Why this topic was chosen?

Fracture of bone is one of the usual medical conditions across the globe. The annual incidence of fractures is most significantly affected by sports injuries, osteoporotic bone fragility due to aging, road accidents and work-related hazards. So, while today's hospitals may be equipped with digital imaging technology — CT, MRI and X-ray machines — the human visual inspection measures by radiologists are still an important part of fracture diagnosis. This manual process is time-consuming and susceptible to human error, especially when the fracture is small, minimally displaced or associated with surrounding bone.

Radiology specialists are not always available at many rural and semi-urban hospitals, which leads to slow diagnosis. These delays may result in permanent hampering, internal haemorrhage, and malformation of the bones when dealing with an emergency such as a trauma case. The medical system can therefore benefit immensely by coming up with a smart medical imaging system which is able to identify fractures automatically. The conventional image processing methods such as edge detection, thresholding and morphological process have been used to identify the fractures before. Nevertheless, in cases of change in X-ray quality or very small fractures, these methods cannot generalize sufficiently. Due to this problem, Conventional Neural Networks (CNNs) and Deep Learning (DL) have transformed the process of detecting fractures.

Automated Bone Fracture Detection Using Image Processing, Deep Learning, and Ablation-Based Optimization

Abstract

It has been a long-standing challenge for the field of medical imaging to detect fractures of the bones using X-rays due to the varying characteristics of the bone's microstructure, differences in the overall quality of the images, and the complex characteristics of fractures. Therefore, this study aims to develop an approach to automatically detect fractures using deep learning and image processing techniques that will increase the accuracy of diagnosis. The study includes a systematic approach that consists of uploading and assessing publicly available medical image datasets to provide the datasets used to test and refine deep learning algorithms, as well as conducting image preprocessing on the uploaded images to improve classification. Preprocessing techniques include applying denoising filters, equalizing the histogram distribution, scaling pixels to the same intensity range, and enhancing contrast. Then, models are created using convolutional neural networks (CNNs) and transfer learning using ResNet-50 to develop a baseline for future classifications. One objective of this study is to identify gaps in the current literature/research and implementations, such as the limitations of the variety of datasets from which medical images have been evaluated, the lack of small fracture detection, and the limited interpretability of existing models.

Deep-learning pipelines can be used to enhance the detection of fractures by repositioning fractures in the body as part of the image augmentation pipeline through the use of ResNeXt-101 as a base architecture and FPN modules. In addition to improving the fracture detection rate, recall, and localization accuracy of a model by a significant amount, the most helpful areas for further improvement based on an ablation study are preprocessing processes and Feature Pyramid Networks for minimum loss. The contribution of a combination of image processing techniques, including Deep Learning Models, will create a robust and reliable system to integrate effectively into any clinical workflow in terms of practical outputs. Future enhancements will include mobile deployment, multimodal AI integration, and real-time diagnostic capabilities.

Structure of YoloV8

YOLO (You Only Look Once), one of the most widely used modules for real-time object identification and picture segmentation, will be regarded as a state-of-the-art (SOTA) method. YOLO is a convolutional neural network that is incredibly quick in predicting bounding boxes and class probabilities for objects in an image with just one forward pass. Despite its great effectiveness, it's important to keep in mind that YOLO was initially intended for widespread, all-purpose use. As a result, highly specialized tasks frequently need for a better comprehension of YOLO's core architecture, particularly those involving non-standard image formats, higher accuracy requirements, or atypical imaging settings. Customizing the model becomes crucial in these situations.

