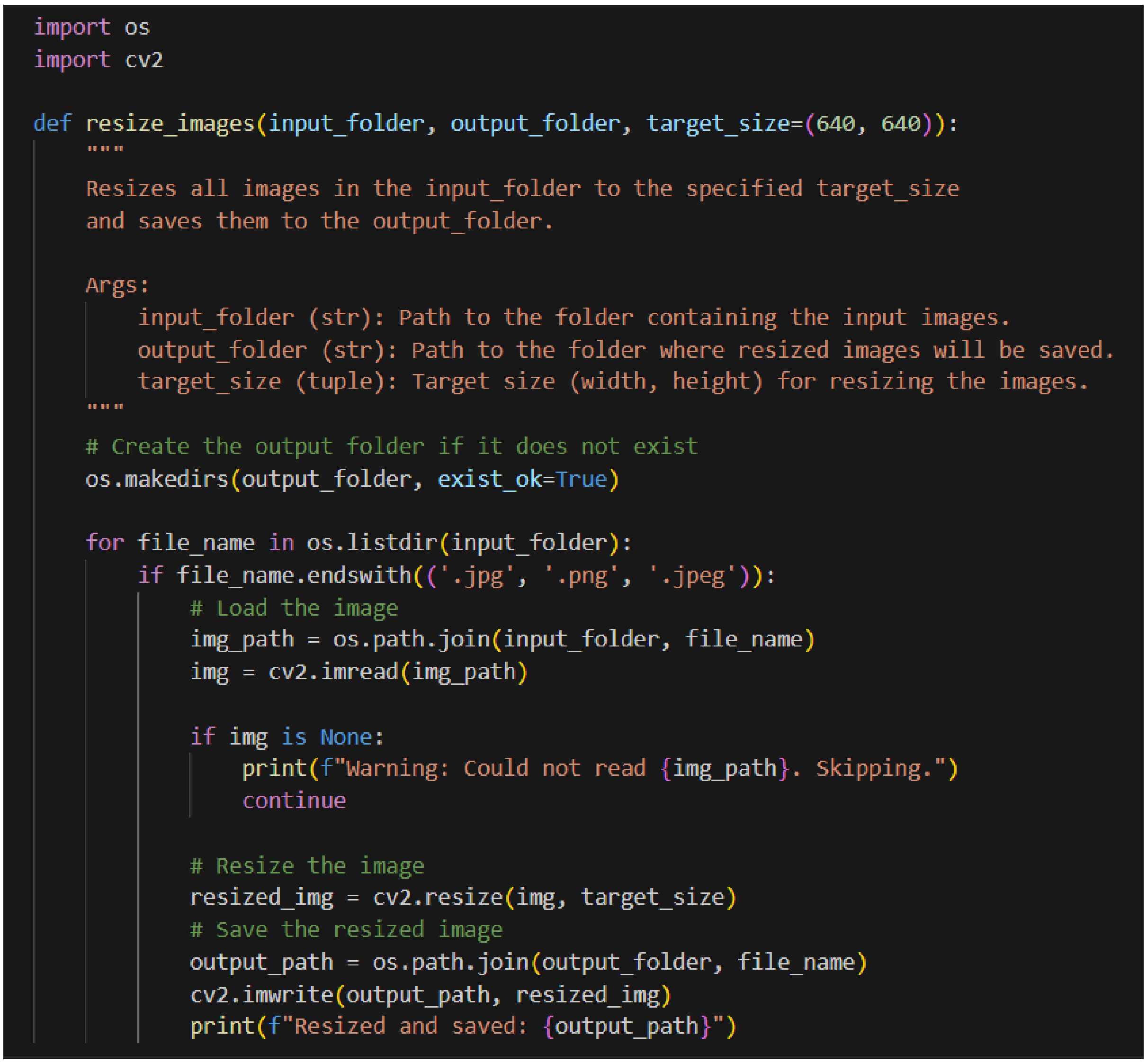
Description: The code snippet shows to visualize the images from the dataset and confirms whether dataset is loaded correctly or not.



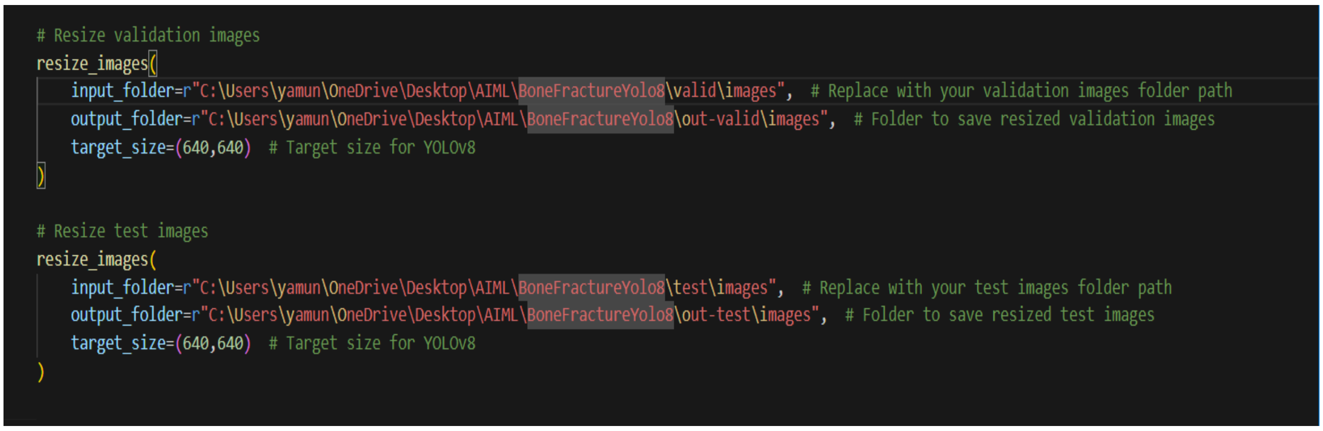
```python
import os
from PIL import Image

input_folder = r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\train\images"
output_folder = r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-train\images"

target_size = (640, 640)


for file_name in os.listdir(input_folder):
    file_path = os.path.join(input_folder, file_name)

    if file_name.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.tiff')):
        try:
            print(f"Processing: {file_name}")
            with Image.open(file_path) as img:
                # Debug: Print original size
                print(f"Original size of {file_name}: {img.size}")

                # Resize image
                resized_img = img.resize(target_size, Image.Resampling.LANCZOS)

                # Save resized image
                resized_file_path = os.path.join(output_folder, file_name)
                resized_img.save(resized_file_path)

                # Debug: Confirm resize
                print(f"Resized and saved: {resized_file_path} with size {resized_img.size}")

        except Exception as e:
            print(f"Error resizing {file_name}: {e}")
    else:
        print(f"Skipping non-image file: {file_name}")

print("Resizing completed!")
```

Figure 3.3: Resizing the training images

Description: This is the step where preprocessing of dataset starts. The above snippet shows first step of preprocessing i.e. resizing of training dataset and stores the preprocessed images in out-train-images folder.



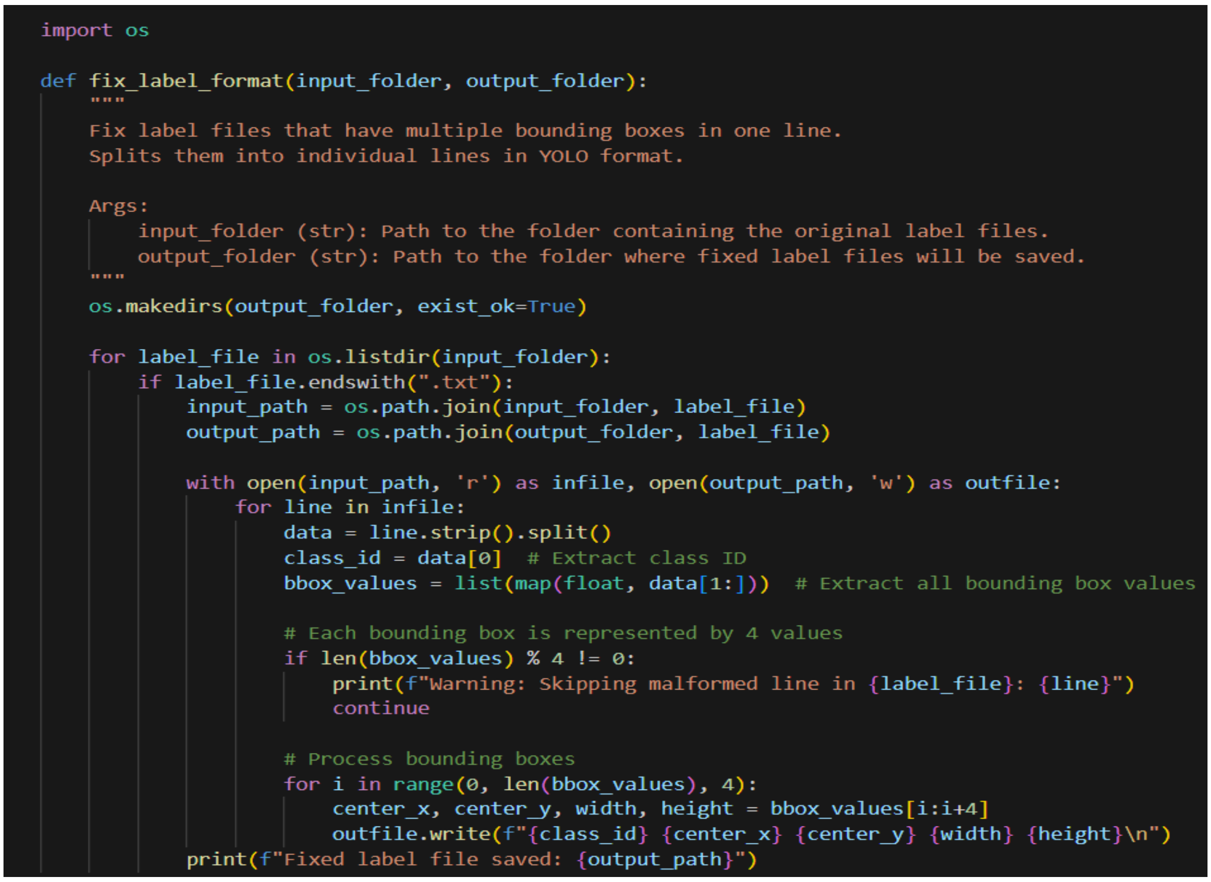
```python
import os
import cv2

def resize_images(input_folder, output_folder, target_size=(640, 640)):
    """
    Resizes all images in the input_folder to the specified target_size
    and saves them to the output_folder.

    Args:
        input_folder (str): Path to the folder containing the input images.
        output_folder (str): Path to the folder where resized images will be saved.
        target_size (tuple): Target size (width, height) for resizing the images.
    """
    # Create the output folder if it does not exist
    os.makedirs(output_folder, exist_ok=True)

    for file_name in os.listdir(input_folder):
        if file_name.endswith(('.jpg', '.png', '.jpeg')):
            # Load the image
            img_path = os.path.join(input_folder, file_name)
            img = cv2.imread(img_path)

            if img is None:
                print(f"Warning: Could not read {img_path}. Skipping.")
                continue

            # Resize the image
            resized_img = cv2.resize(img, target_size)
            # Save the resized image
            output_path = os.path.join(output_folder, file_name)
            cv2.imwrite(output_path, resized_img)
            print(f"Resized and saved: {output_path}")
```



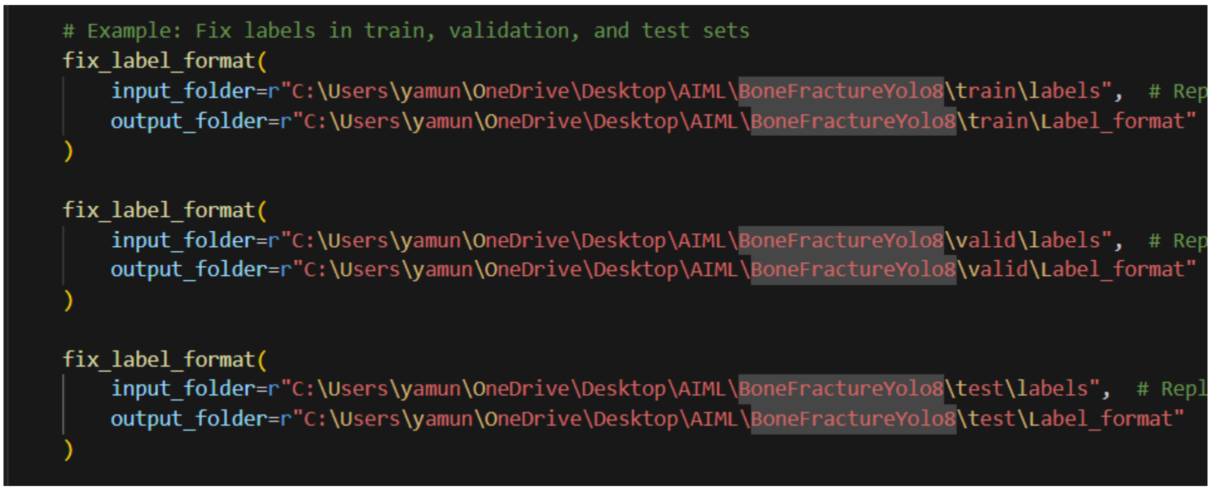
```python
# Resize validation images
resize_images(
    input_folder=r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\valid\images",  # Replace with your validation images folder path
    output_folder=r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-valid\images",  # Folder to save resized validation images
    target_size=(640,640)  # Target size for YOLOv8
)

# Resize test images
resize_images(
    input_folder=r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\test\images",  # Replace with your test images folder path
    output_folder=r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-test\images",  # Folder to save resized test images
    target_size=(640,640)  # Target size for YOLOv8
)
```

Figure 3.4: Resizing the test and validation images

Description: This snippet shows resizing of valid and testing datasets and stores them into respective out-valid and out-test folders.



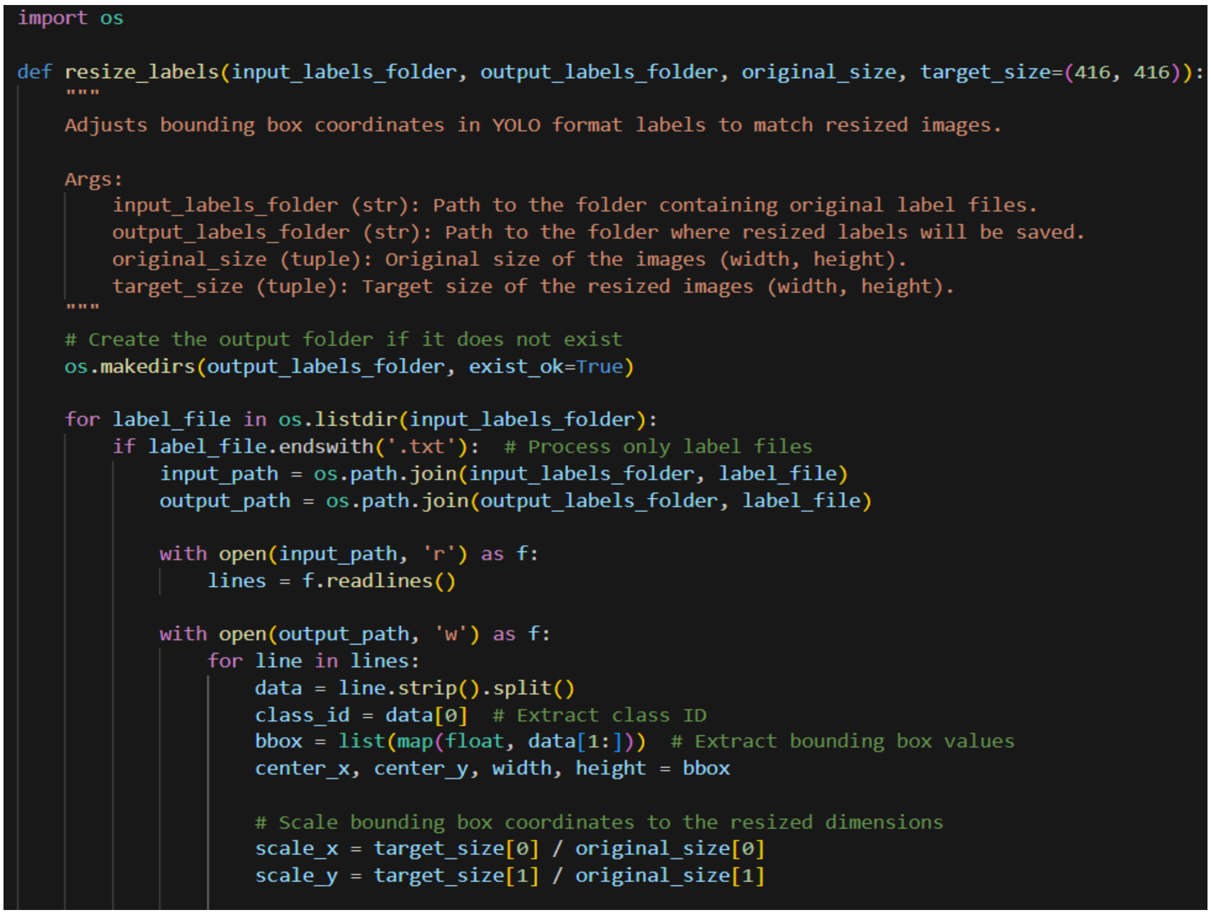
```
import os

def fix_label_format(input_folder, output_folder):
    """
    Fix label files that have multiple bounding boxes in one line.
    Splits them into individual lines in YOLO format.

    Args:
        input_folder (str): Path to the folder containing the original label files.
        output_folder (str): Path to the folder where fixed label files will be saved.
    """
    os.makedirs(output_folder, exist_ok=True)

    for label_file in os.listdir(input_folder):
        if label_file.endswith(".txt"):
            input_path = os.path.join(input_folder, label_file)
            output_path = os.path.join(output_folder, label_file)

            with open(input_path, 'r') as infile, open(output_path, 'w') as outfile:
                for line in infile:
                    data = line.strip().split()
                    class_id = data[0]  # Extract class ID
                    bbox_values = list(map(float, data[1:]))  # Extract all bounding box values

                    # Each bounding box is represented by 4 values
                    if len(bbox_values) % 4 != 0:
                        print(f"Warning: Skipping malformed line in {label_file}: {line}")
                        continue

                    # Process bounding boxes
                    for i in range(0, len(bbox_values), 4):
                        center_x, center_y, width, height = bbox_values[i:i+4]
                        outfile.write(f"{class_id} {center_x} {center_y} {width} {height}\n")
            print(f"Fixed label file saved: {output_path}")
```



