
# รายงานโครงงานสุดท้าย

## การตรวจจับวัตถุต้องห้ามในภาพเอกซเรย์โดยใช้โมเดล YOLOv11

## X-ray Prohibited Item Detection Using YOLOv11

**จัดทำโดย**

**นาย ชวัลวิทย์ เกียรติณัฐกร 6510503310**

**นาย ธัญชนก ปั้นมีรส 6510503441**

**เสนอ**

**ผู้ช่วยศาสตราจารย์ ดร.ภารุจ รัตนวรพันธุ์**

**รายวิชา 01204466 การเรียนรู้เชิงลึก (Deep Learning)**

**ประจำปีการศึกษา 2568**

**ภาควิชาวิศวกรรมคอมพิวเตอร์**

**มหาวิทยาลัยเกษตรศาสตร์**

# กิตติกรรมประกาศ

โครงงานนี้สำเร็จลุล่วงได้ด้วยดี คณะผู้จัดทำขอขอบพระคุณ ผู้ช่วยศาสตราจารย์ ดร.ภารุจ รัตนวรพันธุ์ อาจารย์ประจำวิชา ที่ได้ให้คำแนะนำและแนวทางในการพัฒนาโครงงานมาโดยตลอด

ขอขอบคุณ Zhang et al. (2022) สำหรับบทความวิจัย A Large-scale X-ray Benchmark for Real-World Prohibited Item Detection (PIDray) (arXiv:2211.10763) และสำหรับชุดข้อมูล (dataset) PIDray ซึ่งเป็นทรัพยากรข้อมูลหลักที่สำคัญอย่างยิ่งในการฝึกสอน (train) และประเมินผลโมเดลในโครงงานนี้

ขอขอบคุณ Ultralytics สำหรับ ultralytics library ซึ่งเป็น PyTorch implementation ที่ช่วยให้การพัฒนา ฝึกสอน และ fine-tuning โมเดล YOLOv11s เป็นไปอย่างมีประสิทธิภาพ

สุดท้ายนี้ ขอขอบคุณสมาชิกในกลุ่มที่ได้ร่วมมือกันทำงานอย่างเต็มความสามารถ จนโครงงานนี้สำเร็จลุล่วงได้ด้วยดี

คณะผู้จัดทำ
นาย ชวัลวิทย์ เกียรติณัฐกร
นาย ธัญชนก ปั้นมีรส

# ที่มาและความสำคัญ

ในปัจจุบัน การรักษาความปลอดภัยสาธารณะ (public safety) โดยเฉพาะในพื้นที่ที่มีการสัญจรหนาแน่น เช่น สนามบิน หรือจุดตรวจความปลอดภัย ถือเป็นสิ่งที่มีความสำคัญอย่างยิ่ง หนึ่งในความท้าทายหลักของภารกิจนี้คือการตรวจจับวัตถุต้องห้าม (prohibited items) ซึ่งมักถูกซุกซ่อนไว้ในกระเป๋าสัมภาระ

การตรวจสอบภาพ X-ray ด้วยสายตามนุษย์หรือการใช้ระบบ Computer Vision (CV) แบบดั้งเดิม (Traditional CV) นั้นมีข้อจำกัดหลายประการ ภาพ X-ray มักมีความซับซ้อนสูง วัตถุมีการปะปน (cluttered) และซ้อนทับกัน (occlusion) ทำให้วิธีการดั้งเดิม เช่น Template Matching หรือ Feature Descriptors ทำงานได้ไม่มีประสิทธิภาพเพียงพอต่อความแปรปรวน ทั้งในแง่ของการหมุน (rotation) หรือการเปลี่ยนแปลงขนาด (scale) นอกจากนี้ วิธีการเหล่านี้ยังต้องการการทำ Feature Engineering ซึ่งเป็นกระบวนการที่ซับซ้อนและแทบเป็นไปไม่ได้สำหรับข้อมูลภาพ X-ray ที่มีจำนวนมหาศาล

ด้วยเหตุนี้ เทคโนโลยี Deep Learning (DL) โดยเฉพาะสถาปัตยกรรม Convolutional Neural Network (CNN) จึงเข้ามามีบทบาทสำคัญ โมเดล CNN ถูกออกแบบมาเพื่อเรียนรู้ features ที่ซับซ้อนเหล่านี้ได้โดยอัตโนมัติ (Automatic Feature Extraction) ทำให้มีความทนทาน (robust) ต่อความแปรปรวนในภาพได้ดีกว่ามาก

โครงงานนี้จึงมุ่งเน้นการพัฒนา X-ray Prohibited Item Detection Using YOLOv11 โดยมีจุดประสงค์เพื่อประยุกต์ใช้โมเดล State-of-the-Art (SOTA) อย่าง YOLO ซึ่งเป็นโมเดลแบบ One-Stage Detector ที่มีจุดเด่นด้านความเร็วสูง (real-time performance) ซึ่งเป็นปัจจัยสำคัญสำหรับงานที่ต้องการการตัดสินใจที่รวดเร็วอย่างจุดตรวจความปลอดภัย นอกจากนี้ โครงงานนี้ไม่ได้มุ่งเน้นเพียงการ train โมเดล แต่ยังครอบคลุมถึงการสร้างระบบที่ครบวงจร (end-to-end system) ตั้งแต่การ train model, การสร้าง Backend API สำหรับให้บริการโมเดล ไปจนถึง Frontend ที่ช่วยให้ผู้ใช้สามารถป้อนภาพและได้รับผลลัพธ์ (bounding boxes, label, confidence) กลับไปได้อย่างมีประสิทธิภาพ

## 1. ชื่อหัวข้อโครงงาน

ชื่อภาษาไทย: การตรวจจับวัตถุต้องห้ามในภาพเอกซเรย์โดยใช้โมเดล YOLOv11

ชื่อภาษาอังกฤษ: X-ray Prohibited Item Detection Using YOLOv11

## 2. ความน่าสนใจของหัวข้อ

ที่เลือกหัวข้อนี้เพราะเป็นปัญหาที่มีความสำคัญในการใช้งานจริง และเกี่ยวข้องกับความปลอดภัยสาธารณะ โดยตรง เช่น การตรวจจับวัตถุต้องห้ามในสนามบิน หรือในระบบรักษาความปลอดภัย ซึ่งความน่าสนใจของโปรเจคนี้คือ

1. มีความยากในการทำข้อมูล: จากตัวอย่างภาพ X-ray วัตถุต้องห้ามมักจะปะปน (cluttered) กับวัตถุอื่นๆ และมีการซ้อนทับกัน (occlusion) ทำให้การตรวจจับด้วยสายตามนุษย์หรือวิธีดั้งเดิมทำได้ยาก