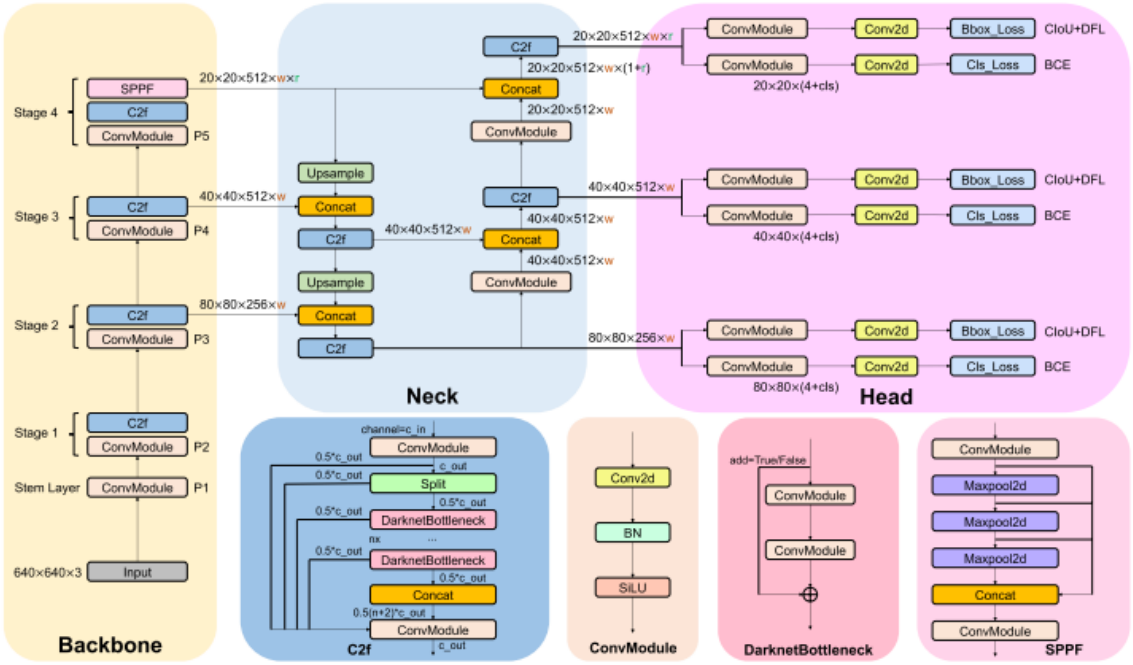


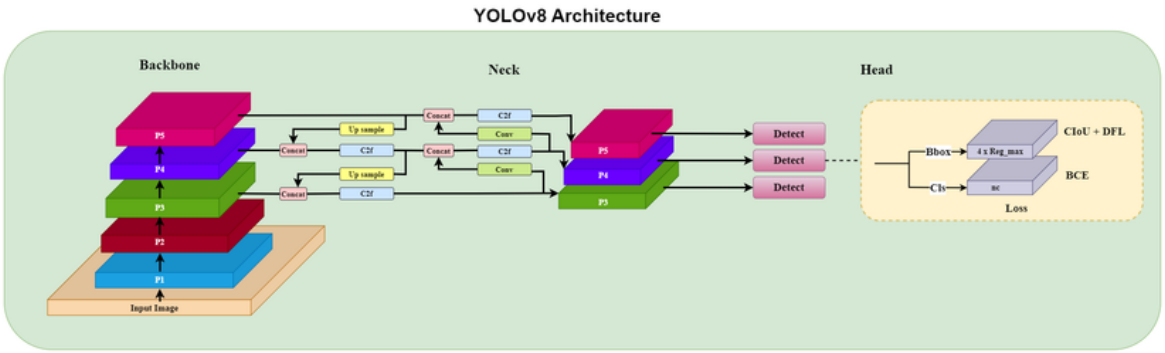
Figure 4. Detailed illustration of YOLOv8 model architecture. The Backbone, Neck, and Head are the three parts of our model, and C2f, ConvModule, DarknetBottleneck, and SPPF are modules.

This essay is the first in a series that focuses on the architecture of YOLOv8 to assist computer vision engineers who want to go deeper into YOLO. Recognizing that the algorithm is divided into three main blocks—the Backbone, Neck, and Head—is the first step towards comprehending how YOLO works within. Each of these has a unique and crucial function in YOLO's image processing.