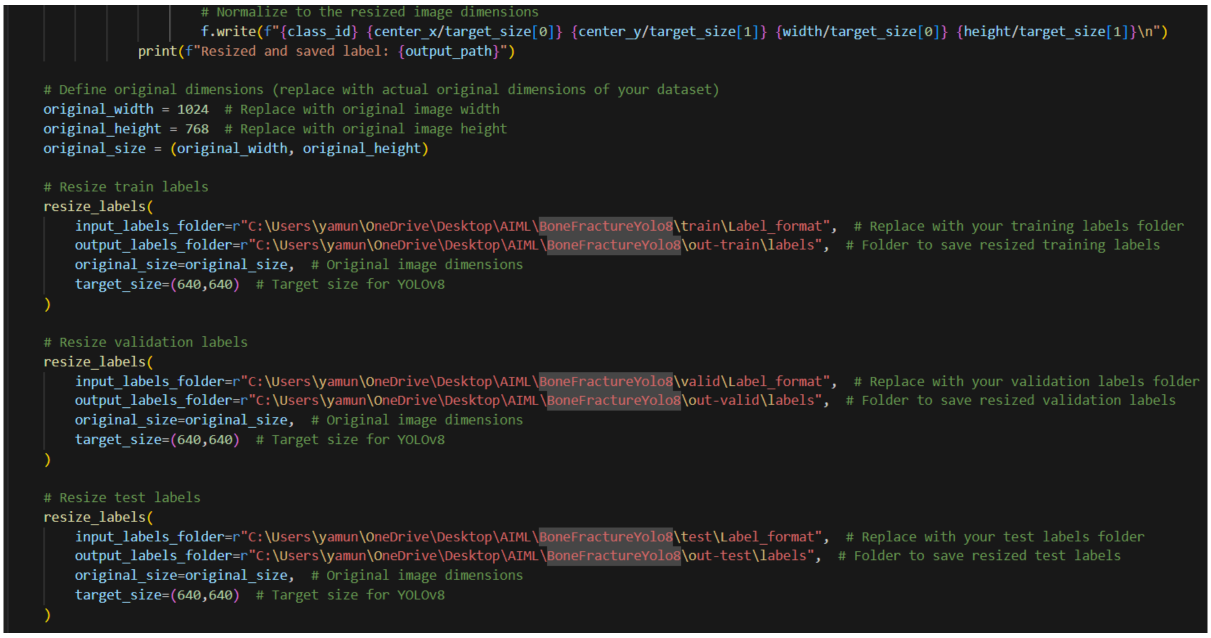
```
# Example: Fix labels in train, validation, and test sets
fix_label_format(
    input_folder=r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\train\labels",  # Rep
    output_folder=r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\train\Label_format"
)

fix_label_format(
    input_folder=r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\valid\labels",  # Rep
    output_folder=r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\valid\Label_format"
)

fix_label_format(
    input_folder=r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\test\labels",  # Repl
    output_folder=r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\test\Label_format"
)
```

Figure 3.5: Fix label format

Description: Here we have to fix the format of labels where Yolov8 model accepts the format as {class_id center_x center_y width height}.



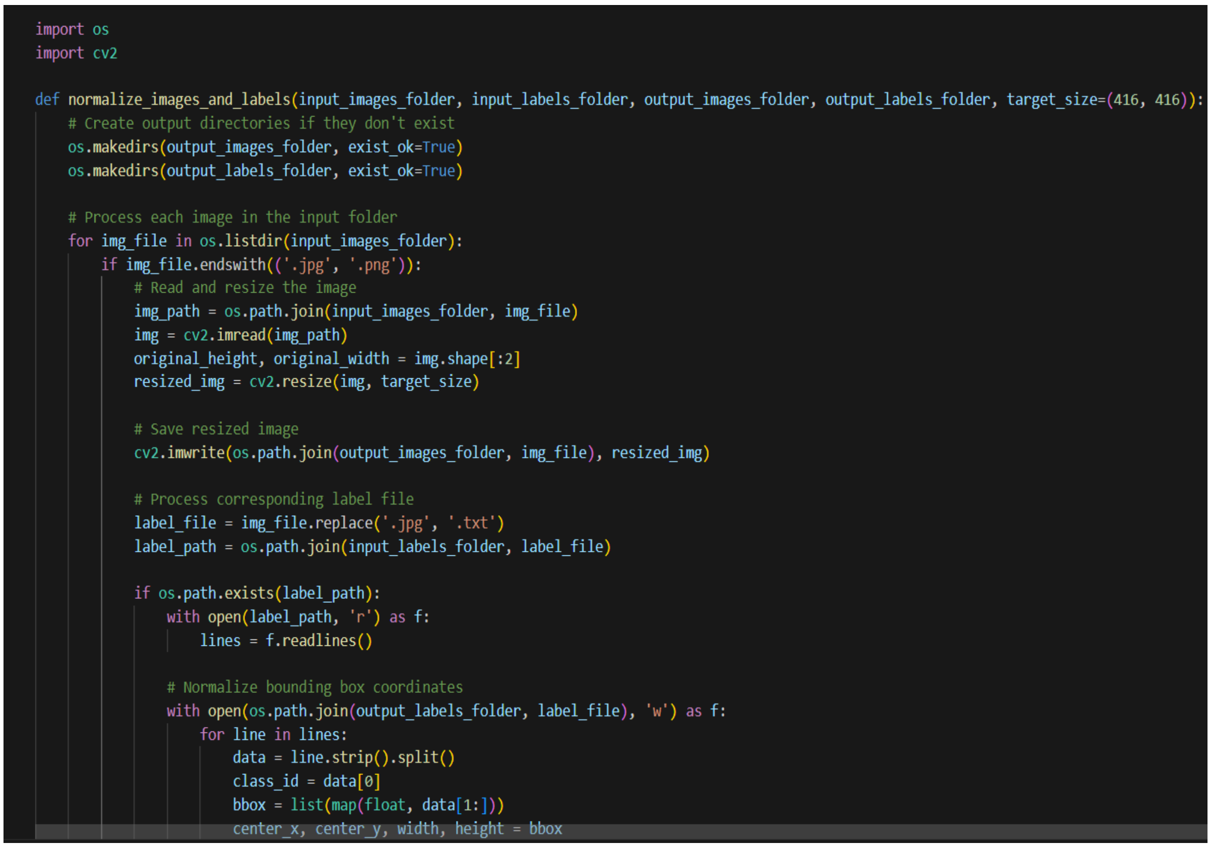
```python
import os

def resize_labels(input_labels_folder, output_labels_folder, original_size, target_size=(416, 416)):
    """
    Adjusts bounding box coordinates in YOLO format labels to match resized images.

    Args:
        input_labels_folder (str): Path to the folder containing original label files.
        output_labels_folder (str): Path to the folder where resized labels will be saved.
        original_size (tuple): Original size of the images (width, height).
        target_size (tuple): Target size of the resized images (width, height).
    """
    # Create the output folder if it does not exist
    os.makedirs(output_labels_folder, exist_ok=True)

    for label_file in os.listdir(input_labels_folder):
        if label_file.endswith('.txt'):  # Process only label files
            input_path = os.path.join(input_labels_folder, label_file)
            output_path = os.path.join(output_labels_folder, label_file)

            with open(input_path, 'r') as f:
                lines = f.readlines()

            with open(output_path, 'w') as f:
                for line in lines:
                    data = line.strip().split()
                    class_id = data[0]  # Extract class ID
                    bbox = list(map(float, data[1:]))  # Extract bounding box values
                    center_x, center_y, width, height = bbox

                    # Scale bounding box coordinates to the resized dimensions
                    scale_x = target_size[0] / original_size[0]
                    scale_y = target_size[1] / original_size[1]
```



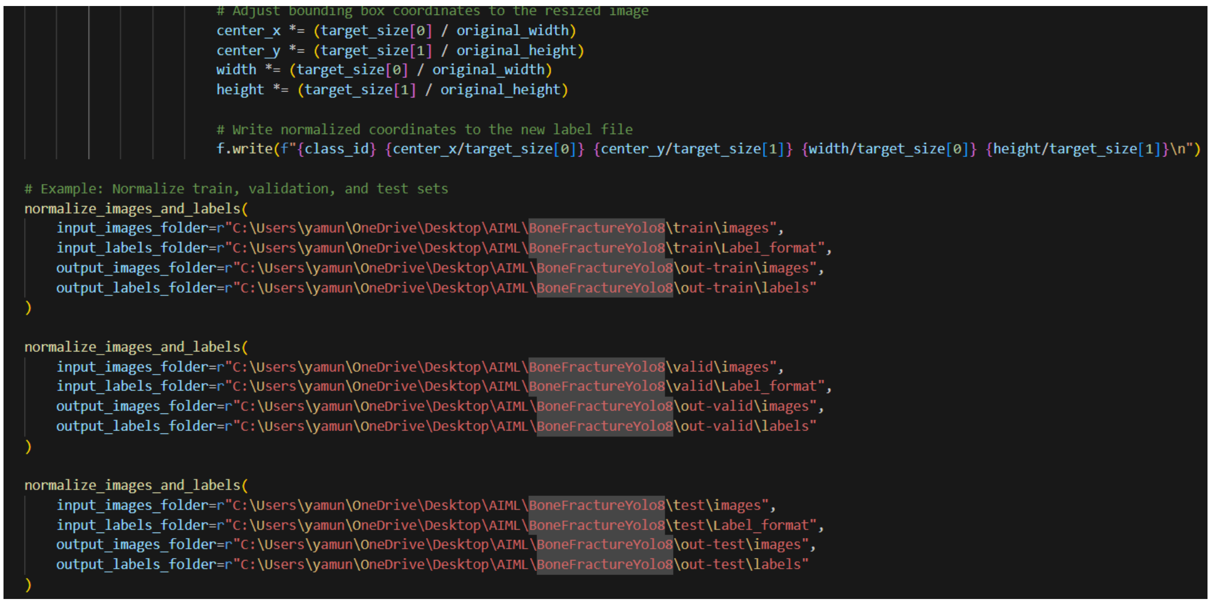
```python
                    # Normalize to the resized image dimensions
                    f.write(f"{class_id} {center_x/target_size[0]} {center_y/target_size[1]} {width/target_size[0]} {height/target_size[1]}\n")
            print(f"Resized and saved label: {output_path}")

# Define original dimensions (replace with actual original dimensions of your dataset)
original_width = 1024  # Replace with original image width
original_height = 768  # Replace with original image height
original_size = (original_width, original_height)

# Resize train labels
resize_labels(
    input_labels_folder=r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\train\Label_format",  # Replace with your training labels folder
    output_labels_folder=r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-train\labels",  # Folder to save resized training labels
    original_size=original_size,  # Original image dimensions
    target_size=(640,640)  # Target size for YOLOv8
)

# Resize validation labels
resize_labels(
    input_labels_folder=r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\valid\Label_format",  # Replace with your validation labels folder
    output_labels_folder=r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-valid\labels",  # Folder to save resized validation labels
    original_size=original_size,  # Original image dimensions
    target_size=(640,640)  # Target size for YOLOv8
)

# Resize test labels
resize_labels(
    input_labels_folder=r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\test\Label_format",  # Replace with your test labels folder
    output_labels_folder=r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-test\labels",  # Folder to save resized test labels
    original_size=original_size,  # Original image dimensions
    target_size=(640,640)  # Target size for YOLOv8
)
```

Figure 3.6: Resizing train, test and valid labels

Description: The preprocessing should be done for labels for train, valid and test datasets. The above code shows resizing of labels and storing them in the respective out-train-valid-test-Labels folders.



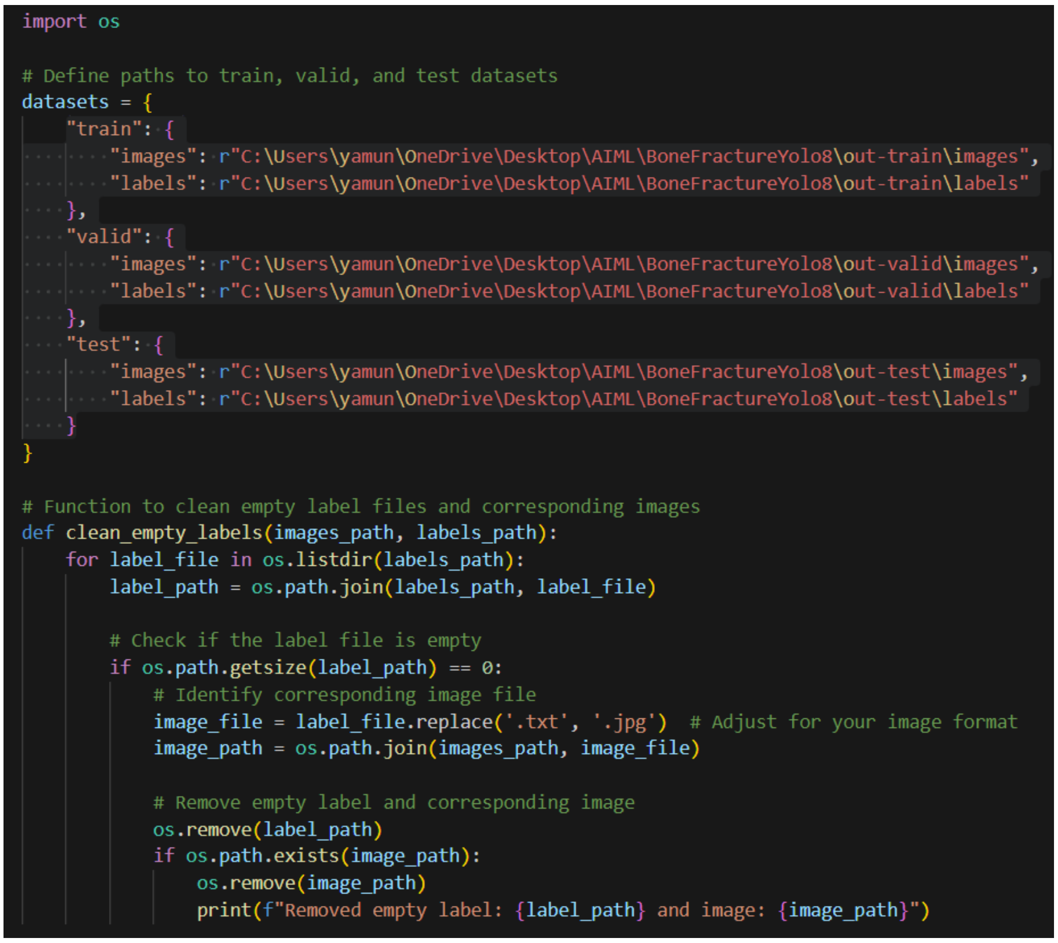
```python
import os
import cv2

def normalize_images_and_labels(input_images_folder, input_labels_folder, output_images_folder, output_labels_folder, target_size=(416, 416)):
    # Create output directories if they don't exist
    os.makedirs(output_images_folder, exist_ok=True)
    os.makedirs(output_labels_folder, exist_ok=True)

    # Process each image in the input folder
    for img_file in os.listdir(input_images_folder):
        if img_file.endswith(('.jpg', '.png')):
            # Read and resize the image
            img_path = os.path.join(input_images_folder, img_file)
            img = cv2.imread(img_path)
            original_height, original_width = img.shape[:2]
            resized_img = cv2.resize(img, target_size)

            # Save resized image
            cv2.imwrite(os.path.join(output_images_folder, img_file), resized_img)

            # Process corresponding label file
            label_file = img_file.replace('.jpg', '.txt')
            label_path = os.path.join(input_labels_folder, label_file)

            if os.path.exists(label_path):
                with open(label_path, 'r') as f:
                    lines = f.readlines()

                # Normalize bounding box coordinates
                with open(os.path.join(output_labels_folder, label_file), 'w') as f:
                    for line in lines:
                        data = line.strip().split()
                        class_id = data[0]
                        bbox = list(map(float, data[1:]))
                        center_x, center_y, width, height = bbox
```