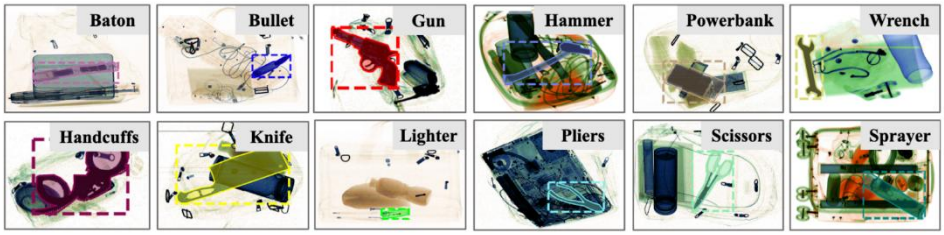
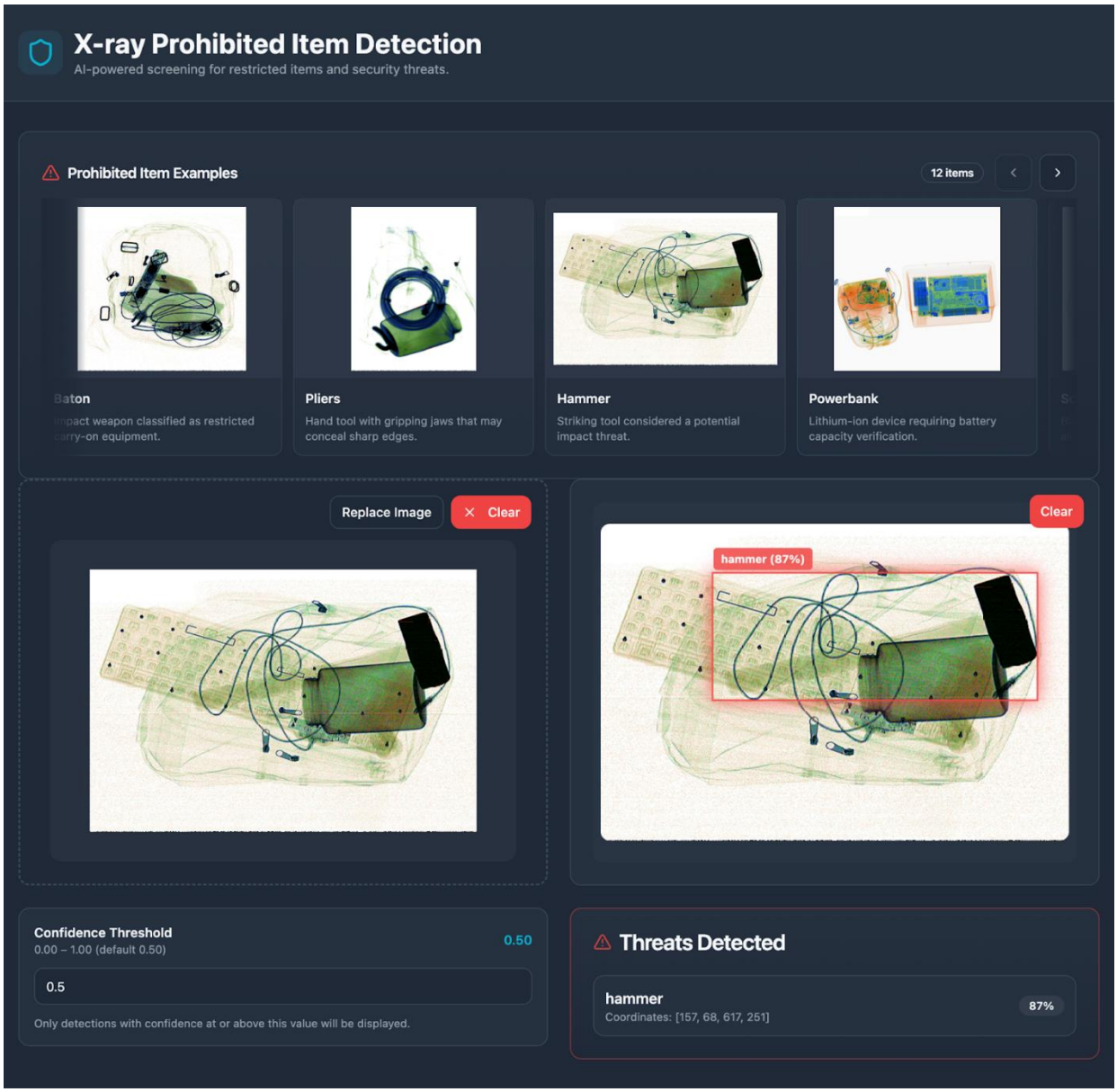


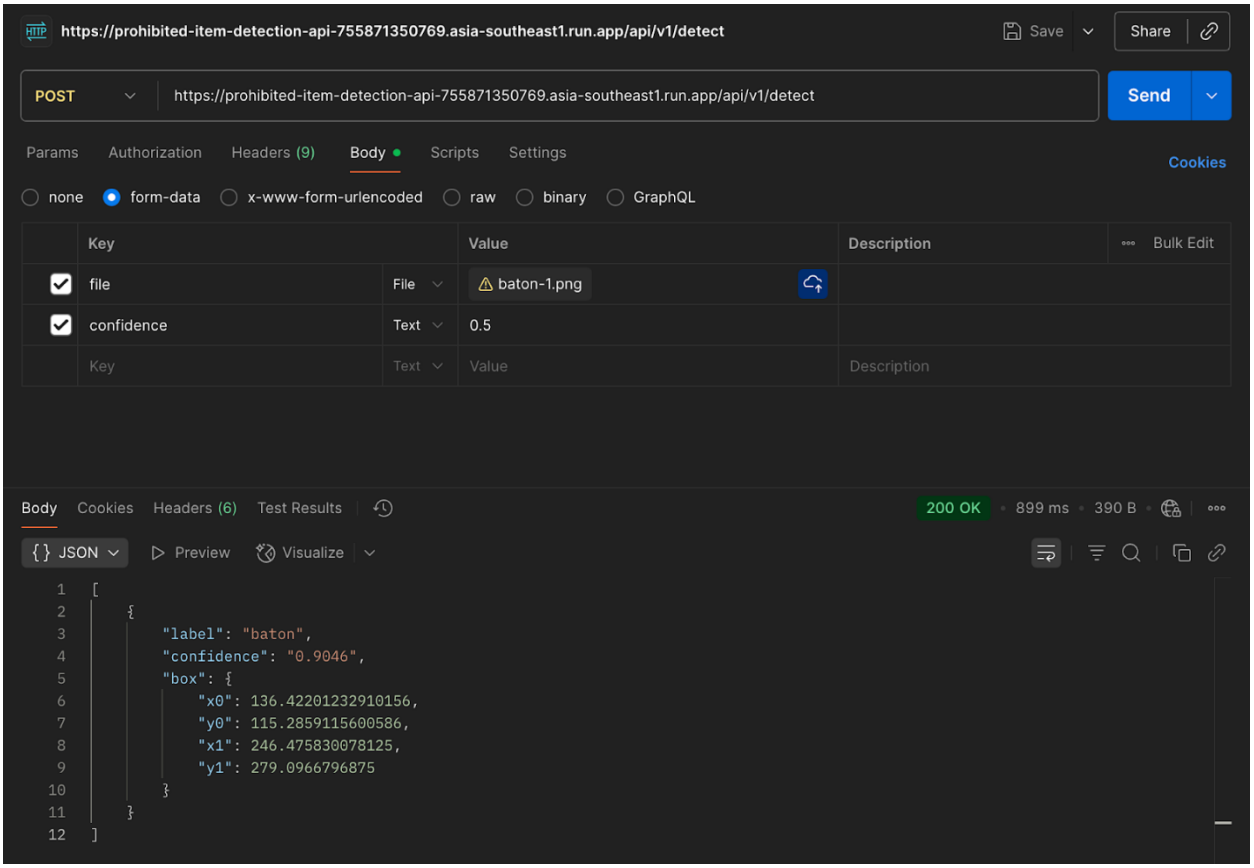

**Fig. 2** Example images in the PIDray dataset with 12 categories of prohibited items. Each image is provided with image-level and instance-level annotation. For clarity, we show one category per image.

