

ДОДАТОК А

Програмний код

Лістинг А.1 – Програмний збереження обрізаних за допомогою Yolo хребців із анотацією

```
import os
import cv2
import numpy as np
import pydicom
from ultralytics import YOLO

def save_spine_detections(dicom_path, model_path,
                        conf_threshold=0.25,
iou_threshold=0.45,
                        img_size=384,
output_dir='output_crops'):

    model = YOLO(model_path)
    dicom = pydicom.dcmread(dicom_path)
    image = dicom.pixel_array

    image_norm = cv2.normalize(image, None, 0, 255,
cv2.NORM_MINMAX, cv2.CV_8U)
    image_resized = cv2.resize(image_norm, (img_size,
img_size))
    image_rgb = np.stack([image_resized] * 3, axis=-1)

    results = model.predict(
        source=image_rgb,
        conf=conf_threshold,
        iou=iou_threshold
    )
```

```

class_names = ['L1/L2', 'L2/L3', 'L3/L4', 'L4/L5',
'L5/S1']

if results[0].boxes is not None:
    boxes = results[0].boxes.cpu().numpy()
    detections = []

    for box in boxes:
        cls_id = int(box.cls[0])
        conf = float(box.conf[0])
        x1, y1, x2, y2 = map(int, box.xyxy[0])
        detections.append((y1, cls_id, conf, x1, y1,
x2, y2))

    detections.sort()

    os.makedirs(output_dir, exist_ok=True)

    for idx, (_, cls_id, conf, x1, y1, x2, y2) in
enumerate(detections):
        label = class_names[cls_id].replace('/', '_')
        crop = image_resized[y1:y2, x1:x2]
        label_dir = os.path.join(output_dir, label)
        os.makedirs(label_dir, exist_ok=True)
        filename = f"{label}_{idx}_conf{conf:.2f}.png"
        save_path = os.path.join(label_dir, filename)
        cv2.imwrite(save_path, crop)
    else:
        print("No detections found.")

```

Лістинг А.2 – Програмний попередньої обробки набору даних сканів хребців

```

# Custom Dataset to handle normal and abnormal labels
class CustomDataset(Dataset):

```

```

        def __init__(self, json_file, root_dir, transform=None,
mode="train"):
        """
        Args:
            json_file (string): Path to the json file with
image paths and labels.
            root_dir (string): Path to the directory where
images are stored (images).
            transform (callable, optional): Optional
transform to be applied on an image.
            mode (string): Either "train" or "test" to
specify the dataset split.
        """
        # Load the JSON file containing image paths and
labels

        with open(json_file, 'r') as f:
            self.data = json.load(f)

        self.root_dir = root_dir
        self.transform = transform

        # For training, we only load normal images (label
== 0)

        if mode == "train":
            self.image_paths =
self.data["train"].get(str(0), []) # Only normal images for
training
            self.labels = [0] * len(self.image_paths) #
All labels are 0 for normal images
        elif mode == "test":
            # For testing, we load both normal (label == 0)
and abnormal (label == 1) images
            self.image_paths =
self.data["test"].get(str(0),
+
self.data["test"].get(str(1), [])

```

```

        self.labels = [0] *
len(self.data["test"].get(str(0), [])) + [1] *
len(self.data["test"].get(str(1), [])) # Mix of 0 and 1 for
test images

    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self, idx):
        # Get the image path (relative path from the
dataset)
        image_path = self.image_paths[idx]

        # Combine with the root_dir to get the absolute
path of the image
        image_path = os.path.join(self.root_dir,
image_path)

        # Load the image
        image = Image.open(image_path).convert('L') #
Convert to grayscale

        # Apply transformations if any
        if self.transform:
            image = self.transform(image)

        # Get the corresponding label (0 or 1)
        label = self.labels[idx]

        return image, label

# Define transformation (resize, to tensor, and normalize)
transform = transforms.Compose([
    transforms.Resize((64, 64)),

```

```
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.5],      std=[0.5])      #
Normalize to [-1, 1] range
    ])

    data_json_file = os.path.join(DATA_DIR, 'data.json')      #
JSON file location
    images_dir = os.path.join(DATA_DIR, 'images')      # Images
folder location

    # Create datasets for training and testing
    train_dataset = CustomDataset(json_file=data_json_file,
root_dir=images_dir, transform=transform, mode="train")
    test_dataset = CustomDataset(json_file=data_json_file,
root_dir=images_dir, transform=transform, mode="test")

    # Create DataLoaders for each subset
    train_dataloader = DataLoader(train_dataset,
batch_size=32, shuffle=True)
    test_dataloader = DataLoader(test_dataset, batch_size=32,
shuffle=False)
```