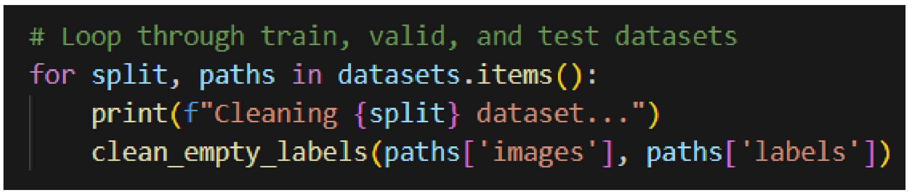
```python
                        # Adjust bounding box coordinates to the resized image
                        center_x *= (target_size[0] / original_width)
                        center_y *= (target_size[1] / original_height)
                        width *= (target_size[0] / original_width)
                        height *= (target_size[1] / original_height)

                        # Write normalized coordinates to the new label file
                        f.write(f"{class_id} {center_x/target_size[0]} {center_y/target_size[1]} {width/target_size[0]} {height/target_size[1]}\n")

# Example: Normalize train, validation, and test sets
normalize_images_and_labels(
    input_images_folder=r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\train\images",
    input_labels_folder=r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\train\Label_format",
    output_images_folder=r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-train\images",
    output_labels_folder=r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-train\labels"
)

normalize_images_and_labels(
    input_images_folder=r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\valid\images",
    input_labels_folder=r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\valid\Label_format",
    output_images_folder=r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-valid\images",
    output_labels_folder=r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-valid\labels"
)

normalize_images_and_labels(
    input_images_folder=r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\test\images",
    input_labels_folder=r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\test\Label_format",
    output_images_folder=r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-test\images",
    output_labels_folder=r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-test\labels"
)
```

Figure 3.7: Normalize images and labels

Description: The next of preprocessing involves Normalization of images and labels. The above snippet shows the normalizing of both images and labels and stores them into the respective output folders.



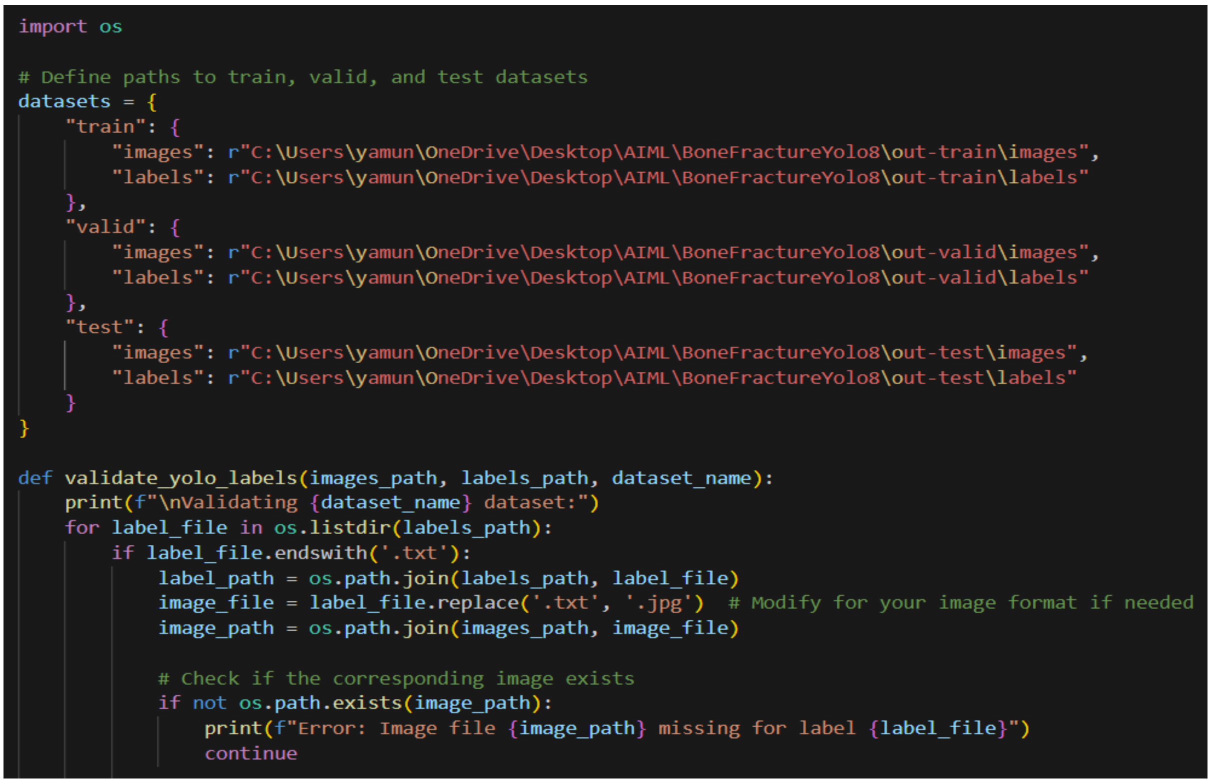
```python
import os

# Define paths to train, valid, and test datasets
datasets = {
    "train": {
        "images": r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-train\images",
        "labels": r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-train\labels"
    },
    "valid": {
        "images": r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-valid\images",
        "labels": r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-valid\labels"
    },
    "test": {
        "images": r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-test\images",
        "labels": r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-test\labels"
    }
}

# Function to clean empty label files and corresponding images
def clean_empty_labels(images_path, labels_path):
    for label_file in os.listdir(labels_path):
        label_path = os.path.join(labels_path, label_file)

        # Check if the label file is empty
        if os.path.getsize(label_path) == 0:
            # Identify corresponding image file
            image_file = label_file.replace('.txt', '.jpg')  # Adjust for your image format
            image_path = os.path.join(images_path, image_file)

            # Remove empty label and corresponding image
            os.remove(label_path)
            if os.path.exists(image_path):
                os.remove(image_path)
                print(f"Removed empty label: {label_path} and image: {image_path}")
```



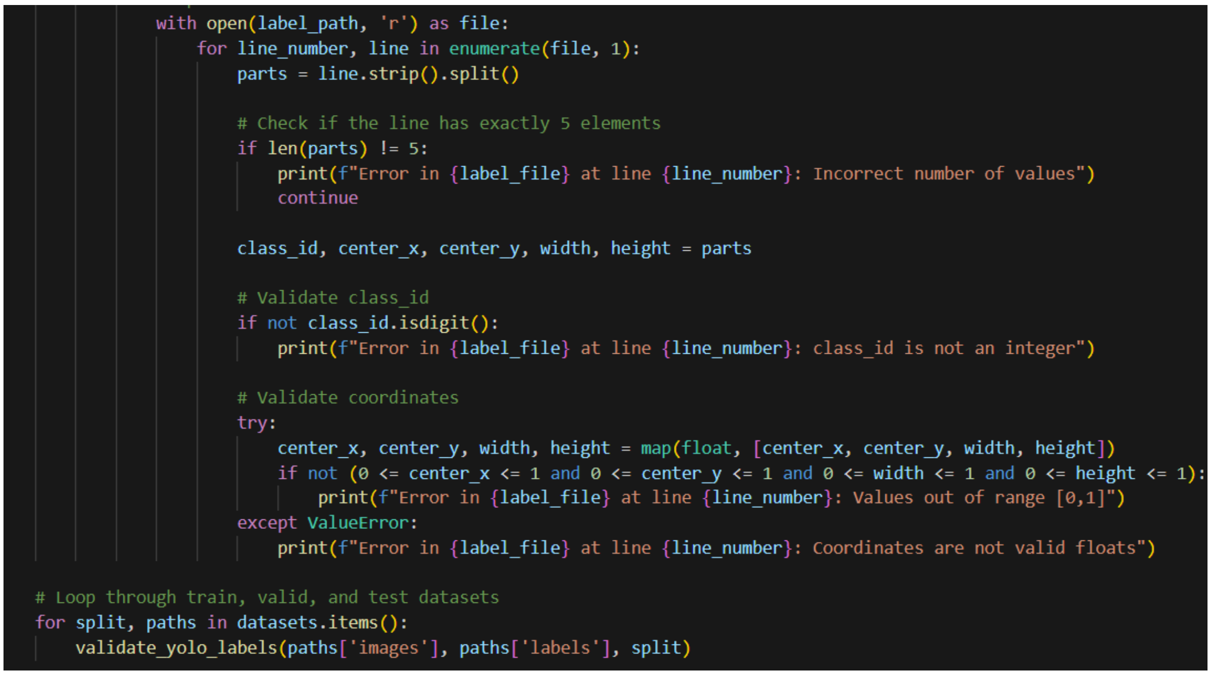
```python
# Loop through train, valid, and test datasets
for split, paths in datasets.items():
    print(f"Cleaning {split} dataset...")
    clean_empty_labels(paths['images'], paths['labels'])
```

Figure 3.8: Cleaning empty labels

Description: The model requires images only with labels and it can’t process without labels. This is the step to remove empty labels from the dataset.



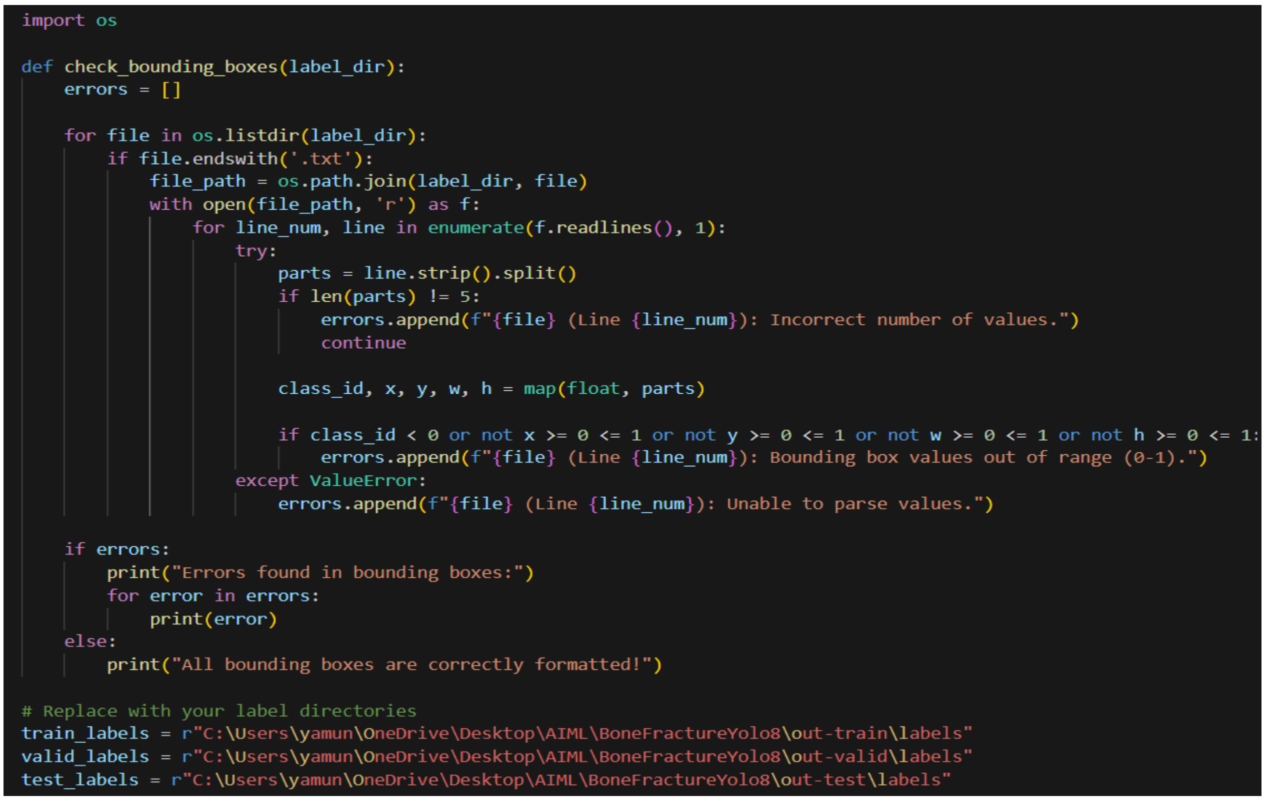
```python
import os

# Define paths to train, valid, and test datasets
datasets = {
    "train": {
        "images": r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-train\images",
        "labels": r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-train\labels"
    },
    "valid": {
        "images": r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-valid\images",
        "labels": r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-valid\labels"
    },
    "test": {
        "images": r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-test\images",
        "labels": r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-test\labels"
    }
}

def validate_yolo_labels(images_path, labels_path, dataset_name):
    print(f"\nValidating {dataset_name} dataset:")
    for label_file in os.listdir(labels_path):
        if label_file.endswith('.txt'):
            label_path = os.path.join(labels_path, label_file)
            image_file = label_file.replace('.txt', '.jpg')  # Modify for your image format if needed
            image_path = os.path.join(images_path, image_file)

            # Check if the corresponding image exists
            if not os.path.exists(image_path):
                print(f"Error: Image file {image_path} missing for label {label_file}")
                continue
```



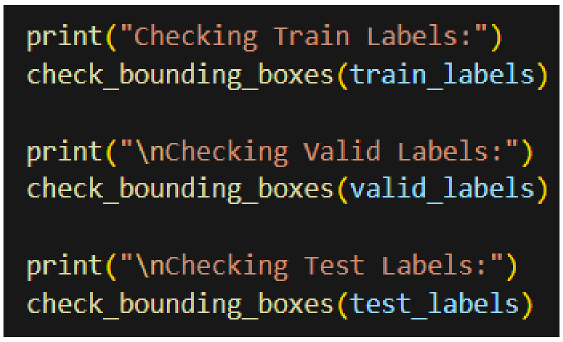
```python
            with open(label_path, 'r') as file:
                for line_number, line in enumerate(file, 1):
                    parts = line.strip().split()

                    # Check if the line has exactly 5 elements
                    if len(parts) != 5:
                        print(f"Error in {label_file} at line {line_number}: Incorrect number of values")
                        continue

                    class_id, center_x, center_y, width, height = parts

                    # Validate class_id
                    if not class_id.isdigit():
                        print(f"Error in {label_file} at line {line_number}: class_id is not an integer")

                    # Validate coordinates
                    try:
                        center_x, center_y, width, height = map(float, [center_x, center_y, width, height])
                        if not (0 <= center_x <= 1 and 0 <= center_y <= 1 and 0 <= width <= 1 and 0 <= height <= 1):
                            print(f"Error in {label_file} at line {line_number}: Values out of range [0,1]")
                    except ValueError:
                        print(f"Error in {label_file} at line {line_number}: Coordinates are not valid floats")

# Loop through train, valid, and test datasets
for split, paths in datasets.items():
    validate_yolo_labels(paths['images'], paths['labels'], split)
```

Figure 3.9: Validate yolo labels

Description: The purpose of a label validation script is to ensure that the label files are correctly formatted and consistent with the corresponding images. This is crucial to avoid training errors or incorrect model performance.