*ภาพตัวอย่างชุดข้อมูลบางส่วนของ PIDray dataset*

2. เป็นโอกาสในการนำโมเดล Object Detection ที่เป็น State-of-the-Art (SOTA) อย่าง YOLO มาประยุกต์ใช้กับปัญหาที่ซับซ้อน
3. มีการสร้างระบบ End-to-End: project นี้ไม่ได้มีแค่การ train โมเดล แต่เราได้สร้างระบบที่ครบวงจร (end-to-end system) ตั้งแต่การ train model, การสร้าง Backend API ไปจนถึง Frontend ให้ผู้ใช้ป้อนภาพและแสดงผลลัพธ์ (bounding boxes, label, confidence)




*ภาพตัวอย่างเว็บแอพพลิเคชั่น (https://x-ray-detection.vercel.app/)*




*ภาพตัวอย่างการทดสอบ API บน Postman*

cURL

```
curl --location 'localhost/api/v1/detect'
--form 'file=@"/path/to/file.jpeg"'
--form 'confidence="0.5"'
```

## 3. Deep Learning ในการแก้ปัญหาหัวข้อนี้

เพราะ การตรวจจับวัตถุในภาพ X-ray มีความซับซ้อนที่วิธี Computer Vision (CV) แบบดั้งเดิม จัดการได้ยาก

- Traditional CV: วิธีอย่าง Template Matching หรือการใช้ Feature Descriptors (เช่น SIFT, SURF) จะทำงานได้ไม่ดีเมื่อวัตถุมีการหมุน (rotation), เปลี่ยนขนาด (scale), หรือถูกซ้อนทับ (occlusion) นอกจากนี้ ยังต้องการการทำ Feature Engineering ด้วยมือ ซึ่งแทบเป็นไปไม่ได้หรือไม่คุ้มที่จะทำในภาพ X-ray เพราะมีรูปภาพจำนวนมาก
- Convolutional Neural Network (CNN): ที่เป็น concept หลักของ YOLO ถูกออกแบบมาเพื่อเรียนรู้ features ที่ซับซ้อนเหล่านี้โดยอัตโนมัติ (Automatic Feature Extraction) จากข้อมูลจำนวนมาก มันจึงทนทาน (robust) ต่อความแปรปรวนต่าง ๆ ในภาพได้ดีกว่า

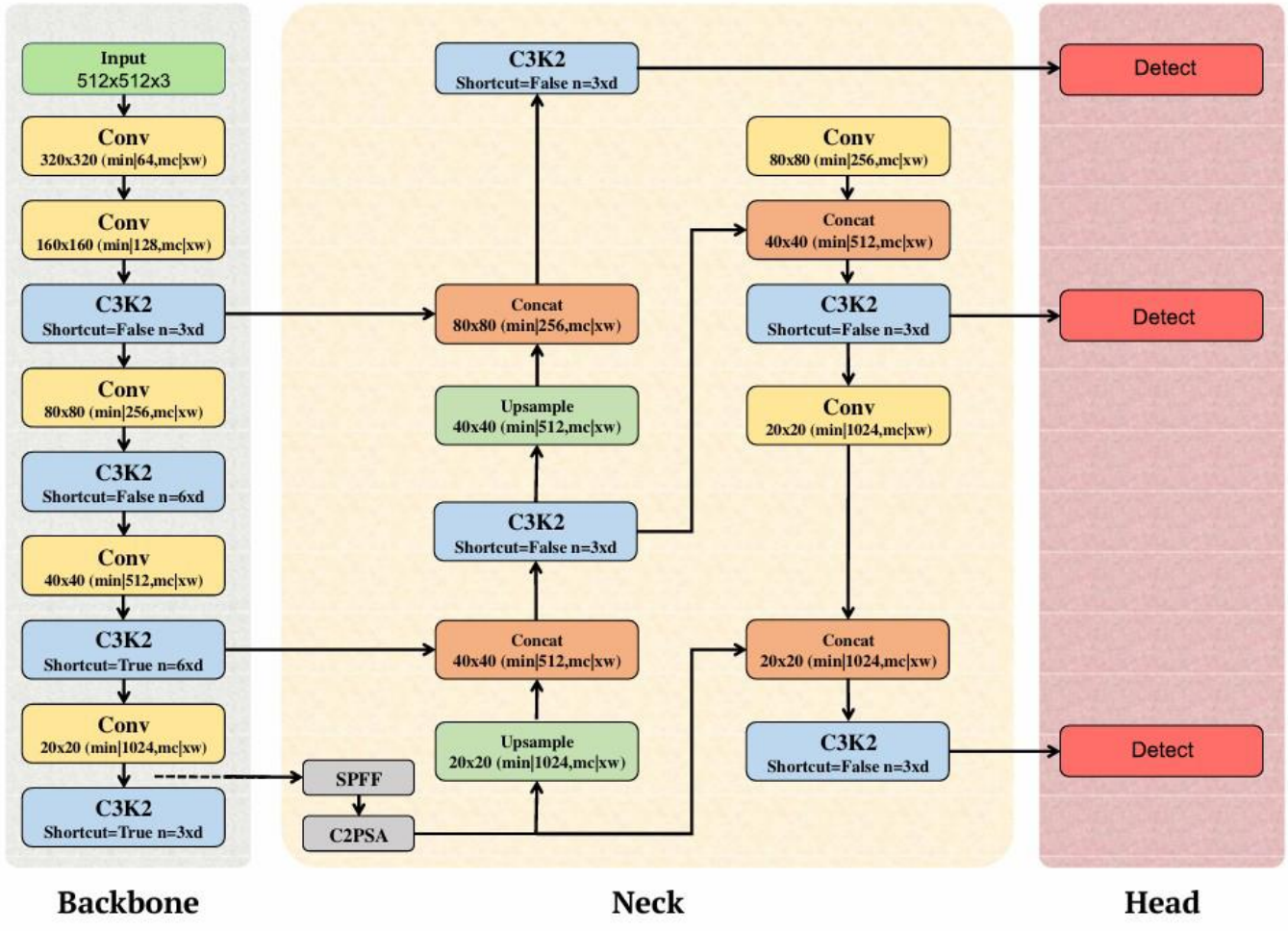
ข้อเด่นของ Deep Learning (YOLO) ในงานนี้:

1. Accuracy & Robustness: สามารถเรียนรู้ feature ที่ซับซ้อนของวัตถุต้องห้าม 12 classes (เช่น 'gun', 'knife', 'lighter') แม้จะมีการซ้อนทับ
2. Performance: YOLO เป็นโมเดลแบบ One-Stage Detector ที่ขึ้นชื่อเรื่องความเร็วสูง (real-time performance) ซึ่งจำเป็นอย่างยิ่งสำหรับงานที่ต้องการการตัดสินใจที่รวดเร็วอย่างจุดตรวจความปลอดภัย

ข้อด้อย:

1. ต้องการข้อมูลจำนวนมากในการ train (ซึ่งเรามี dataset ขนาดใหญ่ถึง 35,000 ภาพสำหรับ train)
2. ใช้ทรัพยากรในการคำนวณสูง
3. เป็นเหมือน black box อธิบายได้ยากว่าทำไมโมเดลถึงตัดสินใจแบบนั้น

## 4. สถาปัตยกรรม Deep Learning ที่ใช้




*ภาพรวมสถาปัตยกรรม YOLOv11 ที่ใช้รูปภาพขนาด 512x512 เป็น input*

ในโครงงานนี้ได้ใช้โมเดล YOLOv11s ซึ่งเป็นโมเดลประเภท Convolutional Neural Network (CNN) สำหรับงาน Object Detection โดยมีโครงสร้างหลักแบ่งออกเป็น 3 ส่วน ดังนี้

1. Backbone: เป็นส่วนของเครือข่ายที่ทำหน้าที่สกัดคุณลักษณะ (Feature Extraction) จากภาพอินพุต โดยผ่านการประมวลผลด้วย convolutional layers หลายชั้น เพื่อแยกแยะลักษณะของวัตถุในระดับต่าง ๆ เช่น ขอบ สี และรูปทรง

2. Neck: ทำหน้าที่รวมและปรับสมดุลข้อมูลจากหลายระดับของ Backbone (Multi-scale Feature Fusion) โดยใช้โครงสร้างแบบ PANet (Path Aggregation Network) เพื่อให้โมเดลสามารถตรวจจับวัตถุได้ทั้งขนาดเล็กและขนาดใหญ่ได้อย่างมีประสิทธิภาพ

3. Head: เป็นส่วนสุดท้ายของโมเดล ใช้ feature maps ที่ได้จาก Neck มาทำนายผลลัพธ์ของการตรวจจับ ซึ่งประกอบด้วย

- Bounding Boxes (box_loss): พิกัดของกรอบวัตถุที่ตรวจพบ
- Class Probabilities (cls_loss): ความน่าจะเป็นของแต่ละคลาส เช่น gun (90%) หรือ knife (5%)
- Objectness Score: ค่าความมั่นใจว่าในกรอบนั้นมีวัตถุจริง ๆ

จากการดูสรุปสถาปัตยกรรมของโมเดลในไฟล์ฝึกสอน (*train.ipynb*) พบว่า YOLOv11s ที่ใช้ในงานนี้มีทั้งหมด 181 layers และมีจำนวนพารามิเตอร์รวม 9,432,436 ค่า ซึ่งครอบคลุมทั้ง Weights และ Biases ของทุกชั้นในเครือข่าย

สำหรับฟังก์ชันกระตุ้น (Activation Function) โมเดล YOLOv11s ใช้ SiLU (Sigmoid Linear Unit) เป็นฟังก์ชันหลัก เนื่องจากมีความสามารถในการรักษาข้อมูลเชิงเส้นและไม่เชิงเส้นได้ดี ช่วยให้โมเดลเรียนรู้ฟีเจอร์ได้อย่างราบรื่นและมีเสถียรภาพมากกว่า ReLU ในเวอร์ชันก่อนหน้า

## 5. อธิบายโค้ดที่ใช้ในโครงงงาน

### 5.1 เตรียมชุดข้อมูล (*dataset.ipynb*)

5.1.1. การโหลดและจัดโครงสร้างข้อมูล

เริ่มจากอ่านไฟล์ train.json และ test.json ซึ่งอยู่ในรูปแบบ COCO (มี key: images, annotations, categories) จากนั้นใช้ฟังก์ชัน *get_dataset_annotations()* เพื่อรวมข้อมูลภาพและ annotation ของแต่ละภาพเข้าด้วยกันให้อยู่ในรูปแบบที่เข้าถึงได้ง่าย เช่น

```
dataset[image_id] = { "image": {filename, width, height}, "objects": [{class_id, bbox}, ...] }
```

ทำให้สามารถเรียกดูข้อมูลของแต่ละภาพและกรอบวัตถุได้โดยตรง

```python
def get_dataset_annotations(data: dict) -> dict:
    images = data["images"]
    annotations = data["annotations"]

    dataset_annotations = {}

    for image in tqdm(images, desc = "Processing image annotations", unit = "image", ncols = 1000):
        filename = image["file_name"]
        image_id = image["id"]
        width = image["width"]
        height = image["height"]

        image_data = {
            "filename": filename,
            "width": width,
            "height": height,
        }

        dataset_annotations[image_id] = {
            "image": image_data,
            "objects": []
        }

        for annotation in annotations:
            if annotation["image_id"] != image_id:
                continue

            class_id = annotation["category_id"]
            bbox = annotation["bbox"]
            object_data = {
                "class_id": class_id,
                "bbox": bbox,
            }

            dataset_annotations[image_id]["objects"].append(object_data)

    return dataset_annotations
```

5.1.2. การสำรวจและสรุปข้อมูลเบื้องต้น

ใช้ฟังก์ชัน *get_dataset_information()* เพื่อสรุปจำนวนภาพทั้งหมด จำนวนกรอบวัตถุ และจำนวนภาพที่ไม่มี annotation นอกจากนี้ยังมี *get_class_distribution()* สำหรับนับจำนวนและสัดส่วนของแต่ละคลาส เพื่อประเมินความสมดุลของข้อมูล

```python
def get_dataset_information(dataset: list | dict) -> dict:
    dataset_info = {}

    if type(dataset) is dict:
        dataset = list(dataset.values())

    total_images = len(dataset)
    total_annotations = sum(len(item["objects"]) for item in dataset)
    annotated_image_count = sum(1 for item in dataset if len(item["objects"]) > 0)
    unannotated_image_count = total_images - annotated_image_count

    dataset_info["total_images"] = total_images
    dataset_info["total_annotations"] = total_annotations
    dataset_info["annotated_image_count"] = annotated_image_count
    dataset_info["unannotated_image_count"] = unannotated_image_count

    return dataset_info
```

```python
def get_class_distribution(split_set: list) -> dict:
    class_distribution = {}
    total_annotations = 0

    for item in split_set:
        for annotation in item["objects"]:
            class_id = annotation["class_id"]

            if class_id not in class_distribution:
                class_distribution[class_id] = {"count": 1}
            else:
                class_distribution[class_id]["count"] += 1

            total_annotations += 1

    for info in class_distribution.values():
        info["percentage"] = f"{info['count'] / total_annotations * 100:.2f}%"

    class_distribution = dict(sorted(class_distribution.items()))

    return class_distribution
```