The backbone

Function: Finds significant patterns in the input image by acting as a feature extractor.

Activities: increasingly captures higher-level characteristics at deeper layers after detecting edges, textures, and fundamental structures in earlier layers. It generates a complex, multi-scale hierarchical representation that serves as the basis for the detection pipeline's later phases.



Neck

Function: Combines feature maps from many layers to act as a link between the Head and the Backbone.

Activities: To detect objects of different sizes, the Neck gathers and combines outputs from the Backbone to execute multi-scale feature fusion. By assisting the model in comprehending its surroundings, it incorporates contextual information to increase detection accuracy. The Neck also speeds up calculation by reducing spatial dimensions, which is advantageous for real-time applications but may have a minor negative impact on detection quality.

Head

Function: The real detection outputs are generated by this last network block.

Activities: The Head creates bounding boxes around possible items, classifies detected objects into categories, and assigns confidence scores that indicate the possibility of an object's presence. In the end, it converts analysed information into predictions that can be put into practice.

Paper analysis

Motivation and problem statement

This research identifies one of the greatest challenges faced by radiologists in treating injured children: accurately and quickly identifying the location of children's wrist fractures.

The incidence of wrist fracture among paediatric patients is very high and occurs with greater frequency than wrist fractures in adults for numerous reasons. First, there is a large percentage of misdiagnosis with paediatric wrist fractures; some studies have shown a 26% error rate in the misdiagnosis of paediatric distal radius/ulna injuries.

Second, there are far fewer qualified radiologists to assist you in hospitals with limited access to all the necessary resources. Paediatric fractures tend to be less severe because of the fact that their bones are not completely developed.

Lastly, the quality of the images produced by x-ray equipment varies from hospital to hospital, and therefore, there is a profound need for hospitals that do not have the ability to provide access to qualified radiologists to have access to automated, artificial intelligence (AI) developed fracture detection systems.

The goal of this research project is to create a state-of-the-art YOLOv8 fracture detection system and demonstrate its potential via a fully functional GUI.

Methodology Overview

The authors modified YOLOv8 to accommodate medical imaging.

Critical Steps:

1. Initial training was performed using pre-trained YOLOv8 models.
    - Input Sizes: 640, 1024.
    - Tested Optimizers: Adam and SGD
2. Domain-Specific Augmentation of the Model
    - Contrast and Brightness Adjustments via the addWeighted() function in OpenCV
    - Parameters: $\gamma = 30$ and $\alpha = 1.2$.

- This created an additional 14,204 training images (total = 28,408 training images).

3. Improvements to the YOLOv8 Architecture
    - C2f Module (improved gradient flow)
    - FP-PAN for Multi-Scale Feature Merging
    - Decoupled Head (combining regression and classification)
    - Bounding Box Prediction without Anchors
4. Selecting the Appropriate Optimizer based on Dataset Size
    - For Small Datasets: Adam
    - For Larger Datasets (after augmentation): SGD performs better.
5. Ablation Studies
    - Performance Comparison (with vs. without enhancements) measured using mAP50 and mAP50–95.

Research gap

1. Small and/or subtle fractures (e.g., mild edema) are not well recognized by typical YOLOv8 models, which have very poor performance in the category of minor bone fracturs with an mAP50 score of only 0.110, therefore making it difficult for deep learning networks (DL) to identify these fractures in paediatric X-rays.

    ML/DL Gap:

    - Deep learning models do not learn to identify micro-features of low-contrast bone in paediatric patients.
    - The ability to discern medical nuances (i.e., subtle bone changes) does not translate from feature extractors developed from large COCO Pretrained Datasets.

2. Severe Class Imbalance Resulting in Bias

While there are nine different classes within the dataset, some of these classes (e.g., home/base and bone lesion) are extremely underrepresented. Due to the imbalance in examples for each class, YOLOv8 is biased toward identifying fractures (the most common example) while more than likely ignoring the identification of minority classes.