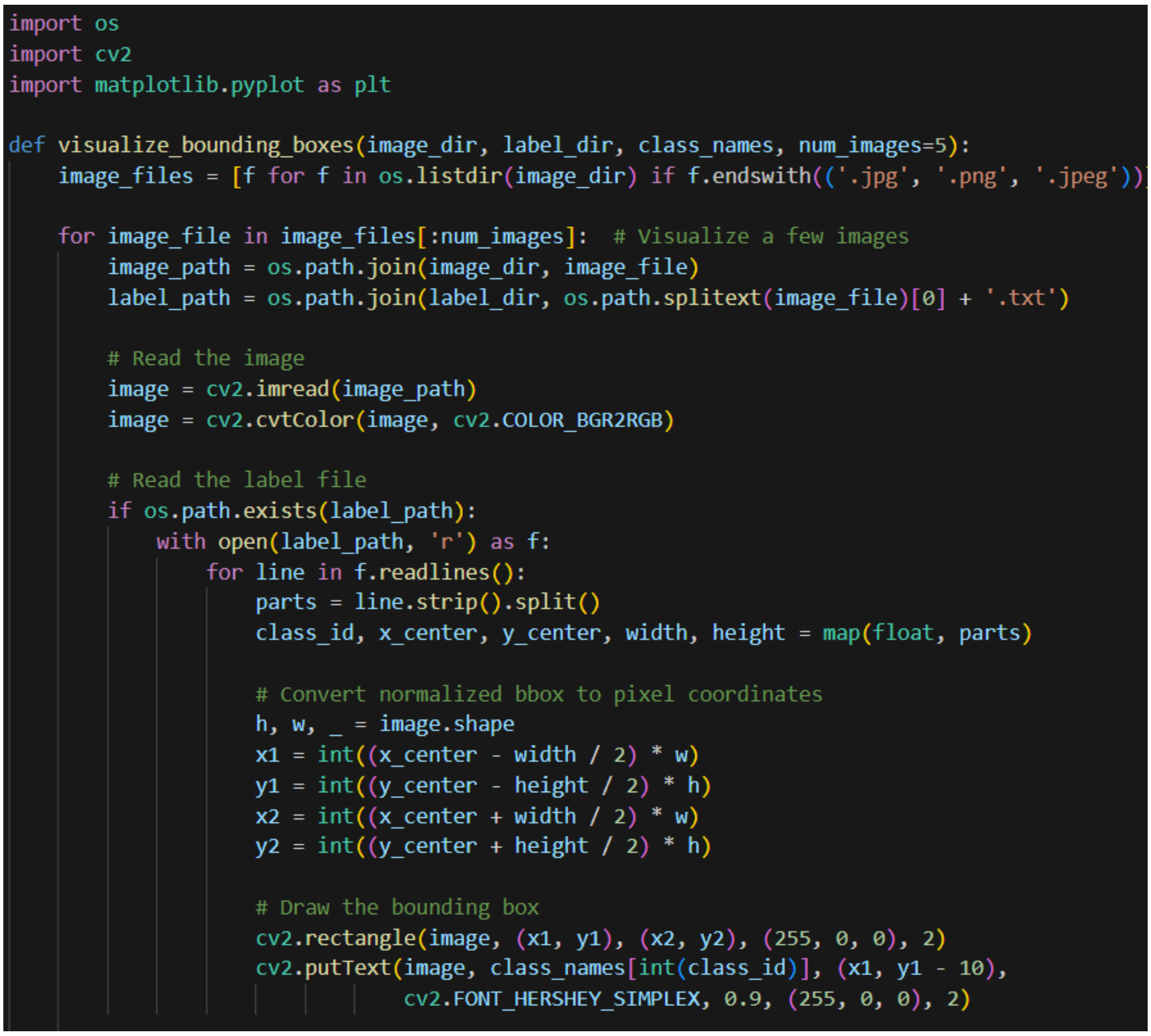
```python
import os

def check_bounding_boxes(label_dir):
    errors = []

    for file in os.listdir(label_dir):
        if file.endswith('.txt'):
            file_path = os.path.join(label_dir, file)
            with open(file_path, 'r') as f:
                for line_num, line in enumerate(f.readlines(), 1):
                    try:
                        parts = line.strip().split()
                        if len(parts) != 5:
                            errors.append(f"{file} (Line {line_num}): Incorrect number of values.")
                            continue

                        class_id, x, y, w, h = map(float, parts)

                        if class_id < 0 or not x >= 0 <= 1 or not y >= 0 <= 1 or not w >= 0 <= 1 or not h >= 0 <= 1:
                            errors.append(f"{file} (Line {line_num}): Bounding box values out of range (0-1).")
                    except ValueError:
                        errors.append(f"{file} (Line {line_num}): Unable to parse values.")

    if errors:
        print("Errors found in bounding boxes:")
        for error in errors:
            print(error)
    else:
        print("All bounding boxes are correctly formatted!")

# Replace with your label directories
train_labels = r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-train\labels"
valid_labels = r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-valid\labels"
test_labels = r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-test\labels"
```



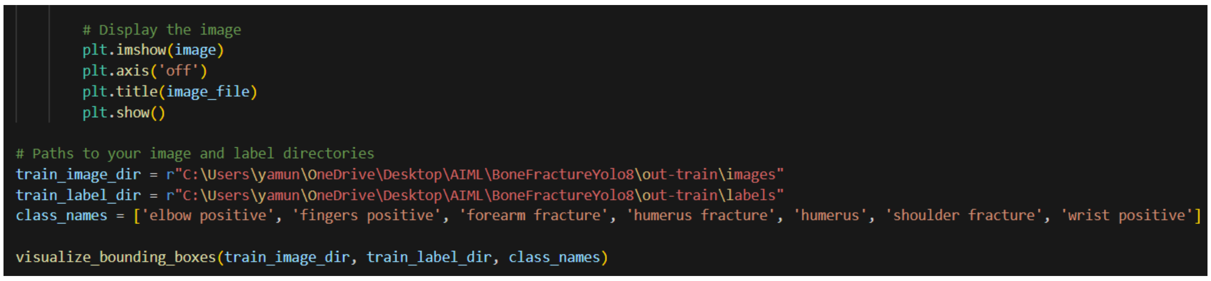
```python
print("Checking Train Labels:")
check_bounding_boxes(train_labels)

print("\nChecking Valid Labels:")
check_bounding_boxes(valid_labels)

print("\nChecking Test Labels:")
check_bounding_boxes(test_labels)
```

Figure 3.10: Check bounding boxes

Description: A bounding box is a rectangular outline used in computer vision and image processing tasks to define the position and size of an object within an image. Bounding boxes are essential for object detection tasks, as they help models locate and classify objects.

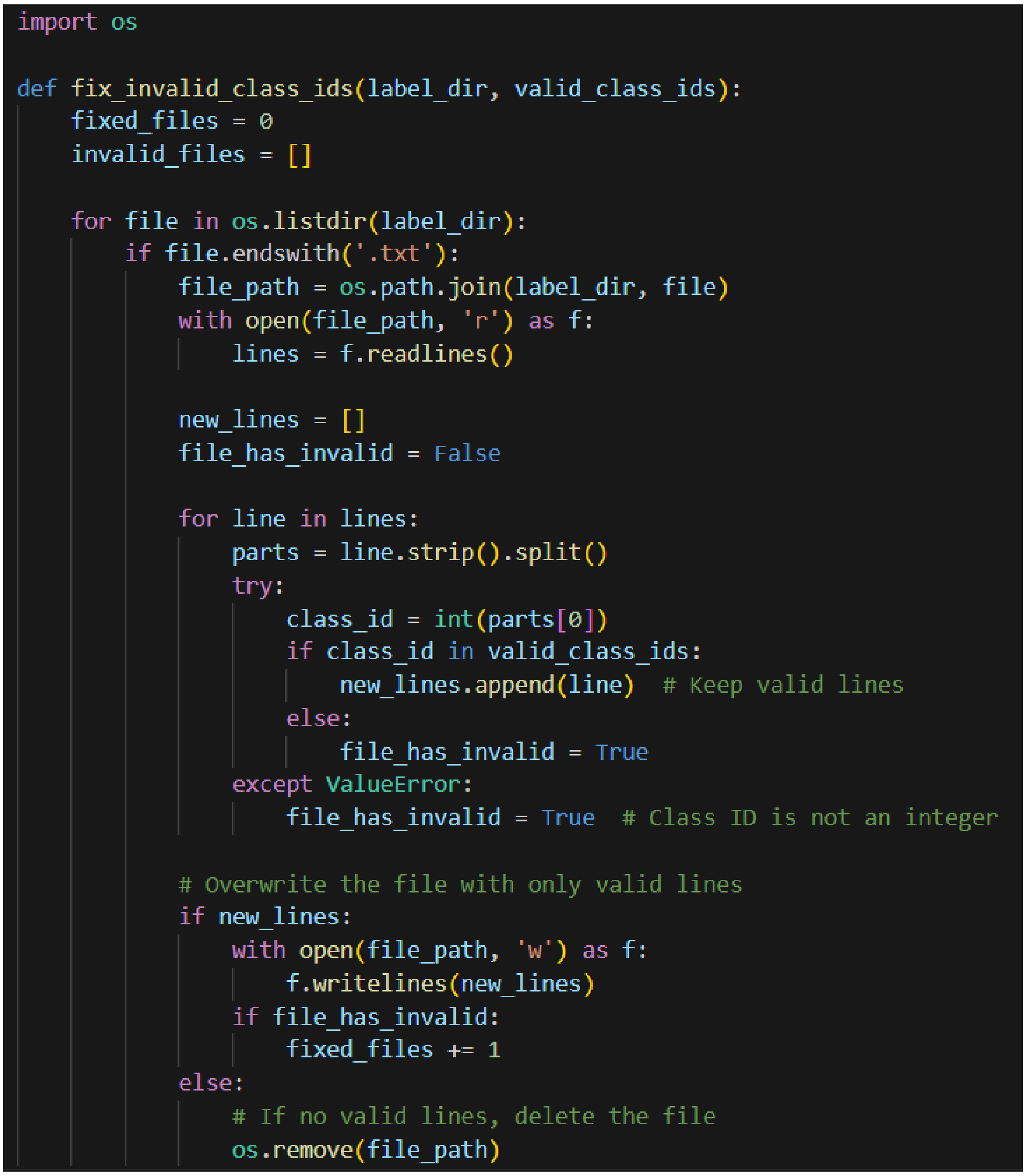


```python
import os
import cv2
import matplotlib.pyplot as plt

def visualize_bounding_boxes(image_dir, label_dir, class_names, num_images=5):
    image_files = [f for f in os.listdir(image_dir) if f.endswith(('.jpg', '.png', '.jpeg'))

    for image_file in image_files[:num_images]:  # Visualize a few images
        image_path = os.path.join(image_dir, image_file)
        label_path = os.path.join(label_dir, os.path.splitext(image_file)[0] + '.txt')

        # Read the image
        image = cv2.imread(image_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        # Read the label file
        if os.path.exists(label_path):
            with open(label_path, 'r') as f:
                for line in f.readlines():
                    parts = line.strip().split()
                    class_id, x_center, y_center, width, height = map(float, parts)

                    # Convert normalized bbox to pixel coordinates
                    h, w, _ = image.shape
                    x1 = int((x_center - width / 2) * w)
                    y1 = int((y_center - height / 2) * h)
                    x2 = int((x_center + width / 2) * w)
                    y2 = int((y_center + height / 2) * h)

                    # Draw the bounding box
                    cv2.rectangle(image, (x1, y1), (x2, y2), (255, 0, 0), 2)
                    cv2.putText(image, class_names[int(class_id)], (x1, y1 - 10),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 0, 0), 2)
```

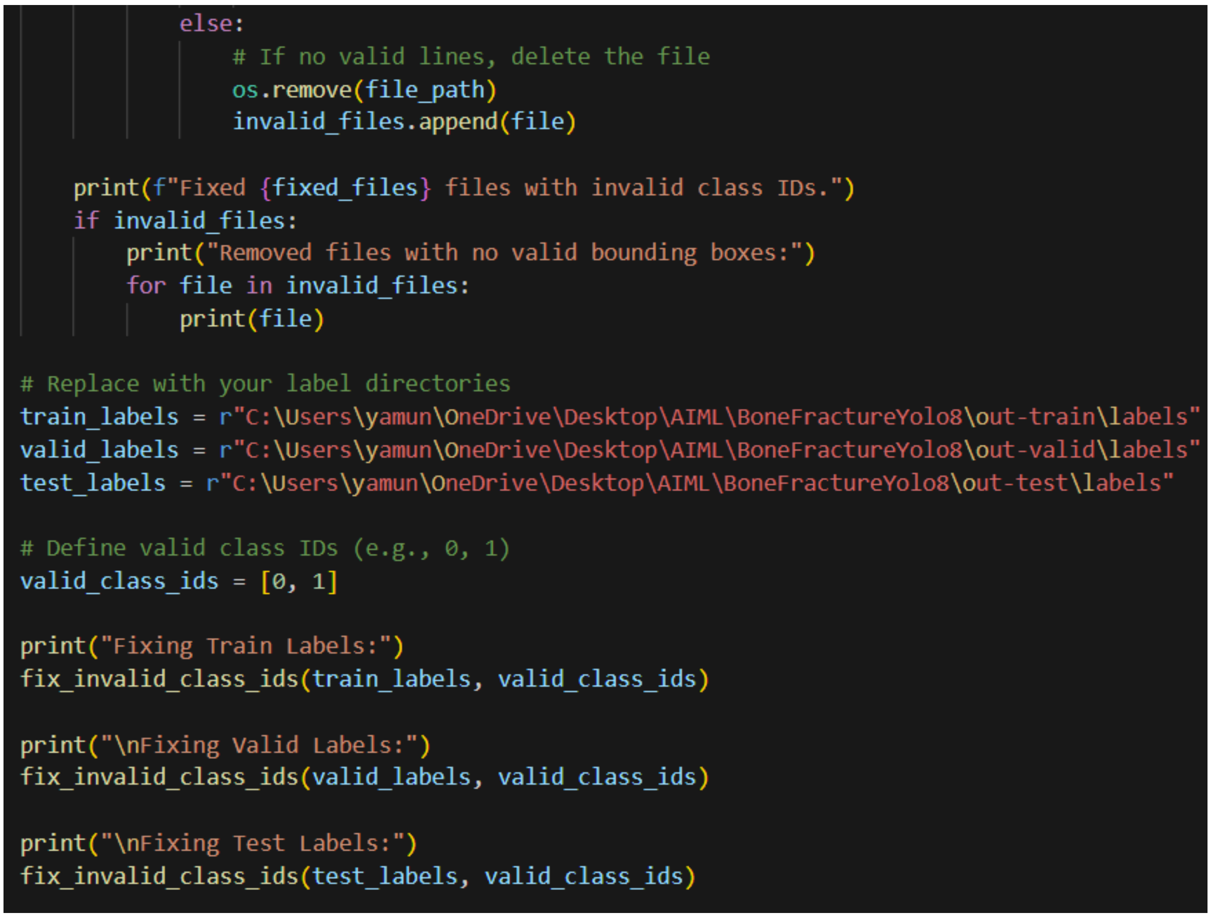


```python
        # Display the image
        plt.imshow(image)
        plt.axis('off')
        plt.title(image_file)
        plt.show()

# Paths to your image and label directories
train_image_dir = r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-train\images"
train_label_dir = r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-train\labels"
class_names = ['elbow positive', 'fingers positive', 'forearm fracture', 'humerus fracture', 'humerus', 'shoulder fracture', 'wrist positive']

visualize_bounding_boxes(train_image_dir, train_label_dir, class_names)
```