5.1.3. การแบ่งชุดข้อมูล

ทำการแบ่งชุดข้อมูลออกเป็น Training / Validation / Testing ดังนี้

- ฟังก์ชัน *undersample()* ใช้เพื่อลดจำนวนภาพที่ไม่มีวัตถุ (กำหนด null_ratio=0.3) เพื่อบาลานซ์ข้อมูล
- ฟังก์ชัน *split_dataset()* แบ่งชุดข้อมูลฝึกออกเป็น Train 35,000 ภาพ และ Validation 5,000 ภาพ
- ชุด Testing ถูกสุ่มเลือกมา 5,000 ภาพ

```python
def undersample(
        split_set: list,
        total_ratio: float = None,
        total_size: int = None,
        null_ratio: float = None,

) -> list:

    if total_ratio is None and total_size is None:
        raise ValueError("You must specify either 'total_ratio' or 'total_size'.")

    if total_ratio is not None and total_size is not None:
        raise ValueError("Specify only one of 'total_ratio' or 'total_size', not both.")

    random.shuffle(split_set)

    if total_ratio:
        total_size = int(len(split_set) * total_ratio)

    if null_ratio is not None:
        null_set = [item for item in split_set if len(item["objects"]) == 0]
        pos_set = [item for item in split_set if len(item["objects"]) > 0]

        null_size = int(total_size * null_ratio)
        pos_size = total_size - null_size

        sampled_set = pos_set[:pos_size] + null_set[:null_size]

    else:
        sampled_set = split_set[:total_size]

    return sampled_set
```

```python
def split_dataset(split_set: list, first_ratio: float = None, first_size: int = None) -> tuple:
    if first_ratio is None and first_size is None:
        raise ValueError("You must specify either 'first_ratio' or 'first_size'.")

    if first_ratio is not None and first_size is not None:
        raise ValueError("Specify only one of 'first_ratio' or 'first_size', not both.")

    random.shuffle(split_set)

    if first_ratio:
        first_size = int(len(split_set) * first_ratio)

    first_split = split_set[:first_size]
    second_split = split_set[first_size:]

    return first_split, second_split
```

5.1.4. การแปลง Bounding Box ให้อยู่ในรูปแบบ YOLO

ฟังก์ชัน *normalize_bbox()* ทำการแปลงพิกัดจาก COCO [x_min, y_min, width, height] ให้เป็นรูปแบบ YOLO <class_id> <x_center_norm> <y_center_norm> <w_norm> <h_norm> โดย normalize ค่าให้อยู่ระหว่าง 0–1 ตามขนาดภาพ

```python
def normalize_bbox(bbox: list, image_width: int, image_height: int) -> list:
    x, y, w, h = bbox

    x_center = (x + w / 2) / image_width
    y_center = (y + h / 2) / image_height
    w_norm = w / image_width
    h_norm = h / image_height

    normalized_bbox = [x_center, y_center, w_norm, h_norm]

    return normalized_bbox
```

5.1.5. การสร้างโฟลเดอร์ Dataset สำหรับ YOLO

ใช้ฟังก์ชัน *create_dataset()* เพื่อสร้างโครงสร้างโฟลเดอร์และไฟล์สำหรับเทรนโมเดล YOLO ได้แก่

```
dataset/
├── images/{train,val,test}/
└── labels/{train,val,test}/
```

ในแต่ละภาพจะมีไฟล์ .txt ที่เก็บ annotation ในรูปแบบ YOLO พร้อมใช้งานกับ Ultralytics ได้ทันที

```python
def create_dataset(source_dir: str, dest_dir: str, training_set: list, validation_set: list, testing_set: list) -> None:
    source_dir = Path(source_dir)
    dataset_dir = Path(dest_dir)
    images_dir = dataset_dir / "images"
    labels_dir = dataset_dir / "labels"

    splits = [
        ("train", training_set),
        ("val", validation_set),
        ("test", testing_set),
    ]

    for split_name, split_set in splits:
        split_images_dir = images_dir / split_name
        split_labels_dir = labels_dir / split_name

        split_images_dir.mkdir(parents = True, exist_ok = True)
        split_labels_dir.mkdir(parents = True, exist_ok = True)

        for item in tqdm(split_set, desc = f"Creating {split_name} set", unit = "image", ncols = 1000):
            filename = item["image"]["filename"]

            source_image_path = source_dir / filename
            dest_image_path = split_images_dir / filename

            copyfile(source_image_path, dest_image_path)

            label_filename = Path(filename).with_suffix(".txt").name
            label_path = split_labels_dir / label_filename

            with open(label_path, mode = "w", encoding = "utf-8") as f:
                image_width = item["image"]["width"]
                image_height = item["image"]["height"]

                for annotation in item["objects"]:
                    class_id = annotation["class_id"]
                    bbox = annotation["bbox"]
                    bbox = normalize_bbox(bbox, image_width, image_height)
                    bbox_str = " ".join(map(str, bbox))

                    f.write(f"{class_id - 1} {bbox_str}\n")
```

5.1.6 การเปลี่ยนชื่อไฟล์ (Rename)

สุดท้ายใช้ฟังก์ชัน *rename_files()* เพื่อเปลี่ยนชื่อไฟล์ตามระดับความยาก เช่น xray_easy_00001.png, xray_hard_00002.png, xray_hidden_00003.png เพื่อให้ง่ายต่อการอ้างอิงและวิเคราะห์ภายหลัง

```python
def rename_files(images_dir: str, labels_dir: str) -> None:
    images_dir = Path(images_dir)
    labels_dir = Path(labels_dir)

    for split in ["train", "val", "test"]:
        split_images_dir = images_dir / split
        split_labels_dir = labels_dir / split

        image_paths = sorted(list(split_images_dir.iterdir()))

        easy_count, hard_count, hidden_count = 0, 0, 0

        for idx, image_path in enumerate(tqdm(image_paths, desc = f"Renaming files in {split} set", unit = "file", ncols = 1000)):
            label_filename = image_path.with_suffix(".txt").name
            label_path = split_labels_dir / label_filename

            if "easy" in image_path.name:
                new_name = f"xray_easy_{easy_count:05d}"
                easy_count += 1
            elif "hard" in image_path.name:
                new_name = f"xray_hard_{hard_count:05d}"
                hard_count += 1
            elif "hidden" in image_path.name:
                new_name = f"xray_hidden_{hidden_count:05d}"
                hidden_count += 1
            else:
                new_name = f"xray_{idx:05d}"

            new_label_filename = new_name + ".txt"
            new_label_path = split_labels_dir / new_label_filename

            label_path.rename(new_label_path)

            new_filename = new_name + ".png"
            new_image_path = split_images_dir / new_filename

            image_path.rename(new_image_path)
```