- ML/DL Gap:

- Models trained on datasets with imbalanced distributions will create skewed decision boundaries.
- Loss functions that under-represent uncommon medical conditions will not be able to assist in the proper training of models.

3. The absence of domain-specific preprocessing in Baseline Deep Learning (DL) models

   X-ray images can have a number of issues, including variations in brightness, noise, and contrast. The amount that Baseline YOLOv8 models generalize (and thus perform) in practise is much lower without the use of image augmentation techniques.

   ML/DL gap:

   - The features that radiologists or medical imaging people would use do not always produce a good result when applying generic deep learning models.
   - To obtain acceptable model performance, it is necessary to preprocess the data to fit the domain of the X-ray or radiology.

4. YOLOv8 Backbone Pre-trained on Medical Images

   Natural image databases (e.g., MS COCO) were used to create the YOLO backbone prior to training. The medical images can vary tremendously based on:

   - Texture
   - Intensity distribution
   - Anatomy variability

   The result is poor feature extraction ability (especially for small anomalies) for any models trained on non-medical images.

ML/DL gap:

- There is a lack of ability to use transfer learning from natural image databases to medical image databases.

Project results

Train.py

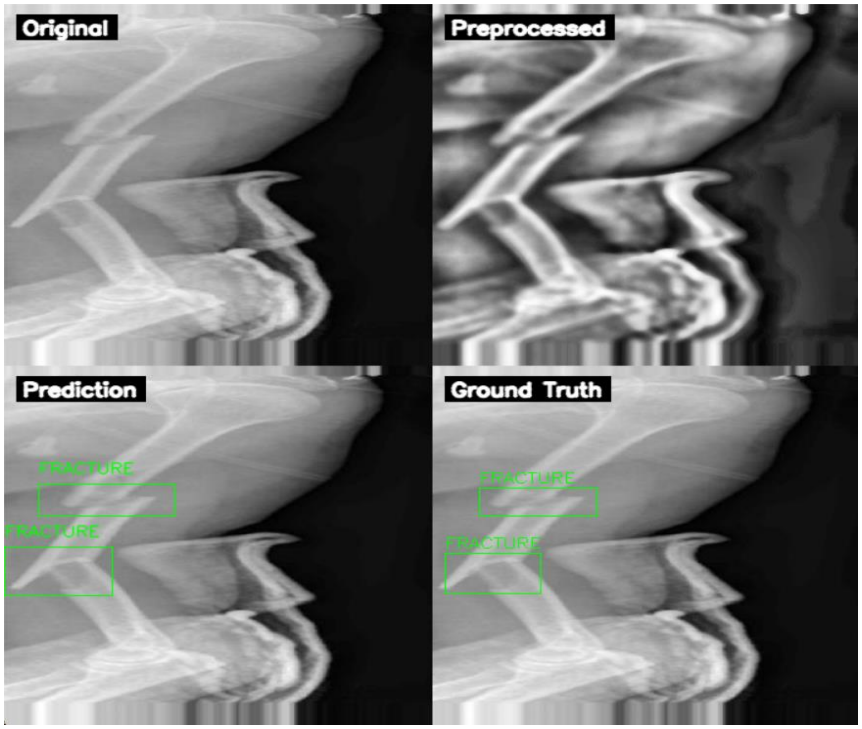
```python
try:
    from ultralytics import YOLO
except Exception as e:
    print("Error importing ultralytics:", e)
    print("Install requirements with: python -m pip install -r req.txt")
    sys.exit(1)


def main():
    base_dir = Path(__file__).resolve().parent

    # Locate data.yaml next to this script
    config_file = base_dir / "data.yaml"
    if not config_file.exists():
        print(f"data.yaml not found at {config_file}")
        print("Make sure data.yaml is placed next to Step03-Train.py or update the path.")
        sys.exit(1)

    # choose a model. Use a released weights file name so ultralytics can download it if missing.
    model_name = "yolov8l.pt"

    try:
        model = YOLO(model_name)
    except Exception as e:
        print(f"Failed to load model '{model_name}':", e)
        print("You can try a smaller model like 'yolov8n.pt' if you have limited resources.")
        sys.exit(1)

    # safer defaults
    project = str(base_dir / "runs" / "train")
    experiment = "My-Model"
    batch_size = 16

    # start training
    try:
        result = model.train(
            data=str(config_file),
            epochs=1000,
            project=project,
            name=experiment,
            batch=batch_size,
            device=0,
            patience=300,
            imgsz=350,
            verbose=True,
            val=True,
        )
        print("Training started. Results saved to:", project)
    except Exception as e:
        print("Training failed:", e)
        sys.exit(1)


if __name__ == "__main__":
    main()
```