Figure 3.11: Visualize bounding boxes



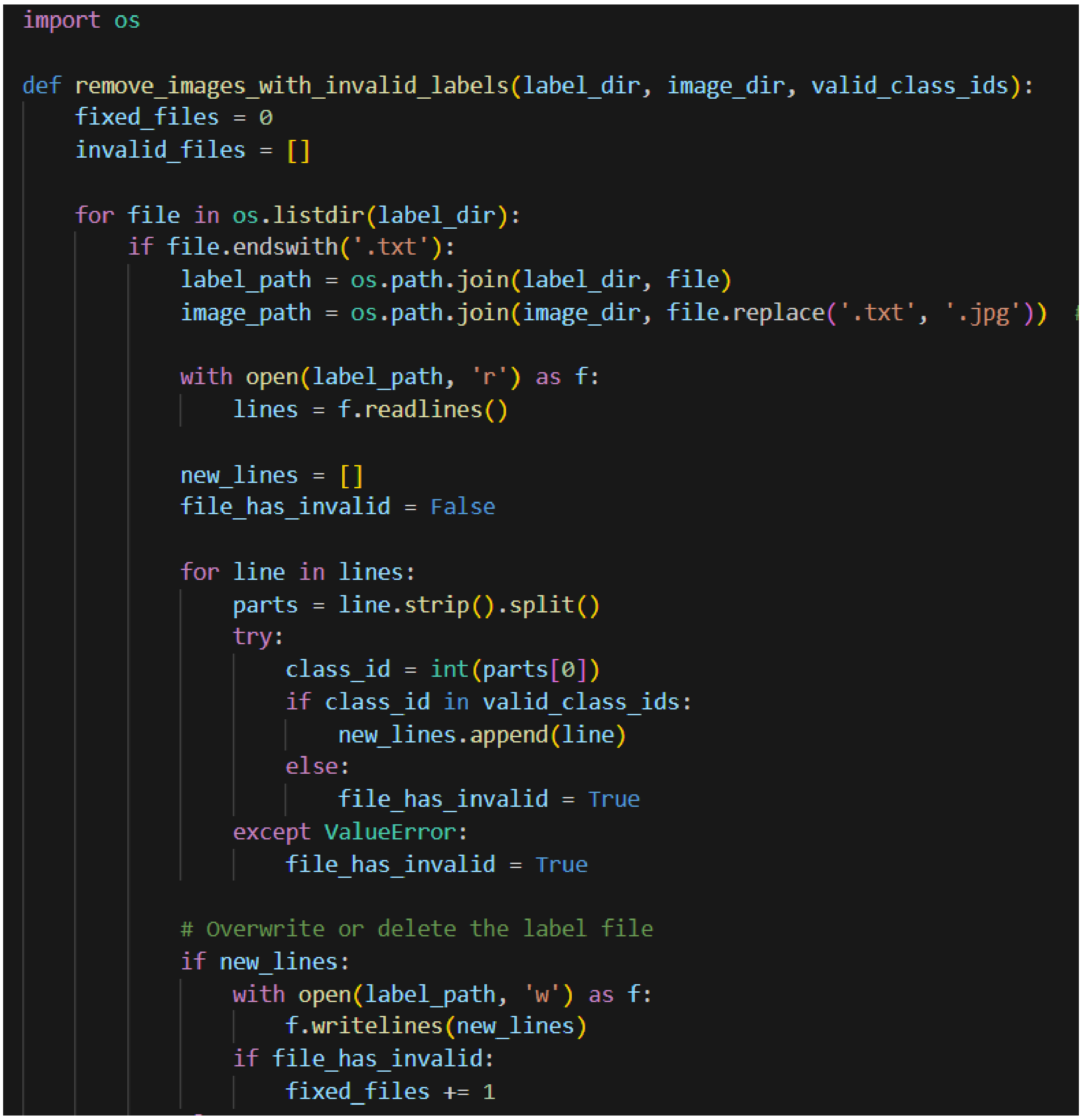
```python
import os

def fix_invalid_class_ids(label_dir, valid_class_ids):
    fixed_files = 0
    invalid_files = []

    for file in os.listdir(label_dir):
        if file.endswith('.txt'):
            file_path = os.path.join(label_dir, file)
            with open(file_path, 'r') as f:
                lines = f.readlines()

            new_lines = []
            file_has_invalid = False

            for line in lines:
                parts = line.strip().split()
                try:
                    class_id = int(parts[0])
                    if class_id in valid_class_ids:
                        new_lines.append(line)  # Keep valid lines
                    else:
                        file_has_invalid = True
                except ValueError:
                    file_has_invalid = True  # Class ID is not an integer

            # Overwrite the file with only valid lines
            if new_lines:
                with open(file_path, 'w') as f:
                    f.writelines(new_lines)
                if file_has_invalid:
                    fixed_files += 1
            else:
                # If no valid lines, delete the file
                os.remove(file_path)
```



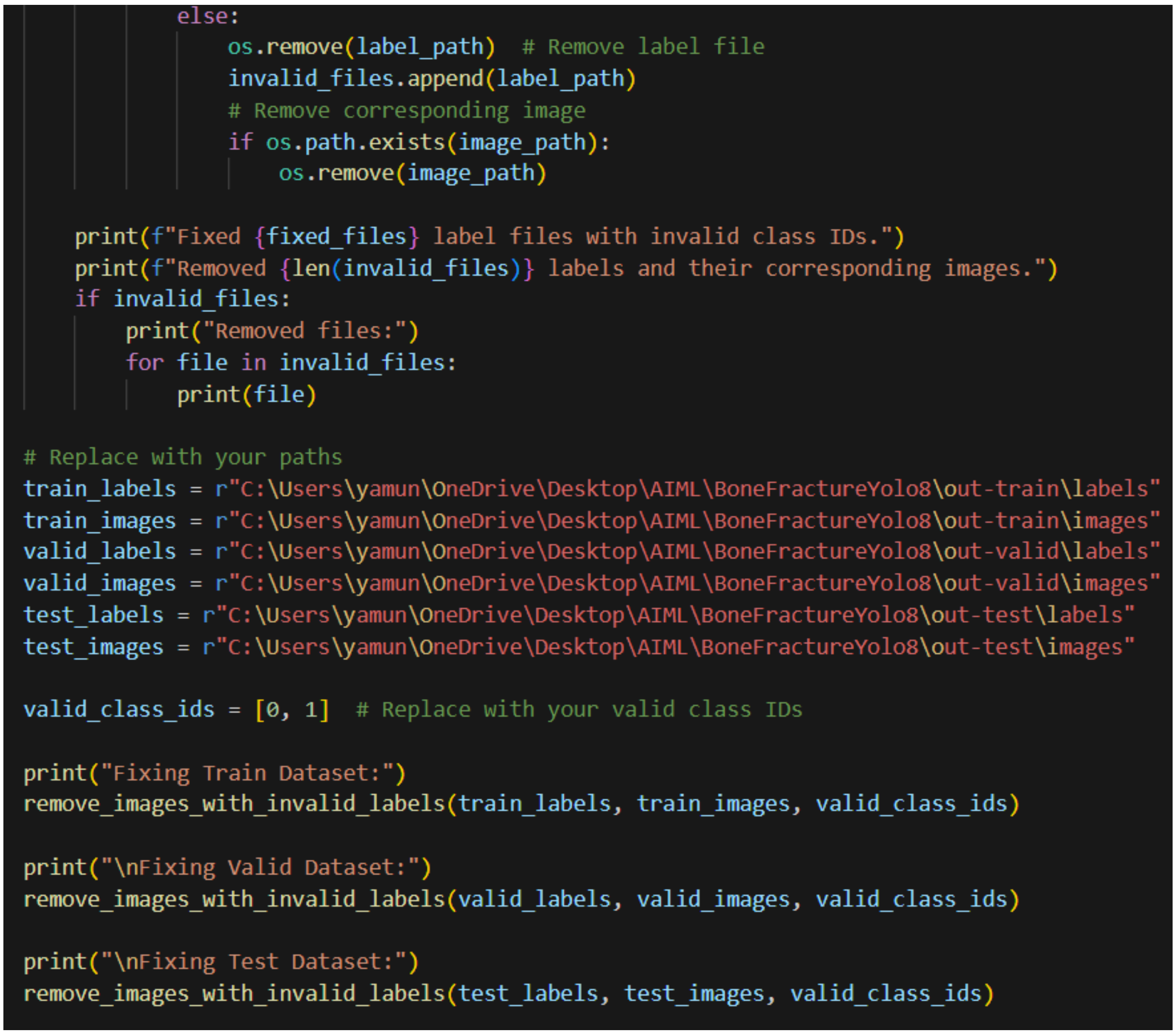
```python
            else:
                # If no valid lines, delete the file
                os.remove(file_path)
                invalid_files.append(file)

    print(f"Fixed {fixed_files} files with invalid class IDs.")
    if invalid_files:
        print("Removed files with no valid bounding boxes:")
        for file in invalid_files:
            print(file)

# Replace with your label directories
train_labels = r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-train\labels"
valid_labels = r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-valid\labels"
test_labels = r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-test\labels"

# Define valid class IDs (e.g., 0, 1)
valid_class_ids = [0, 1]

print("Fixing Train Labels:")
fix_invalid_class_ids(train_labels, valid_class_ids)

print("\nFixing Valid Labels:")
fix_invalid_class_ids(valid_labels, valid_class_ids)

print("\nFixing Test Labels:")
fix_invalid_class_ids(test_labels, valid_class_ids)
```

Figure 3.12: Fix invalid classes

Description: Fixing invalid classes in a dataset involves ensuring that the class_id values in the labels are correct and fall within the range of predefined classes. This is critical for successful training and evaluation in machine learning tasks like object detection.



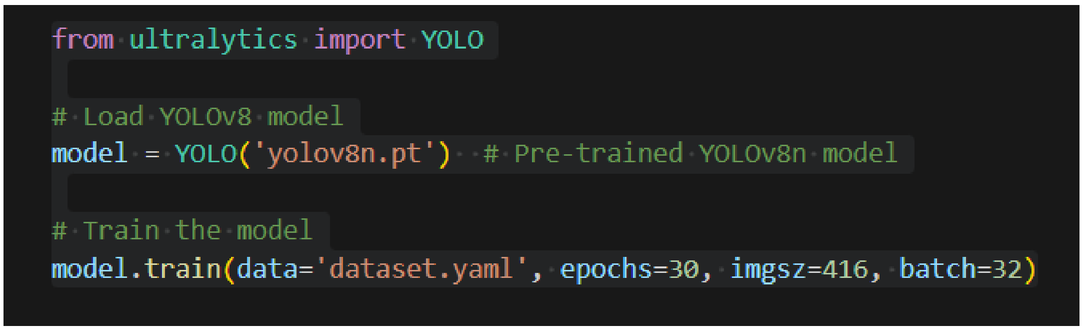
```python
import os

def remove_images_with_invalid_labels(label_dir, image_dir, valid_class_ids):
    fixed_files = 0
    invalid_files = []

    for file in os.listdir(label_dir):
        if file.endswith('.txt'):
            label_path = os.path.join(label_dir, file)
            image_path = os.path.join(image_dir, file.replace('.txt', '.jpg'))  #

            with open(label_path, 'r') as f:
                lines = f.readlines()

            new_lines = []
            file_has_invalid = False

            for line in lines:
                parts = line.strip().split()
                try:
                    class_id = int(parts[0])
                    if class_id in valid_class_ids:
                        new_lines.append(line)
                    else:
                        file_has_invalid = True
                except ValueError:
                    file_has_invalid = True

            # Overwrite or delete the label file
            if new_lines:
                with open(label_path, 'w') as f:
                    f.writelines(new_lines)
                if file_has_invalid:
                    fixed_files += 1
```



```
                else:
                    os.remove(label_path)  # Remove label file
                    invalid_files.append(label_path)
                    # Remove corresponding image
                    if os.path.exists(image_path):
                        os.remove(image_path)

    print(f"Fixed {fixed_files} label files with invalid class IDs.")
    print(f"Removed {len(invalid_files)} labels and their corresponding images.")
    if invalid_files:
        print("Removed files:")
        for file in invalid_files:
            print(file)

# Replace with your paths
train_labels = r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-train\labels"
train_images = r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-train\images"
valid_labels = r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-valid\labels"
valid_images = r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-valid\images"
test_labels = r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-test\labels"
test_images = r"C:\Users\yamun\OneDrive\Desktop\AIML\BoneFractureYolo8\out-test\images"

valid_class_ids = [0, 1]  # Replace with your valid class IDs

print("Fixing Train Dataset:")
remove_images_with_invalid_labels(train_labels, train_images, valid_class_ids)

print("\nFixing Valid Dataset:")
remove_images_with_invalid_labels(valid_labels, valid_images, valid_class_ids)

print("\nFixing Test Dataset:")
remove_images_with_invalid_labels(test_labels, test_images, valid_class_ids)
```

Figure 3.13: Remove images with invalid labels

Description: Removing invalid labels from a dataset involves identifying and discarding labels that do not conform to the required format or fall outside acceptable bounds. Invalid labels can disrupt the training process and lead to poor model performance, so it is essential to address them systematically.



```python
from ultralytics import YOLO

# Load YOLOv8 model
model = YOLO('yolov8n.pt')  # Pre-trained YOLOv8n model

# Train the model
model.train(data='dataset.yaml', epochs=30, imgsz=416, batch=32)
```

Figure 3.14: Training the Model

Description: Training a YOLOv8 model involves preparing data, setting up the YOLOv8 framework, configuring the model, and executing the training process to detect objects in images. YOLOv8, developed by Ultralytics, is a highly efficient and versatile deep learning model for object detection, segmentation, and classification tasks.

RESULTS

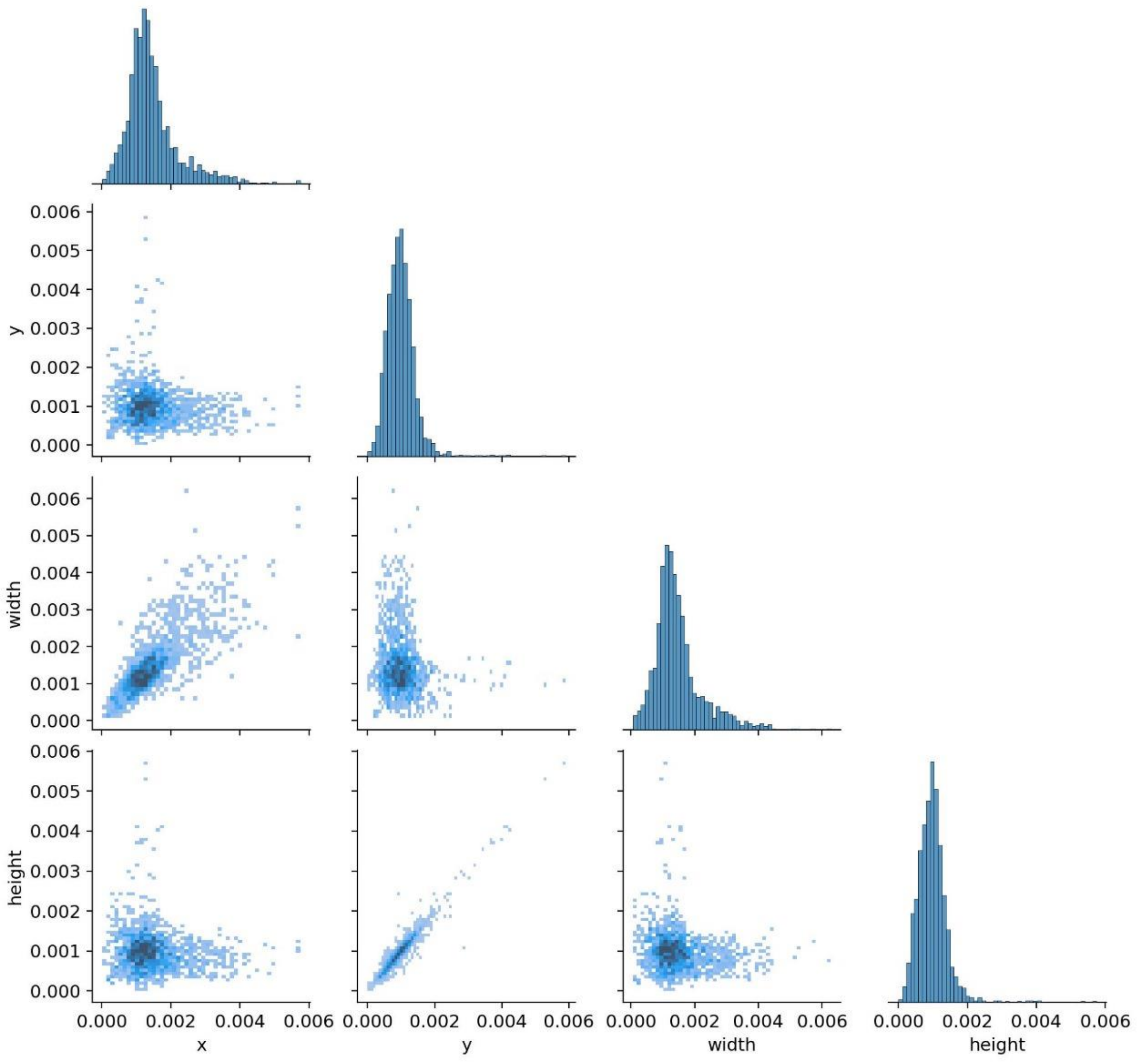


Figure 4.1: Pair Plot

Description: This pair plot visualizes relationships between four variables that are x, y, width, and height. The diagonal shows histograms of individual variables, while the scatter plots display pairwise relationships. Most values are concentrated near zero, with some correlations evident, especially between width and height.