## 5.2 การฝึกโมเดล (*train.ipynb*)

5.2.1 การติดตั้งและนำเข้าไลบรารี

เริ่มจากติดตั้งไลบรารี Ultralytics ซึ่งเป็นเฟรมเวิร์กสำหรับการใช้งาน YOLO จากนั้น import ไลบรารีพื้นฐานที่จำเป็น เช่น cv2, matplotlib, และ pathlib.Path เพื่อใช้แสดงผลและจัดการพาธไฟล์ รวมถึงมีการ mount Google Drive เพื่อนำ dataset มาใช้ฝึกโมเดล และเก็บผลลัพธ์ของการเทรนโมเดลในภายหลัง

5.2.2 การกำหนดพาธข้อมูล (*data.yaml*)

ไฟล์ *data.yaml* ช่วยให้โมเดลรู้ตำแหน่งของข้อมูลสำหรับการฝึก ตรวจสอบ และทดสอบอย่างเป็นระบบ โดยใช้เพื่อระบุพาธของชุดข้อมูลและชื่อคลาสทั้งหมด 12 คลาส ดังนี้

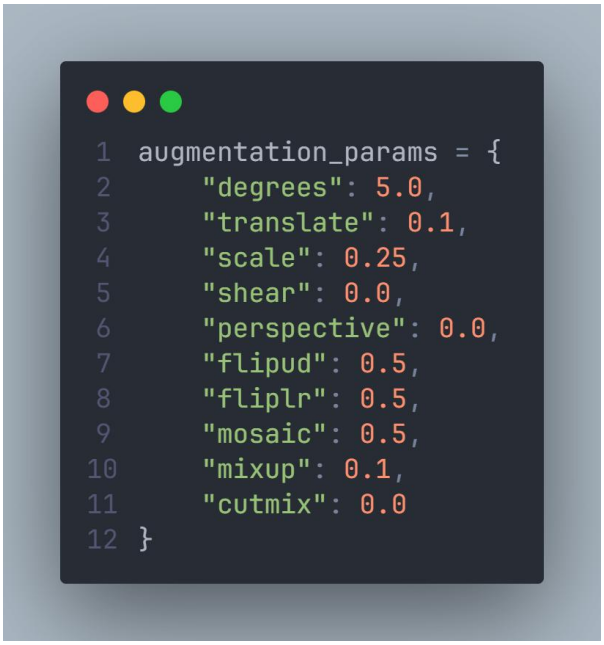

5.2.3 การโหลดโมเดลเริ่มต้น (Pretrained Model)

โหลดโมเดล YOLOv11s ที่ผ่านการฝึกบน dataset COCO มาก่อนแล้ว จากนั้นนำมาทำ Fine-tuning กับชุดข้อมูล X-ray ที่จัดเตรียมไว้

5.2.4 การตั้งค่าการฝึกสอน (Training Parameters)

กำหนดค่าพารามิเตอร์หลักที่ใช้ในการฝึก เช่น จำนวนรอบการฝึก (epochs), ขนาดภาพ (image size), การใช้ early stopping (patience), และการเปิดใช้ cosine learning rate schedule เพื่อปรับค่าเรียนรู้อย่างต่อเนื่อง นอกจากนี้ยังมีการเปิดใช้ data augmentation เช่น การกลับภาพ (flip), mosaic, และ mixup เพื่อเพิ่มความหลากหลายของข้อมูลและช่วยลดการ overfitting ของโมเดล

```python
training_params = {
    "project": project_dir / "runs",
    "name": get_timestamp(),
    "data": "/content/data.yaml",
    "epochs": 1000,
    "patience": 5,
    "batch": -1,
    "imgsz": 512,
    "cos_lr": True,
    "save": True,
    "save_period": 10,
    "cache": True,
    "seed": 42
}
```



```python
augmentation_params = {
    "degrees": 5.0,
    "translate": 0.1,
    "scale": 0.25,
    "shear": 0.0,
    "perspective": 0.0,
    "flipud": 0.5,
    "fliplr": 0.5,
    "mosaic": 0.5,
    "mixup": 0.1,
    "cutmix": 0.0
}
```

5.2.5 การเริ่มฝึกโมเดล (Training Process)

เริ่มฝึกสอนโมเดล YOLOv11s โดยจะโหลดน้ำหนักจาก pretrained model จากนั้นปรับจำนวนคลาสเป็น 12 คลาส และเริ่มทำการ fine-tune ตามชุดข้อมูลที่กำหนด ระหว่างการฝึก โมเดลจะบันทึกผลลัพธ์และค่าประสิทธิภาพในแต่ละ epoch ไว้ในโฟลเดอร์ runs/


5.2.6 ผลลัพธ์หลังการฝึกสอน

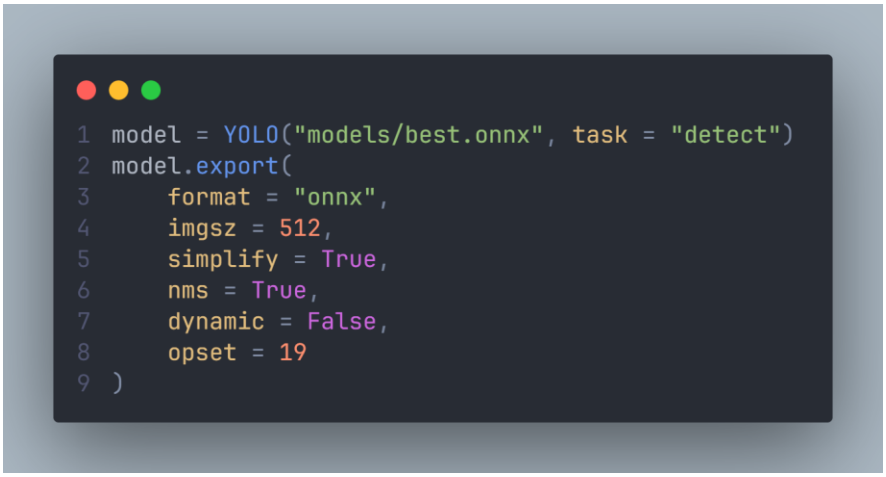
หลังการฝึกสอนเสร็จสิ้น จะได้ไฟล์น้ำหนักโมเดลและค่าประเมินประสิทธิภาพ ได้แก่

- best.pt (โมเดลที่มีประสิทธิภาพสูงสุด) และ last.pt (โมเดลล่าสุด)
- เมตริกสรุปผล เช่น Precision, Recall, mAP@50, และ mAP@50–95
- กราฟแสดงค่า loss และ precision–recall curve เพื่อวิเคราะห์ผลการฝึก

## 5.3 การประเมิลผลโมเดล (*evaluate.ipynb*)

5.3.1 การแปลงและโหลดโมเดล

หลังการฝึกสอน โมเดลถูกแปลงเป็น ONNX format เพื่อให้ inference ได้รวดเร็วและรองรับการรันข้ามแพลตฟอร์ม จากนั้นโหลดกลับมาทดสอบด้วย



```python
model = YOLO("models/best.onnx", task = "detect")
model.export(
    format = "onnx",
    imgsz = 512,
    simplify = True,
    nms = True,
    dynamic = False,
    opset = 19
)
```