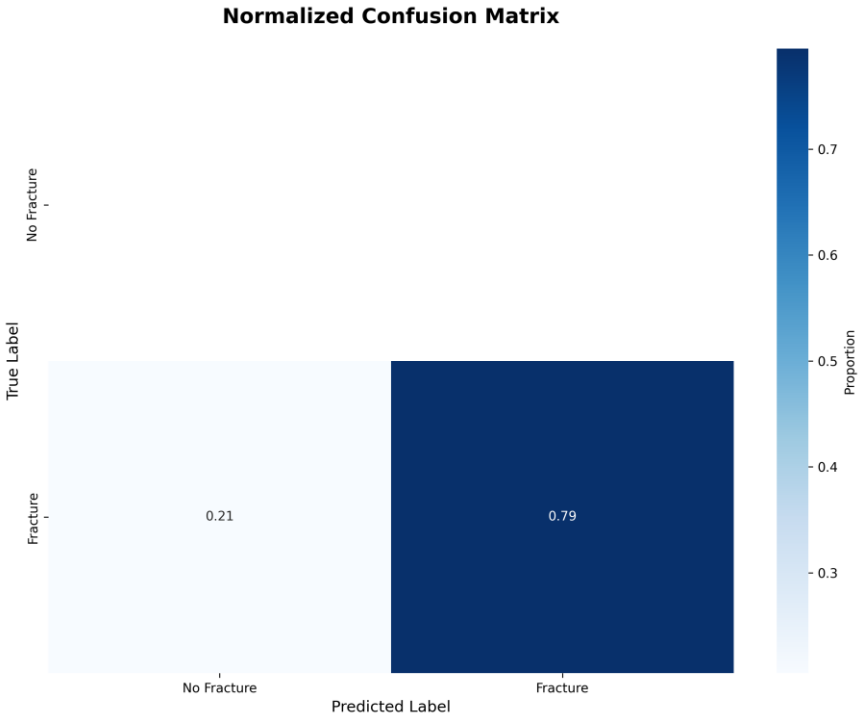
Predict.py

```python
from ultralytics import YOLO
import cv2
import os
from pathlib import Path
import torch
import sys
import numpy as np

# Check Python environment
venv_path = os.path.join(os.path.dirname(sys.executable), '.venv')
if '.venv' in sys.executable or 'venv' in sys.executable.lower():
    print("⚠️  WARNING: Running from .venv environment")
    print("   For best compatibility, use conda environment: conda activate YoloV8")
    print("   Or update ultralytics in .venv: pip install --upgrade ultralytics\n")

# Fix for PyTorch 2.6+ compatibility with model loading
# PyTorch 2.6+ requires explicit safe globals for loading model weights
print("Setting up PyTorch 2.6+ compatibility...")
try:
    from ultralytics.nn.tasks import DetectionModel
    from torch.nn.modules.container import Sequential, ModuleList, ModuleDict
    from torch.nn import Module

    # Add all necessary safe globals for PyTorch 2.6+
    safe_globals = [
        DetectionModel,
        Sequential,
        ModuleList,
        ModuleDict,
        Module,
    ]

    # Add common torch modules that might be in the checkpoint
    try:
        from torch.nn import Conv2d, BatchNorm2d, ReLU, SiLU, Upsample, MaxPool2d, AdaptiveAvgPool2d
        safe_globals.extend([Conv2d, BatchNorm2d, ReLU, SiLU, Upsample, MaxPool2d, AdaptiveAvgPool2d])
    except:
        pass

    # Add safe globals
    torch.serialization.add_safe_globals(safe_globals)
    print("✓ PyTorch 2.6+ compatibility: Safe globals added successfully\n")
except Exception as e:
    print(f"⚠️  Warning: Could not add safe globals: {e}")
    print("   If model loading fails, try:")
    print("   1. Activate conda environment: conda activate YoloV8")
    print("   2. Update ultralytics: pip install --upgrade ultralytics\n")

# Get the base directory (where this script is located)
base_dir = Path(__file__).resolve().parent

def show_preprocessing_comparison(image_name, split='test', save_comparison=True, display=True):
    """



    """
    # Construct paths
    original_path = base_dir / "dataset" / split / "images" / image_name
    preprocessed_path = base_dir / "preprocessing_output" / split / "preprocessed_images" / image_name

    # Check if files exist
    if not original_path.exists():
        print(f"Error: Original image not found at {original_path}")
        return None

    if not preprocessed_path.exists():
0
```