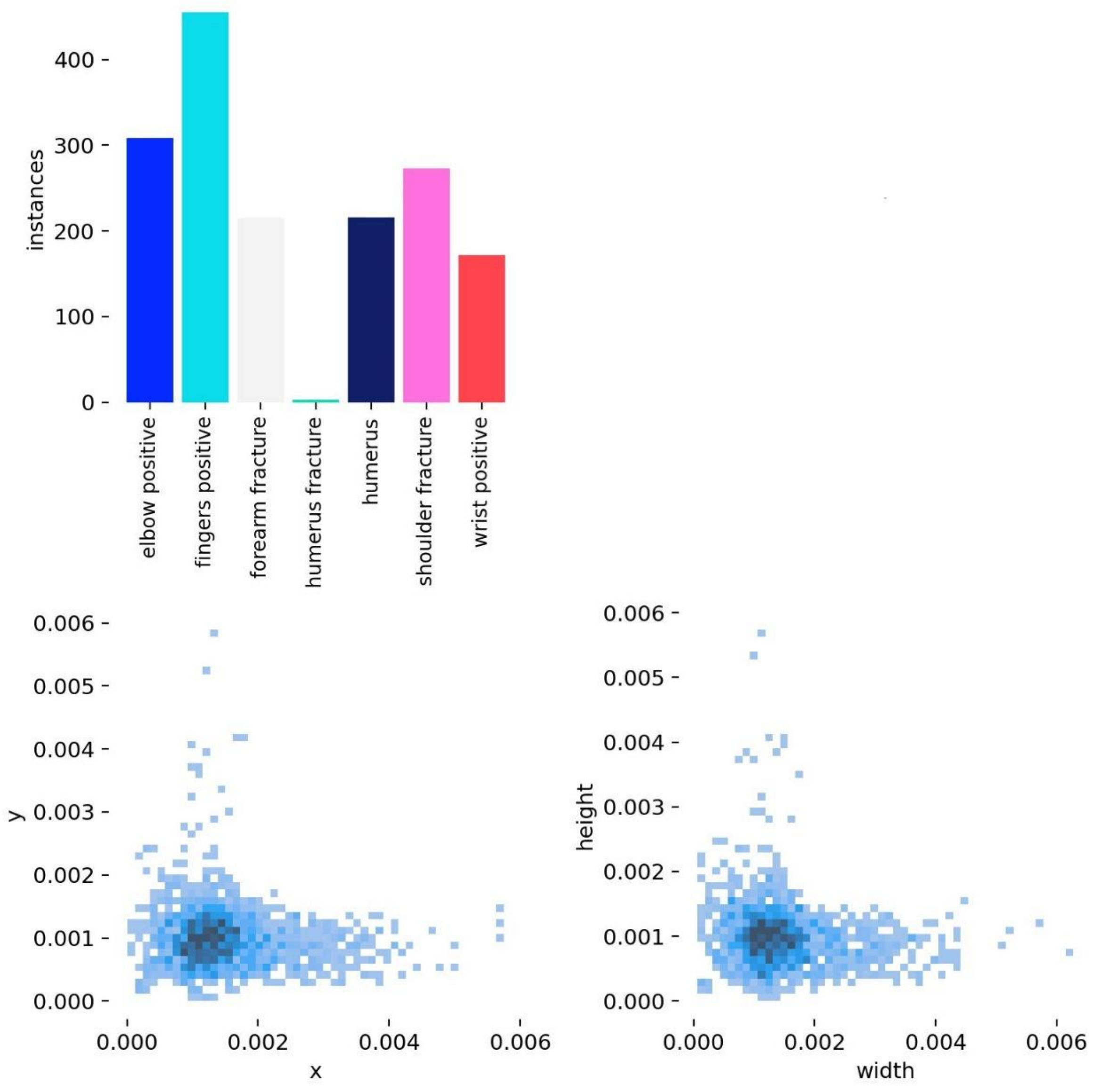


Figure 4.2: Class Distribution (Bar Chart) and Bounding Box Characteristics (Scatter Plot)

Description: This image contains both bar chart and scatter plots. The bar chart at the top shows the count of instances for various classes like "elbow positive," "fingers positive," and others, with "fingers positive" having the highest count. The scatter plots below shows the relationships between x vs. y and width vs. height, showing a dense cluster of values near zero.

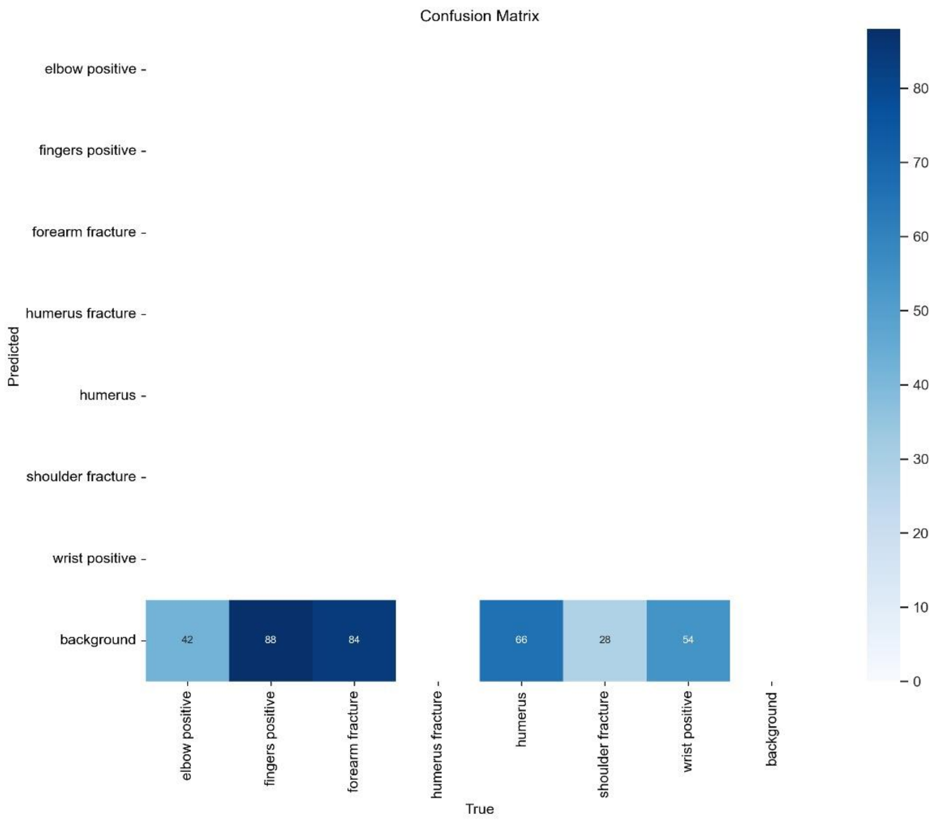


Figure 4.3: Confusion Matrix

Description: This image shows a confusion matrix for a multi-class classification task, illustrating the performance of predictions versus true labels. Each cell indicates the number of instances where a specific class was predicted compared to its actual label, with higher values represented in darker blue shades. The matrix highlights both correct predictions (diagonal entries) and misclassifications (off-diagonal entries).

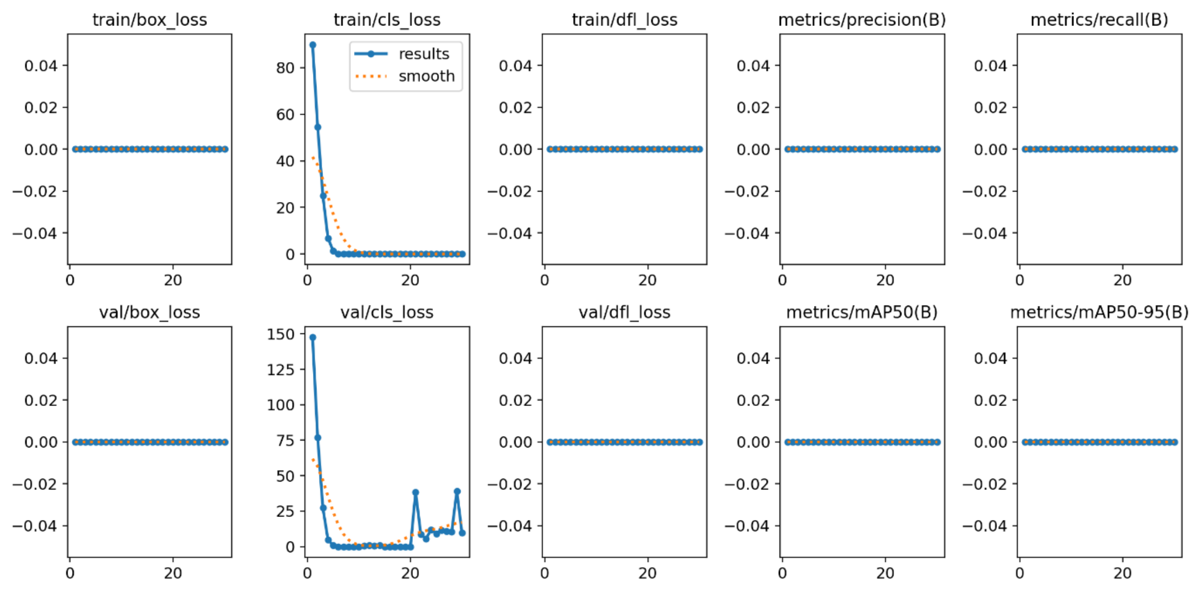
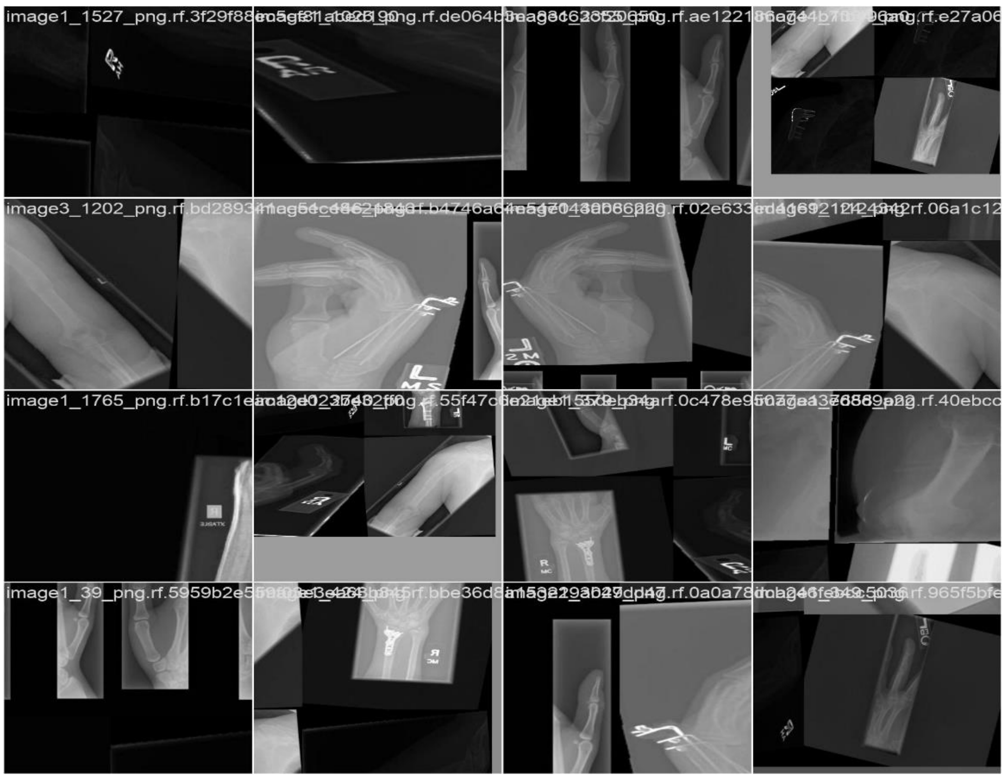


Figure 4.4: Results Graphs

Description: The image shows a grid of training and validation loss and metric plots for a machine learning model. Most metrics, including train/box_loss, train/dfl_loss, metrics/precision(B), and metrics/recall(B), remain constant at zero, suggesting no improvement. The train/cls_loss and val/cls_loss show a decrease over epochs, indicating some optimization in classification loss.