5.3.2 การประเมินเชิงภาพ (Qualitative Evaluation)

โมเดลถูกนำมาทดสอบกับชุดภาพจาก dataset/images/test โดยใช้คลาส ProhibitedDetector เพื่อพยากรณ์ตำแหน่งและคลาสของวัตถุ

ผลลัพธ์แต่ละภาพจะแสดง bounding box, label, และ confidence score เพื่อให้ตรวจสอบความถูกต้องเชิงสายตา เช่น hammer, lighter, pliers, gun เป็นต้น

5.3.3 การจัดชุดทดสอบตามระดับความยาก

ข้อมูลทดสอบถูกแยกเป็น 3 กลุ่ม เพื่อให้สามารถประเมินความสามารถของโมเดลในแต่ละเงื่อนไขได้อย่างละเอียด โดยแบ่งได้ดังนี้

- Easy: มีวัตถุต้องห้ามเพียงหนึ่งชิ้นในภาพ
- Hard: มีวัตถุต้องห้ามหลายชิ้นในภาพเดียว
- Hidden: มีวัตถุต้องห้ามที่ถูกซ่อนหรือดัดแปลงรูปร่างให้ตรวจจับได้ยาก

5.3.4 การประเมินเชิงปริมาณ (Quantitative Evaluation)

วัดประสิทธิภาพของโมเดลในแต่ละชุดทดสอบ (easy, hard, hidden และ overall) โดยผลลัพธ์ถูกจัดสรุปออกเป็น 2 ตารางหลัก คือ

1. ตารางระดับชุดข้อมูล (Dataset-level): สรุปภาพรวมของแต่ละ subset เช่น จำนวนภาพ, จำนวนวัตถุ (instances), และค่าเมตริกสำคัญ ได้แก่ mAP@50, mAP@75, และ mAP@50–95 ซึ่งใช้วัดความแม่นยำโดยรวมของโมเดลในแต่ละระดับความยาก
2. ตารางระดับคลาส (Class-level): แสดงผลลัพธ์แยกตามวัตถุทั้ง 12 คลาส โดยมีค่า Precision, Recall, F1-score, mAP50, และ mAP50–95 เพื่อประเมินว่าโมเดลสามารถตรวจจับวัตถุแต่ละประเภทได้ดีเพียงใด




## 6. ลิงค์ไปยัง Repository

พัฒนาโมเดล: https://github.com/chavalvit-k/prohibited-item-detection

Backend: https://gitlab.com/chavalvit.keart/prohibited-item-detection-api

Frontend: https://github.com/Lucky-TP/X-ray-Prohibited-Item-Detection

## 7. อธิบายการฝึกโมเดล

ในการฝึกสอนโมเดล โครงงานนี้ใช้แนวทาง Transfer Learning (Fine-tuning) แทนการเทรนจากศูนย์ (from scratch) โดยเริ่มจากโมเดล YOLOv11s ที่ผ่านการฝึกมาก่อนบนชุดข้อมูล COCO Dataset ซึ่งมีวัตถุ 80 คลาส โมเดลนี้จึงมีพื้นฐานความเข้าใจเกี่ยวกับรูปร่าง ขอบเขต และลักษณะทั่วไปของวัตถุอยู่แล้ว จากนั้นเรานำมาปรับเพิ่มเติม (fine-tune) ให้จำแนกเฉพาะ 12 คลาสของวัตถุต้องห้ามในภาพ X-ray

ก่อนจะเริ่มกระบวนการฝึกสอนโมเดล เราจำเป็นต้องมีการตั้งค่า Hyperparameters เพื่อกำหนดทิศทางและลักษณะการเรียนรู้ของโมเดล เช่น จะเรียนรู้เร็วแค่ไหน เรียนกี่รอบ หรือจะเพิ่มความหลากหลายของข้อมูลอย่างไร เพราะโมเดล deep learning ไม่สามารถตัดสินใจสิ่งเหล่านี้ได้ด้วยตัวเอง หากตั้งค่าไม่เหมาะสม เช่น learning rate สูงเกินไป อาจทำให้โมเดลเรียนรู้ผิดพลาดหรือไม่เสถียร ในทางกลับกัน หากตั้งค่าต่ำเกินไปหรือรอบฝึกน้อยเกินไป โมเดลอาจไม่สามารถเรียนรู้ได้เต็มที่

นอกจากพารามิเตอร์ที่ควบคุมการฝึกโดยตรงแล้ว ยังมีส่วนของ augmentation parameters ที่ใช้ในการปรับแต่งข้อมูลก่อนส่งเข้าโมเดล เช่น การหมุน การกลับภาพ การซ้อนภาพหลายใบ (mosaic) หรือการผสมภาพ (mixup) ซึ่งช่วยให้ข้อมูลฝึกมีความหลากหลายมากขึ้น ทำให้โมเดลเรียนรู้ได้ robust และสามารถ generalize ได้ดีขึ้นกับภาพที่ไม่เคยเห็นมาก่อน การกำหนด hyperparameters ทั้งสองส่วนนี้จึงเป็นเหมือน “กติกาในการเรียนรู้” ที่ช่วยให้โมเดลสามารถปรับน้ำหนักได้อย่างมีประสิทธิภาพ และให้ผลลัพธ์ที่แม่นยำสูงสุดในการตรวจจับวัตถุในภาพ X-ray

| ชื่อพารามิเตอร์ | ค่าที่ใช้ | คำอธิบาย |
|---|---|---|
| epochs | 1000 | จำนวนรอบสูงสุดของการฝึก เพื่อให้โมเดลได้เรียนรู้ครบทุกข้อมูล โดยมี Early Stopping ช่วยหยุดอัตโนมัติเมื่อผลไม่ดีขึ้น |
| patience | 5 | จำนวนรอบที่อนุญาตให้ performance ไม่ดีขึ้นก่อนหยุดการฝึก ป้องกัน overfitting และประหยัดเวลา |
| imgsz | 512 | ขนาดภาพที่ใช้ระหว่างการฝึก (512×512 px) ใกล้เคียงกับความละเอียดเฉลี่ยของภาพในชุดข้อมูล PIDray (≈500×500 px) และเป็นค่าที่หารด้วย 32 ลงตัวตามโครงสร้างของ YOLO ทำให้รักษารายละเอียดของวัตถุได้ดีและประมวลผลได้มีประสิทธิภาพ |
| batch | -1 | ให้ Ultralytics คำนวณ batch size อัตโนมัติตามหน่วยความจำของ GPU เพื่อใช้ทรัพยากรได้เต็มประสิทธิภาพ |
| cos_lr | True | เปิดใช้ Cosine Learning Rate Schedule เพื่อให้ learning rate ลดลงอย่างราบรื่น ช่วยให้โมเดลเสถียรในช่วงท้ายของการฝึก |
| seed | 42 | กำหนดค่า seed สำหรับการสุ่ม เพื่อให้สามารถทำซ้ำผลลัพธ์ได้อย่างสม่ำเสมอ |
| flipud / fliplr | 0.5 | กลับภาพในแนวตั้งและแนวนอนครึ่งหนึ่งของข้อมูล เพื่อให้โมเดลเข้าใจมุมมองที่หลากหลาย |
| mosaic | 0.5 | รวมภาพ 4 ภาพเข้าด้วยกัน เพิ่มความหลากหลายของฉากและการซ้อนทับวัตถุ เหมาะกับงานตรวจจับวัตถุหลายชิ้น |
| mixup | 0.1 | ผสมภาพสองภาพเข้าด้วยกัน เพื่อช่วยให้โมเดล generalize ได้ดีขึ้นและลดการ overfitting |
| degrees | 5.0 | หมุนภาพในช่วง ±5 องศา เพื่อจำลองความหลากหลายของมุมมองในสภาพจริง |
| translate | 0.1 | ขยับภาพเล็กน้อยในแนวแกน X/Y เพื่อเพิ่มความทนทานต่อการเคลื่อนตำแหน่งของวัตถุ |
| scale | 0.25 | ปรับขนาดวัตถุแบบสุ่มในช่วง ±25% เพื่อให้โมเดลเข้าใจวัตถุหลายขนาด |