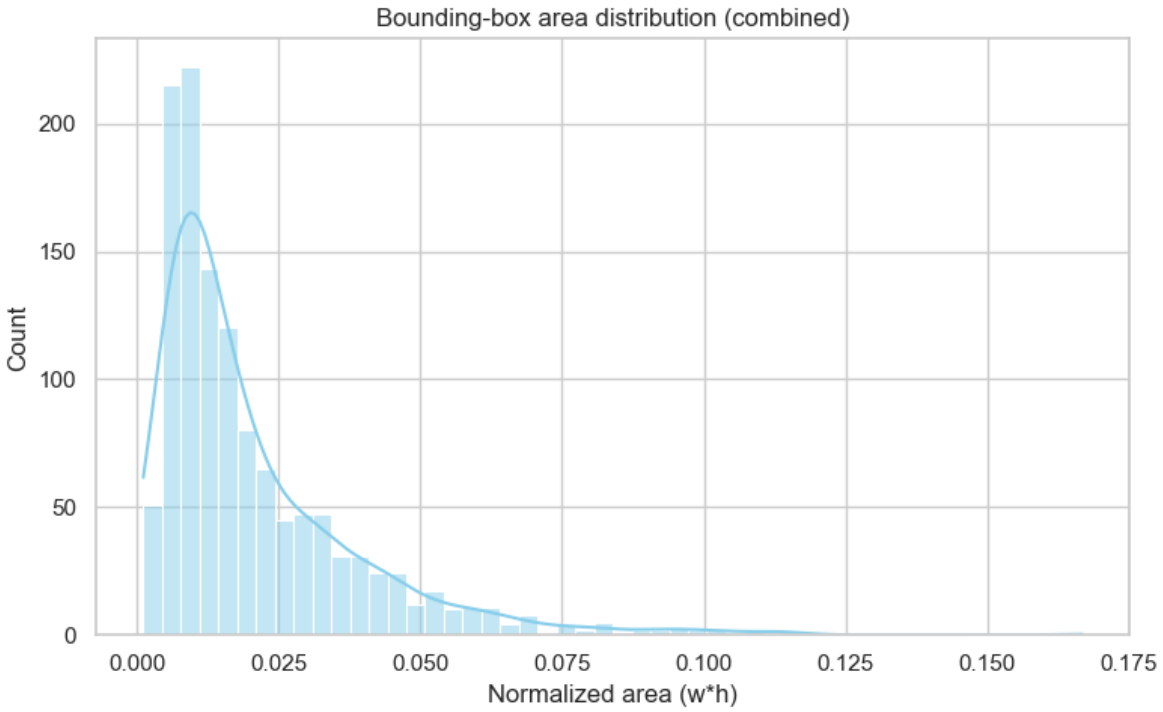
```python
def show_preprocessing_comparison(image_name, split='test', save_comparison=True, display=True):
    # Debug: Print image statistics
    print(f"\nPreprocessed image stats (before enhancement):")
    print(f"  Min: {img_preprocessed.min()}, Max: {img_preprocessed.max()}")
    print(f"  Mean: {img_preprocessed.mean():.2f}, Std: {img_preprocessed.std():.2f}")

    # Apply contrast enhancement to preprocessed image for better visualization
    # The preprocessed image might be normalized and hard to see
    p_min, p_max = img_preprocessed.min(), img_preprocessed.max()
    p_range = p_max - p_min

    if p_range < 100:  # Low contrast image
        # Apply contrast stretching
        if p_range > 0:
            img_preprocessed = ((img_preprocessed.astype(np.float32) - p_min) / p_range * 255).astype(np.uint8)
            print(f"  Applied contrast stretching (range was {p_min}-{p_max})")
        else:
            # All pixels are the same value - apply histogram equalization
            img_preprocessed = cv2.equalizeHist(img_preprocessed)
            print(f"  Applied histogram equalization (all pixels were {p_min})")
    else:
        # Already has good contrast, but apply slight enhancement for visibility
        clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
        img_preprocessed = clahe.apply(img_preprocessed)
        print(f"  Applied CLAHE enhancement for better visibility")

    print(f"  After enhancement - Min: {img_preprocessed.min()}, Max: {img_preprocessed.max()}")

    # Convert original to grayscale for fair comparison (preprocessed is grayscale)
    img_original_gray = cv2.cvtColor(img_original, cv2.COLOR_BGR2GRAY)

    # Resize images to same height if they differ
    h1, w1 = img_original_gray.shape
    h2, w2 = img_preprocessed.shape

    if h1 != h2 or w1 != w2:
        # Resize preprocessed to match original dimensions
        img_preprocessed = cv2.resize(img_preprocessed, (w1, h1), interpolation=cv2.INTER_LINEAR)
        print(f"Note: Resized preprocessed image to match original dimensions ({w1}x{h1})")

    # Ensure both images are uint8 and properly scaled
    img_original_gray = img_original_gray.astype(np.uint8)
    img_preprocessed = img_preprocessed.astype(np.uint8)

    # Add labels to images
    img_original_labeled = img_original_gray.copy()
    img_preprocessed_labeled = img_preprocessed.copy()

    # Add text labels
    font = cv2.FONT_HERSHEY_SIMPLEX
    font_scale = 0.7
    thickness = 2
    color = (255, 255, 255)  # White text

    # Get text size for centering
    (text_width_orig, text_height_orig), _ = cv2.getTextSize("Original", font, font_scale, thickness)
```