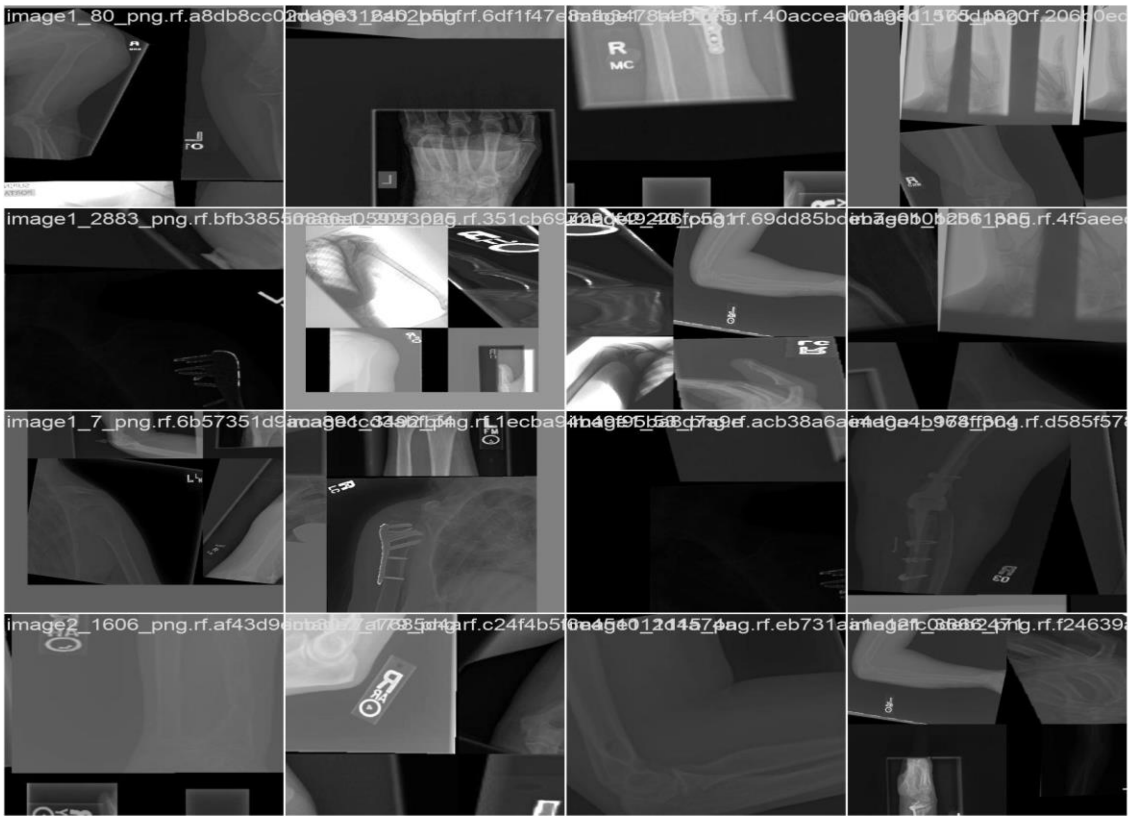
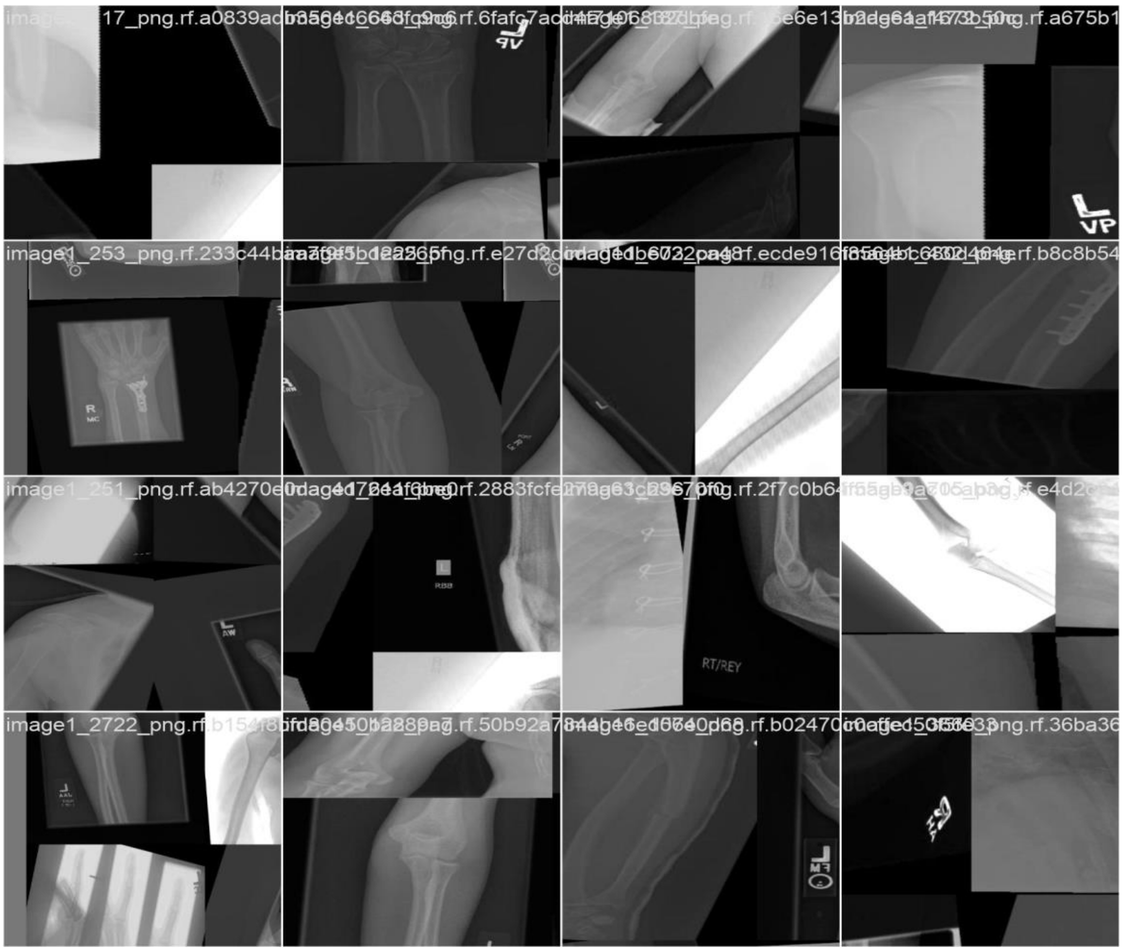
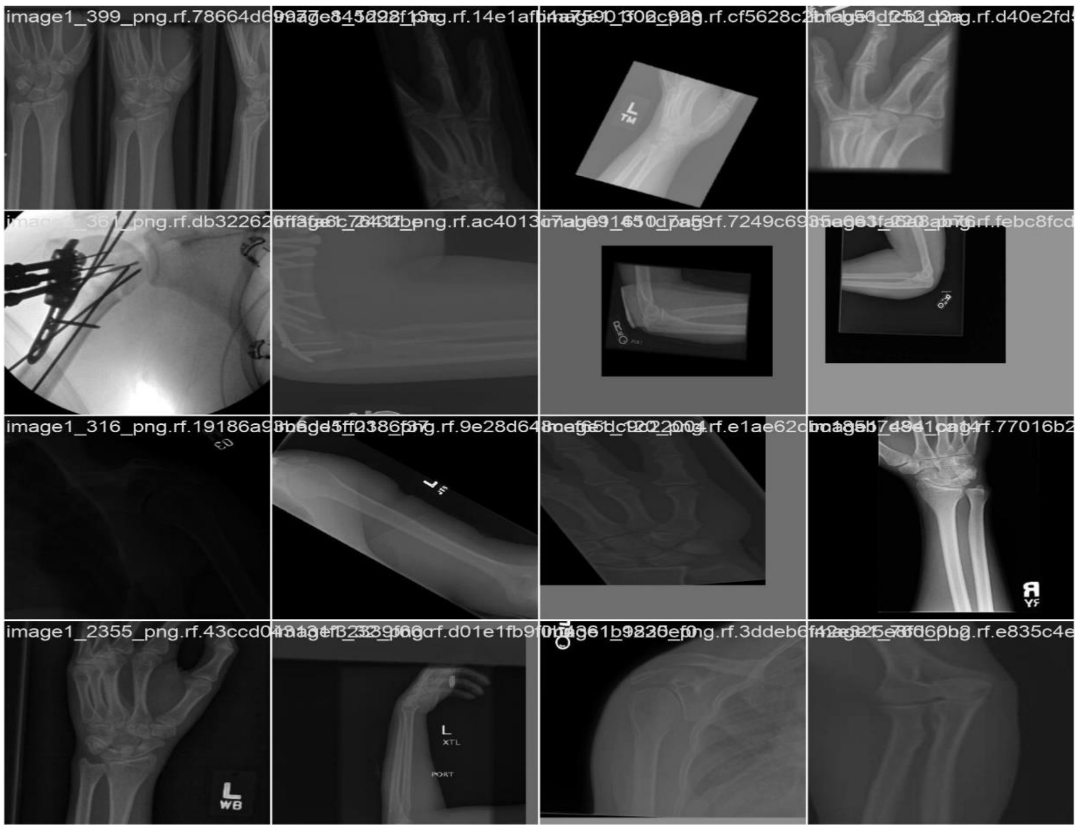


image1_80_png.rf.a8db8cc0
image1_2883_png.rf.bfb385
image1_7_png.rf.6b57351d9
image2_1606_png.rf.af43d9


17_png.rf.a0839ad
image1_253_png.rf.233c44b
image1_251_png.rf.ab4270e
image1_2722_png.rf



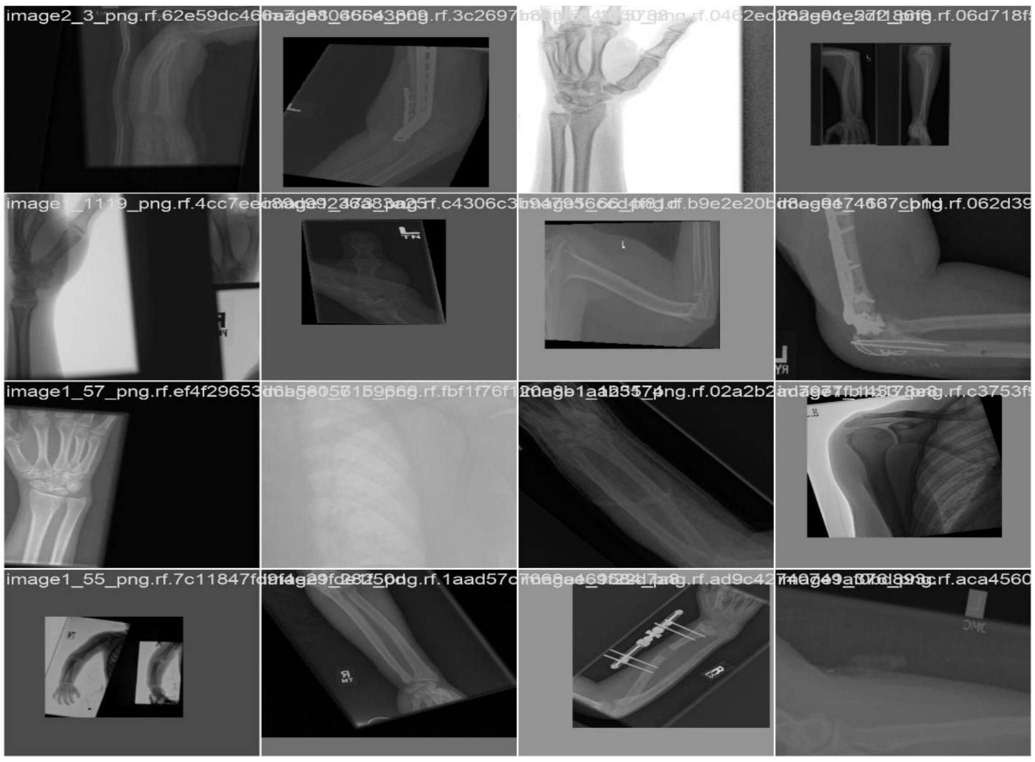


image2_3_png.rf.62e59dc4
image1_57_png.rf.ef4f29653
image1_55_png.rf.7c11847f

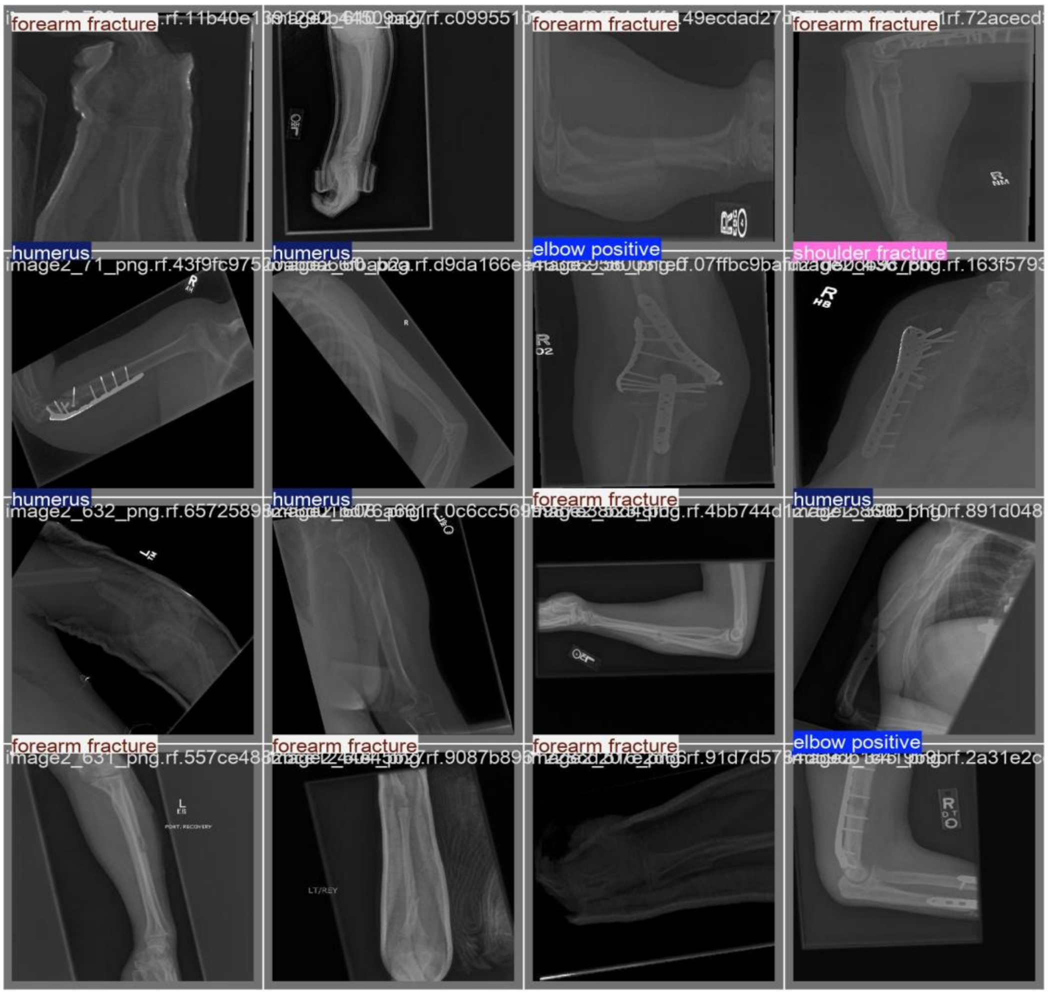
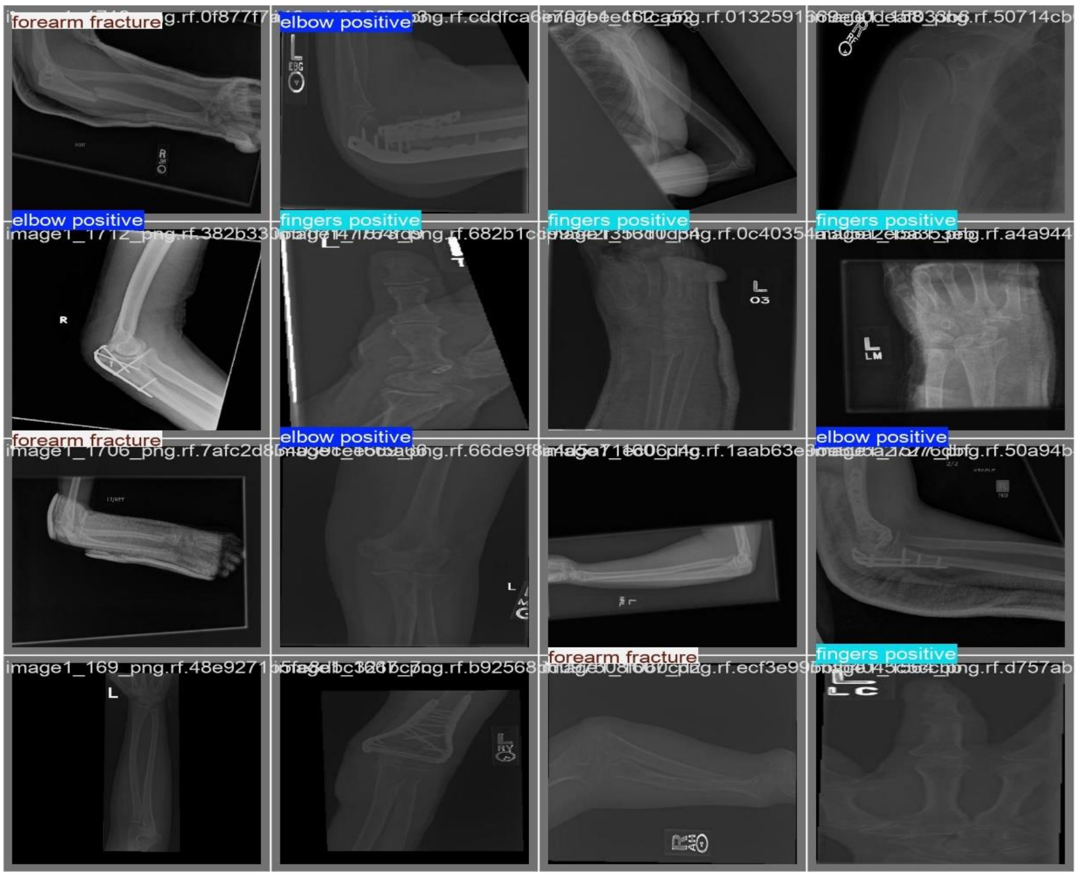

forearm fracture
forearm fracture
forearm fracture
humerus
humerus
elbow positive
shoulder fracture
image2_71_png.rf.43f9fc975
humerus
humerus
forearm fracture
humerus
image2_632_png.rf.6572589
forearm fracture
forearm fracture
forearm fracture
elbow positive
image2_631_png.rf.557ce48

image2_720_png.rf.11b40e1
image2_71_png.rf.43f9fc975
image2_632_png.rf.65725898
image2_631_png.rf.557ce483


forearm fracture
elbow positive
elbow positive
fingers positive
fingers positive
fingers positive
image1_1712_png.rf.382b330
forearm fracture
elbow positive
elbow positive
image1_1706_png.rf.7afc2d8
forearm fracture
fingers positive
image1_169_png.rf.48e9271

