Spill Detection Model Documentation

1 Introduction

This project aims to develop a machine learning model to accurately detect liquid spills on floors using image data. The goal is to create a model that can both identify the presence of spills and locate them using bounding box annotations. The dataset contains 44 images, and the model needs to handle this data effectively, process it for training, and produce two models: a primary model (75MB) and a lightweight model (10MB).

2 Approach

2.1 Problem Understanding

The primary challenge is to detect and locate liquid spills in images, which requires an object detection model capable of both classification and localization. This task involves:

- Identifying the presence of spills.
- Drawing bounding boxes around the spills.
- From Bounding box find the precise center of it to locate the spill

2.2 Chosen Approach

- **Object Detection**: Using a **YOLO (You Only Look Once)** as object detection model due to its speed and effectiveness for real-time applications. YOLOv5 or YOLOv8 will be used to detect spills in images, as it provides state-of-the-art performance in terms of both speed and accuracy.
- **Data pre-processing and augmentation**: Given the limited dataset (44 images), here using effective pre-processing and augmentation to increase the data size from (44-80)
- **Choosing Model**: To meet the size requirement, we will use a **yolov8m** as a priimary model and **yolov5n** as a light weight model. While training hyperperameter tuned accordingly.

- **Correctly determine the location**: To cross examine the detected spill for reducing False positive and correctly determine it's location here using a U-Net segmentation model to segment the spill detected by the yolo and determine the center of the segmentation.

3 Data Preparation

3.1 Data Overview

- **Number of images**: 44 images with spill-related scenarios.
- **Annotations**: Bounding boxes around spills using labelImg.
- **data preprocessing and augmentation:** Increased size to 80 images
- **Training-Validation Split**:
  - 80% training (64 images).
  - 20% validation (16 images).

3.2 Data Labeling

- **Tool Used**: `LabelImg` was used to annotate the images and generate YOLO-compatible labels (class and bounding box coordinates).

  `labelMe` was used to polygon annotation for UNet model to segment the spill later.
- **Class Label**: Spill (class index: 0).

3.3 Data Preprocessing and Augmentation

Script : image_preprocessing.ipynb

- **A. Resize(640, 640)**: Resizes the image to a fixed size of 640x640 pixels.
- **B. HorizontalFlip(p=0.5)**: Horizontally flips the image with a 50% probability.
- **C. Rotate(limit=30, p=0.5)**: Randomly rotates the image within a range of ±30 degrees (50% chance).
- **D.RandomBrightnessContrast(0.1, 0.1, p=0.5)**: Randomly adjusts brightness and contrast by ±10% with 50% probability.

4 Model Development

4.1 Primary Model (YOLOv8m) (Size: 52 MB)

- **Model Architecture**: YOLOv8 was chosen for its balance between performance and efficiency.
- **Pre-training**: The model was pre-trained on the COCO dataset and fine-tuned on the spill detection dataset.
- **Hyper perameter tuning**:

Hyperparameter	Value	Description
`epochs`	100	Total number of training epochs.
`imgsz`	640	Image size used for training and validation (input resolution).
`batch`	16	Batch size – number of images per batch.
`lr0`	0.001	Initial learning rate.
`lrf`	0.01	Final learning rate (after cosine decay).
`momentum`	0.937	Momentum for SGD optimizer.
`weight_decay`	0.0005	L2 regularization to reduce overfitting.
`warmup_epochs`	5	Number of warm-up epochs (gradually increases learning rate for stability).

Table 1: Tuned Hyperparameters for YOLO Training

4.2 Lightweight Model (YOLOv5n) (Size: 5.3 MB)

- **Model Architecture**: YOLOv5n (nano) was selected due to its ultra-lightweight design.
- **Pre-training**: The model was pre-trained on the COCO dataset and then fine-tuned on the custom spill detection dataset to adapt to the specific task.
- **Hyperparameter Tuning**: Same hyperpermeter tuning followed.