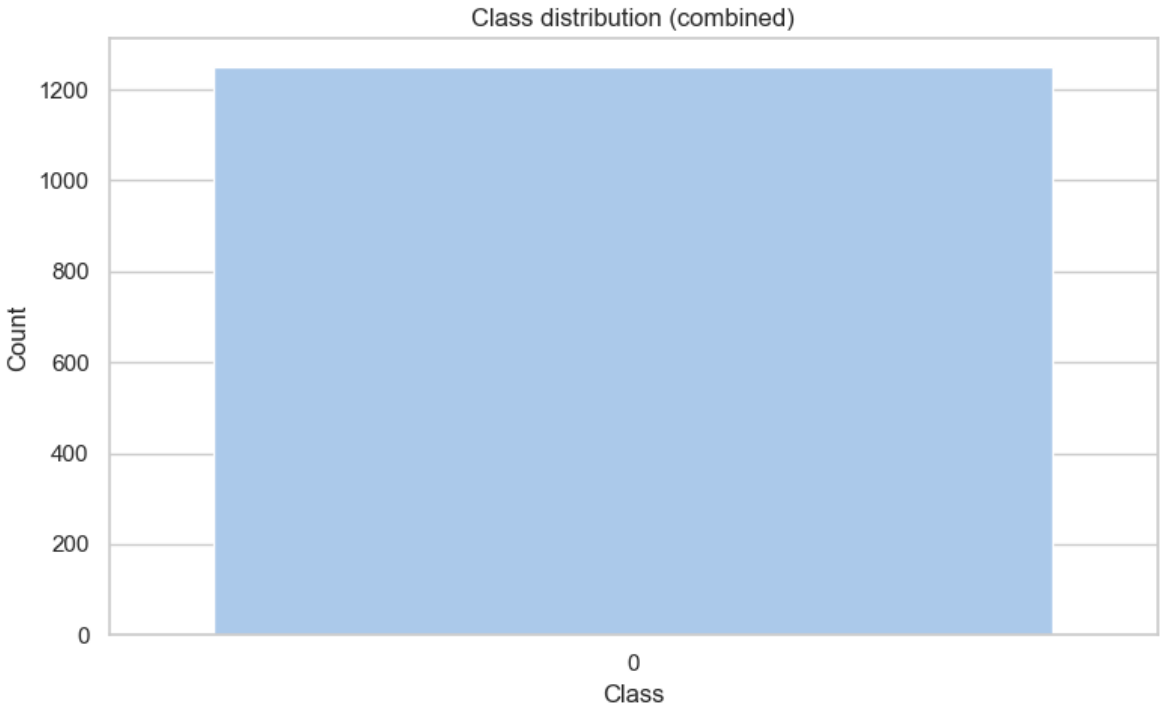
Output






```
Metrics:
  True Positives (TP):  124
  True Negatives (TN):  0
  False Positives (FP): 0
  False Negatives (FN): 32

  Accuracy:  0.79487179
  Precision: 1.00000000
  Recall:    0.79487179
```