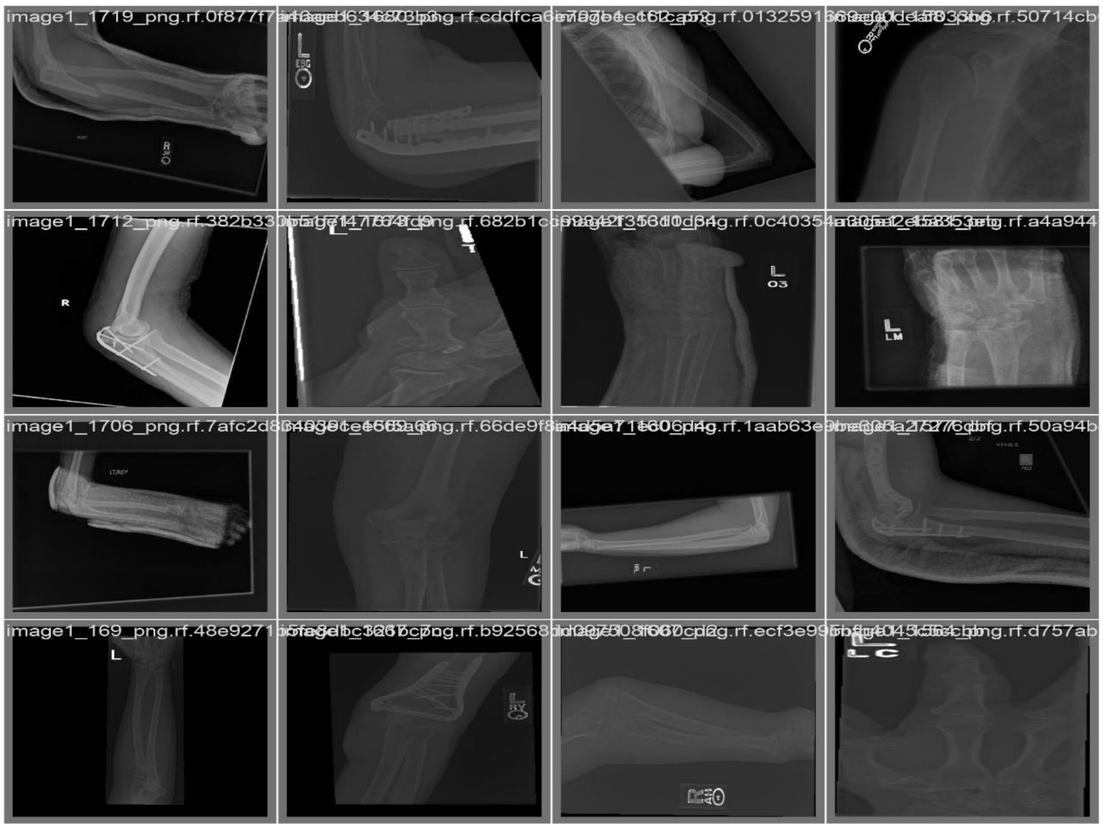
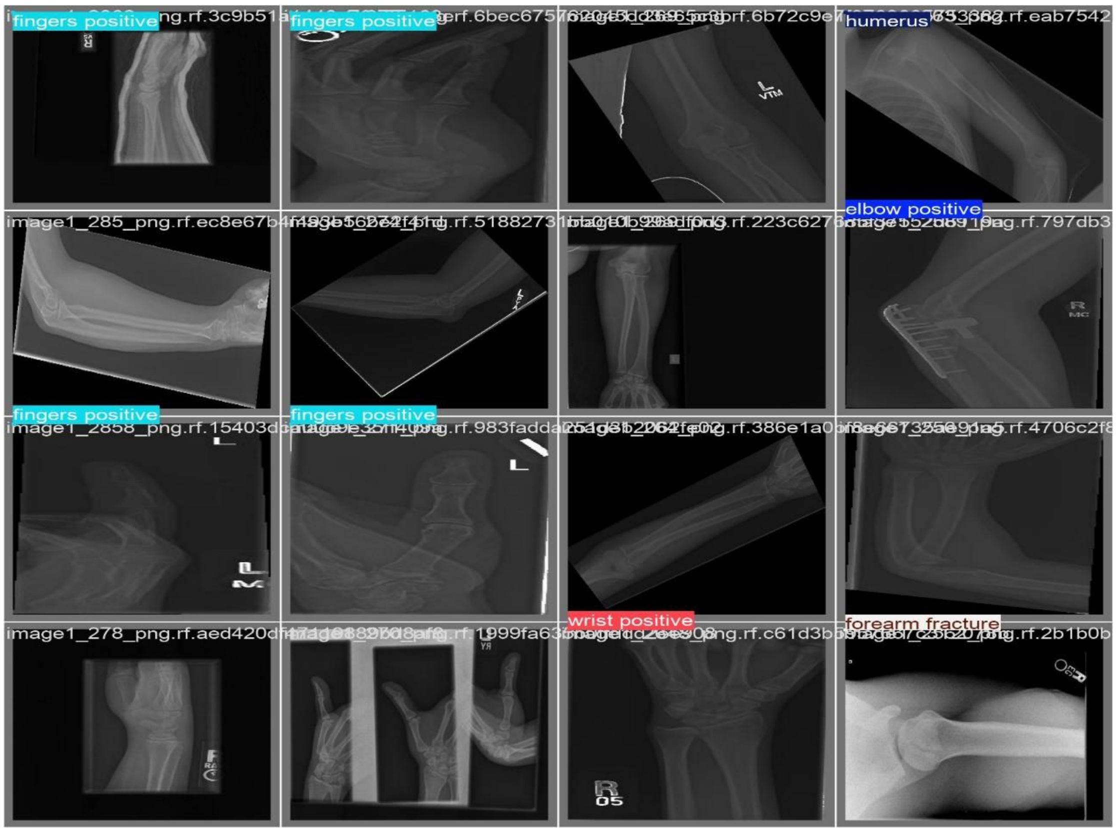


image1_1719_png.rf.0f877f7
image1_1712_png.rf.382b33
image1_1706_png.rf.7afc2d
image1_169_png.rf.48e9271


fingers positive
fingers positive
humerus
image1_285_png.rf.ec8e67b
elbow positive
fingers positive
fingers positive
image1_2858_png.rf.15403d
wrist positive
forearm fracture
image1_278_png.rf.aed420d

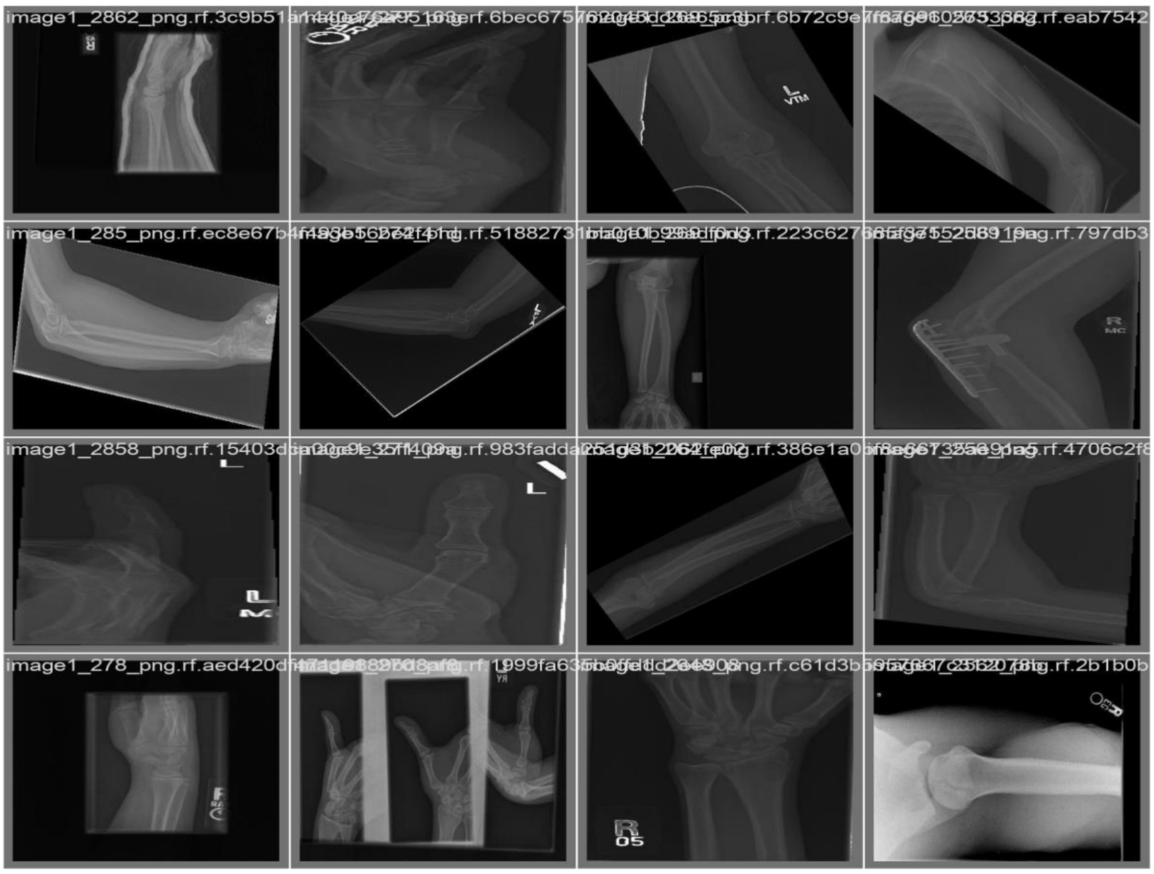
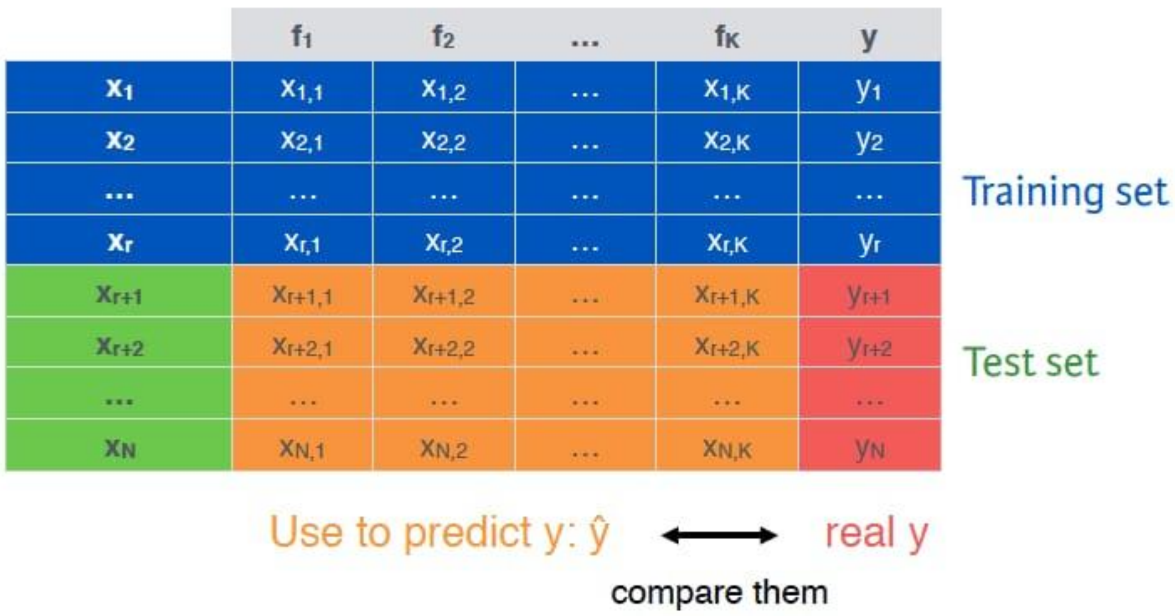


Figure 4.5: Batch Images

Description: The above images are classified as batches indicating all types of bones in each part, showcasing different angles and regions of bones, likely from limbs. Each tile is labeled with filenames or unique identifiers, indicating that these images might be part of a dataset for medical analysis or machine learning tasks. The X-rays vary in perspective, showing both close-up and zoomed-out views of bones.



	f_1	f_2	…	f_K	y
x_1	$x_{1,1}$	$x_{1,2}$	…	$x_{1,K}$	y_1
x_2	$x_{2,1}$	$x_{2,2}$	…	$x_{2,K}$	y_2
…	…	…	…	…	…
x_r	$x_{r,1}$	$x_{r,2}$	…	$x_{r,K}$	y_r
x_{r+1}	$x_{r+1,1}$	$x_{r+1,2}$	…	$x_{r+1,K}$	y_{r+1}
x_{r+2}	$x_{r+2,1}$	$x_{r+2,2}$	…	$x_{r+2,K}$	y_{r+2}
…	…	…	…	…	…
x_N	$x_{N,1}$	$x_{N,2}$	…	$x_{N,K}$	y_N

Table 1: Dataset distribution for training and testing.

ML Model	Hyper-parameter
RF Classifier	n_estimators
	max_depth
	min_samples_split
	min_samples_leaf
	criterion
	max_features
SVM Classifier	C
	kernel
KNN Classifier	n_neighbors
RF Regressor	n_estimators
	max_depth
	min_samples_split
	min_samples_leaf
	criterion
	max_features
SVM Regressor	C
	kernel
	epsilon
KNN Regressor	n_neighbors

Table 2: Hyperparameters used for training and the model.

CONCLUSION

The main goal of this project is to create a system that makes the work of the doctor faster and easier to figure out if a patient has a broken bone. Using machine learning, we're fr

This system is designed to analyse X-ray scans and tell whether a bone is broken or not, or whether it has crack or not. It goes a step further by identifying the type and severity of the fracture. The X-rays used for training the system include all the sorts of fractures like simple, complex and even tiny hairline cracks along with healthy bones for comparison. This way, the model is prepared to handle a variety of cases, making it reliable for real-world use.

Bone fractures are becoming round the world more due to things like aging populations, urban lifestyles and rising accident rates. Spotting even small fractures is crucial because a missed or delayed diagnosis can lead to serious complications or slower recovery times. This system is designed to help doctors and radiologists detect fractures quickly and accurately, saving valuable time and improving patient care.

What makes this system especially helpful is that it doesn't just provide a yes or no answer. It also highlights the part of the X-ray it focused on, so doctors can see exactly why the system is a valuable tool rather than a "black box".

This technology has the potential to make a big difference in healthcare. By reducing the workload on radiologists and cutting down on diagnostic errors, it can help doctors focus more on treatment. It's also particularly useful in places where there aren't enough experienced radiologists, making it a lifesaver in resource limited areas.

The system is designed to work seamlessly with the existing model hospital technology, like PACS (Picture Archiving and Communication Systems), and it's useful and built to be fast and user friendly. In short, this project aims to transform how fractures are detected. By combining smart technology with real world, it's a toward better, faster and more reliable healthcare for patients everywhere. Doctors or technicians can upload an X-ray and get results almost instantly, which means patients get the care they need without unnecessary delays.

REFERENCES

Enhancing diagnosis: ensemble deep-learning model for fracture detection using X-ray https://www.sciencedirect.com/science/article/pii/S0009926024004197

RESEARCH PAPERS:

1. Achawale, M. A. More, Y. Bankar, M. A. Ghuge. M. P and Zole, M. G, 2024 BONE FRACTURE DETECTION USING CONVOLUTIONAL NEURAL NETWORK (CNN). JOURNAL OF BASIC SCIENCE AND ENGINEERING.
2. Alkhatib, A. J, Alharoun. M and AlZoubi, A, 2024. A Deep Learning Framework for timely bone fracture detection and prevention. Information Sciences with Applications.
3. Bone fracture detection and classification using Deep Learning Techniques. In Driving and Smart Medical Diagnosis Through AI-Powered Technologies (AI) APPROACHES. In AIP Conference Proceedings (Vol. 2802. No, 1). AIP Publishing.
4. Khan. M. I, 2024 Bone fracture identification with using deep learning model Resnet50. International Journal of computing and digital systems.
5. Devi. M. S, Aruna. R, Alufti. S, Punitha. P and Kumar, R. L, Bone fracture quantization and systemized attention gate. UNet-based deep learning framework for bone fracture classification Intelligent Data Analysis.
6. M. Singh, S. Kumar, Ruchilekha and M. K. Singh,
7. M. Singh, S. Kumar, Ruchilekha and M. K. Singh, “A Comparative Study of Feature Extraction Techniques and Similarity Measures for image Retrieval,” 2022 International Conference on Futuristic Technologies (INCOFT), Belgaum, India, 2022, pp. 1-7, doi:10,1109/INCOFT55651.2002.10094430.

8. Rinisha Bagaria, Sulochana Wadgwani, Arun Kumar Wadhwani, Bone fractures detection using support vector machine and error backpropagation neural network, Optik, Volume 247,2021,168021,ISSN 0030-4026

- Detection of bone fracture based on machine learning techniques: Kosrat Dlshad Ahmed, Roojwan Hawezi I images: Information System Engineering Department, Erbil Polytechnic University, Erbil, Iraq.
- Analysis of X-Ray Images with Image Processing Techniques: A Review
- Fracture detection from X-ray images using different Machine Learning Techniques.

YouTube links:

https://youtu.be/9qCWNsdazn4?si=54s9NTFre7au0q9e

https://youtu.be/UdTGPXUrHx8?si=cFQObdbpQ7ORNced

https://youtu.be/aQHxCjb58QU?si=mP3g9FKrp4FRsvlk

https://youtu.be/XJpC8VI7Zio?si=Z7ylGDd23hhemLEH