5 Model Evaluation

5.1 Metrics

- **Precision**: Measure of correct spill predictions out of all predictions made.
- **Recall**: Measure of correct spill predictions out of all actual spills.
- **mAP (mean Average Precision)**: Overall measure of model performance across different classes and IoU thresholds.

- **F1 Score**: Harmonic mean of Precision and Recall, balancing both metrics.

5.2 Performance Results

- **Primary Model**:
  - Precision: 94%
  - Recall: 78%
  - mAP50 (B): 86% (represents the average precision at IoU threshold of 0.5).
  - mAP50-95 (B): 56% (represents the average precision at multiple IoU thresholds from 0.5 to 0.95).
  - F1 score: 81%
- **Lightweight Model**:
  - Precision: 95%
  - Recall: 85%
  - mAP50 (B): 89% (represents the average precision at IoU threshold of 0.5).
  - mAP50-95 (B): 60% (represents the average precision at multiple IoU thresholds from 0.5 to 0.95).
  - F1 score: 89%

  In conclusion, the Lightweight Model (YOLOv5n) demonstrates superior performance compared to the Primary Model (YOLOv8x), achieving higher precision, recall, and F1 score.

5.3 Calculating the Location of Spill Using YOLO Bounding Box

- For each detected object (spill), the YOLO model provides bounding box coordinates: top-left (x_1, y_1) and bottom-right (x_2, y_2).
- The center of the spill is calculated using: center $= 1 + 2\ 2$, center $= 1 + 2\ 2$ center x $= 2$ x $1 +$x 2

  ,center y $= 2$ y $1 +$y 2
- This center point $(\text{center}_x, \text{center}_y)$ represents the estimated spill location in the image.
- A red circle is drawn at this center point on the image for visual confirmation.
- The coordinates of this spill location are saved into a CSV file for further reference.

5.4 Inferencing with YOLO Model (inference_yolo.ipynb)

- The YOLOv8 / YOLOv5 model is loaded using a pre-trained checkpoint.
- All images in the validation directory are processed for inference.
- Each image is passed through the model to detect spills with a confidence threshold of 0.5.
- For each detection:
  - Bounding box is drawn in green.
  - The predicted label and confidence score are overlaid above the box.
  - The center of the bounding box (spill location) is highlighted with a red circle and labeled.
- Annotated images are saved into an output directory for visualization.(inference_output_v8finetuned)
- All detection details including image name, label, confidence, bounding box coordinates, and center point are saved into a CSV file (detection_results_yolov8.csv).

5.5 Visualization

(shown below)

- Visualization of output image inferencing yolov5m Figure1
- Edge case: visualuization where not able to fix the location of the spill. Figure2
- After U-Net segementation: solving the edge case Figure3
- Output on critically annoatated image Figure3

6 Challenges and Solutions

6.1 Limited Dataset

- **Challenge**: Only 44 images are available, which limits the model's ability to generalize.
- **Solution**: Augmentation of the dataset is done.

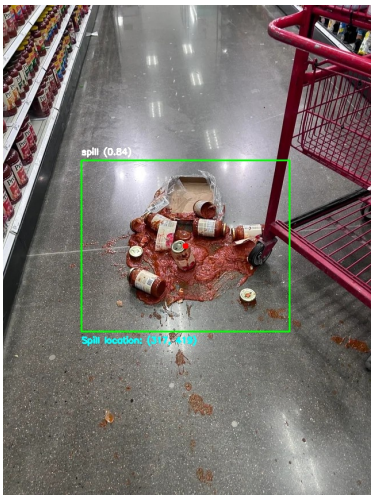

Figure 1: YOLO model output showing predicted bounding box and spill location

Figure 2: Edge case: Model not able to fix the location of the spill

Figure 3: Edge case solving by U-Net model segmenteation

6.2 Bounding Box Accuracy and Location determined

- **Challenge**: Ensuring that bounding boxes accurately surround spills, especially in cluttered or partially visible spill scenarios.
- **1. Solution**: Fine-tuned hyperparameters for efficient training.
- **2. Solution**: For edge cases like model not able to fix the center location and also reducing the false positive here implemented a cross validation step where a **UNet** model is take input as the crop bounding box detected by the yolo model and segment the spill region and give the accurate center of the spill.