Bounding-box area distribution (combined)
Count
200
150
100
50
0
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
Normalized area (w*h)


Class distribution (combined)
Count
1200
1000
800
600
400
200
0
0
Class

Ablation results

```
# Ablation Summary (2025-11-28T14:25:03 UTC)

| Experiment | Model | Data | mAP50 | Precision | Recall | Notes |
| --- | --- | --- | --- | --- | --- | --- |
| baseline_original | yolov8m.pt | data.yaml | n/a | n/a | n/a | YOLOv8m on raw RGB images using the original dataset |
| preprocessed_inputs | yolov8m.pt | data_preprocessed.yaml | n/a | n/a | n/a | Same model trained on grayscale preprocessed images |
| small_model | yolov8n.pt | data.yaml | n/a | n/a | n/a | Quantify impact of model capacity with YOLOv8n |
| high_resolution | yolov8m.pt | data.yaml | n/a | n/a | n/a | Larger 512px input resolution (smaller batch to fit GPU) |
| aggressive_augmentation | yolov8m.pt | data.yaml | n/a | n/a | n/a | Strong color and geometric augmentations |
```

Future scope

The system could be adapted for use beyond wrist fracture applications, enabling use for ankle, elbow, and rib fractures in the future. Future deep learning algorithms will be developed to enhance detection of subtle fractures, utilizing more advanced types of deep learning architectures, including Vision Transformers and hybrid CNN - Transformer architectures. By incorporating multimodal data, including patient history and clinical notes, the reliability of making a diagnosis will be increased. The barrier created by the limited number of available labelled medical images can be eliminated by employing semi-supervised and domain-adapted learning to fill that gap, thus enhancing the effectiveness of each system across multiple institutions. The ability to deploy systems in mobile and edge environments in rural areas will enable real-time usage. Furthermore, incorporation of explainable AI techniques such as Grad-CAM++ will improve transparency, resulting in greater confidence for clinicians and easier integration into existing medical workflows.