7 U-Net for segmentation:(train_segmentation_model.ipynb)

7.1 Data preparation:

- The segmentation dataset was initially annotated using the LabelMe tool, where the spill regions were marked using polygon shapes and saved as `.json` files.
- Each `.json` file contains the base64-encoded image data and shape annotations with labels.
- A Python script was used to:
  - Decode the embedded image from the JSON file.
  - Convert the polygon annotations into a binary mask using 'labelme.utils.shapes_to_label'.
  - Assign unique integer values to each label starting from the background (0).
  - Save the decoded image and its corresponding binary mask as `.png` files in separate folders: one for images and one for masks.
- This preprocessing generated paired image-mask datasets which were used for training the U-Net model.

7.2 Model architecture and training:

- The U-Net model architecture was used for semantic segmentation, which is particularly effective for pixel-level classification tasks.
- The architecture consists of an encoder-decoder structure, where:
  - The encoder progressively reduces the spatial dimensions of the input while extracting features.
  - The decoder upscales the feature maps, combining them with skip connections from the encoder to restore spatial information.
- The model is designed to take images of size 256x256 and output a binary mask indicating the presence of spills.

- The model was trained using the Adam optimizer with a binary cross-entropy loss function and accuracy as the evaluation metric.
- A custom data generator class `SpillDataset` was used to load and preprocess images and masks for training and validation:
  - The generator resizes images and masks to the target size of 256x256 pixels.
  - The masks are converted into binary format, where spill areas are labeled as 1 and the background as 0.
- The dataset was split into training and validation sets with 80% for training and 20% for validation.
- The model was trained for 25 epochs with a batch size of 8.
- After training, the model was saved as `spill_unet_model.h5`.
- The training accuracy and validation accuracy were plotted to visualize model performance over time.

8 Final inference with YOLO and U-Net (final_inference_with_segmentation.py)

In this section, the final inference pipeline is designed to detect spills using YOLOv8 for object detection and U-Net for precise spill segmentation. The following steps outline the process:

8.0.1 Object Detection with YOLOv8

The YOLO model is employed to detect potential spill regions in the input image.

- **Non-Maximum Suppression (NMS):** A threshold is applied to filter out redundant and overlapping bounding boxes. The confidence threshold is set at 0.2 . Then bounding boxes coordinates (x1, y1, x2, y2) are extracted.

8.0.2 Cropping the Detected Spill Region

Once the spill regions are detected, the bounding boxes are used to crop the respective areas from the original images and resize it to 256X256 for further segmentation by the U-Net model.

8.0.3 Spill Segmentation with U-Net

The cropped spill regions are passed through the U-Net-based model for segmentation. A threshold of 0.3 is applied to the predicted mask to classify pixels as either spill (1) or non-spill (0). The resulting binary mask is resized back.

8.0.4 Center Estimation using Image Moments

To accurately localize the spill within the bounding box:

- **Center Calculation:** OpenCV's `cv2.moments` function is applied to the binary mask to compute the center of mass (centroid) of the spill area.
- **Mapping to Original Image:** The centroid coordinates of the spill are adjusted to correspond to the original image's coordinate system by adding the bounding box offset.

8.0.5 Bounding Box and center Visualization and saving

Bounding box detected by the yolo model is visualized by green color and the center calculated from segmentation is marked as red color. The name of the processed image and the corresponding spill coordinates (x, y) are stored in a CSV file for later reference or evaluation. The final processed image, with all annotations, is saved to the specified output directory for future analysis or visual validation.

9 Failed to execute approach

- **Efficientdet model for spill detection**: proven better result than yolov5 model
- **progress achieved**:
  - Dataset annotation in Xml
  - From xml file to tf record file for both train.record and val.record.
- **Failed to train due to environment set up issue**

10 Conclusion

Successfully developed and evaluated two YOLO-based models for spill detection. The models meet the size constraints and perform reasonably well with both accuracy and localization..