## 8. อธิบายชุดข้อมูล

PIDray dataset: https://bit.ly/3X7xdMW

YOLOv11 dataset: https://bit.ly/3WzniQ6

ในโครงงานนี้ได้ใช้ชุดข้อมูล PIDray (Prohibited Item Detection Dataset) ซึ่งเป็นชุดข้อมูลมาตรฐานสำหรับการตรวจจับวัตถุต้องห้ามในภาพ X-ray โดยจัดทำโดยทีมวิจัยจาก Tsinghua University ชุดข้อมูลนี้มีภาพ X-ray รวมทั้งหมด 124,486 ภาพ แบ่งออกเป็นชุด Training 76,913 ภาพ และ Testing 47,573 ภาพ ซึ่งในส่วนของ Testing ยังแบ่งย่อยตามระดับความยากเป็น 3 กลุ่ม คือ

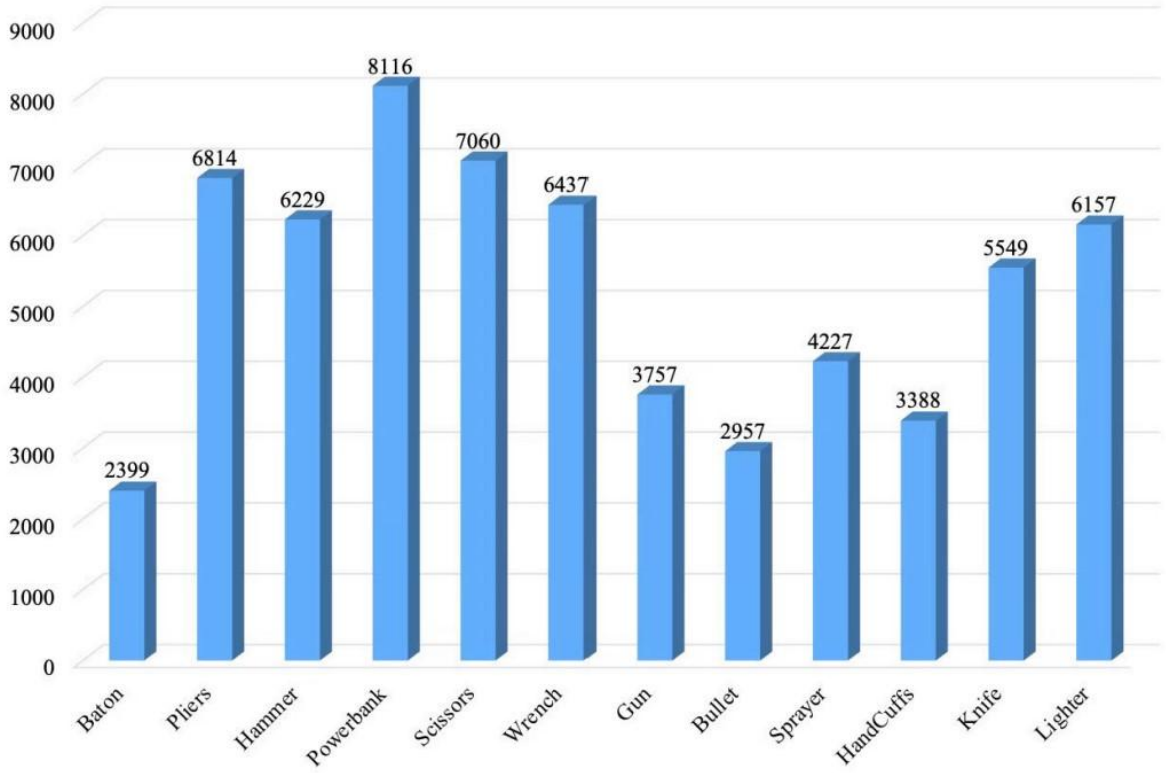
- Easy: ภาพที่มีวัตถุต้องห้ามเพียงชิ้นเดียว
- Hard: ภาพที่มีวัตถุต้องห้ามหลายชิ้น
- Hidden: ภาพที่มีวัตถุต้องห้ามที่ถูกซ่อนหรือบังอยู่บางส่วน

| Mode | Train | Test | | |
|---|---|---|---|---|
| | | Easy | Hard | Hidden |
| Count | 76,913 | 24,758 | 9,746 | 13,069 |
| Total | 124,486 | | | |

*ตารางแสดงจำนวนข้อมูลตามระดับความยาก*

แต่เนื่องจากชุดข้อมูลเต็มมีขนาดใหญ่มาก (กว่า 120,000 ภาพ) อาจทำให้ใช้เวลาในการฝึกโมเดลที่นานมาก และอาจเกินขีดจำกัดของหน่วยความจำ GPU ที่ใช้ในการฝึกสอนโมเดล ดังนั้นในการเทรนจริงจึงได้ทำการ สุ่มตัวอย่าง (sampling) ลดขนาดลงมาเหลือ 40,000 ภาพ โดยยังคงสัดส่วนของแต่ละกลุ่ม (easy, hard, hidden) ให้ใกล้เคียงกับต้นฉบับ เพื่อให้ได้ทั้งความสมดุลและประสิทธิภาพในการฝึก

นอกจากนี้ ในชุดข้อมูล ยังประกอบด้วย 12 คลาสของวัตถุต้องห้าม ได้แก่ baton, pliers, hammer, powerbank, scissors, wrench, gun, bullet, sprayer, handcuffs, knife, lighter โดยมีการกระจายของจำนวนวัตถุในแต่ละคลาสแตกต่างกันดังกราฟด้านล่าง ซึ่งแสดงให้เห็นถึงความไม่สมดุลของคลาส (class imbalance) — เช่น “powerbank” และ “scissors” มีจำนวนมากกว่า “baton” หรือ “bullet” อย่างมีนัยสำคัญ




*ตารางแสดงการกระจายตัวของแต่ละคลาสในชุดข้อมูล*

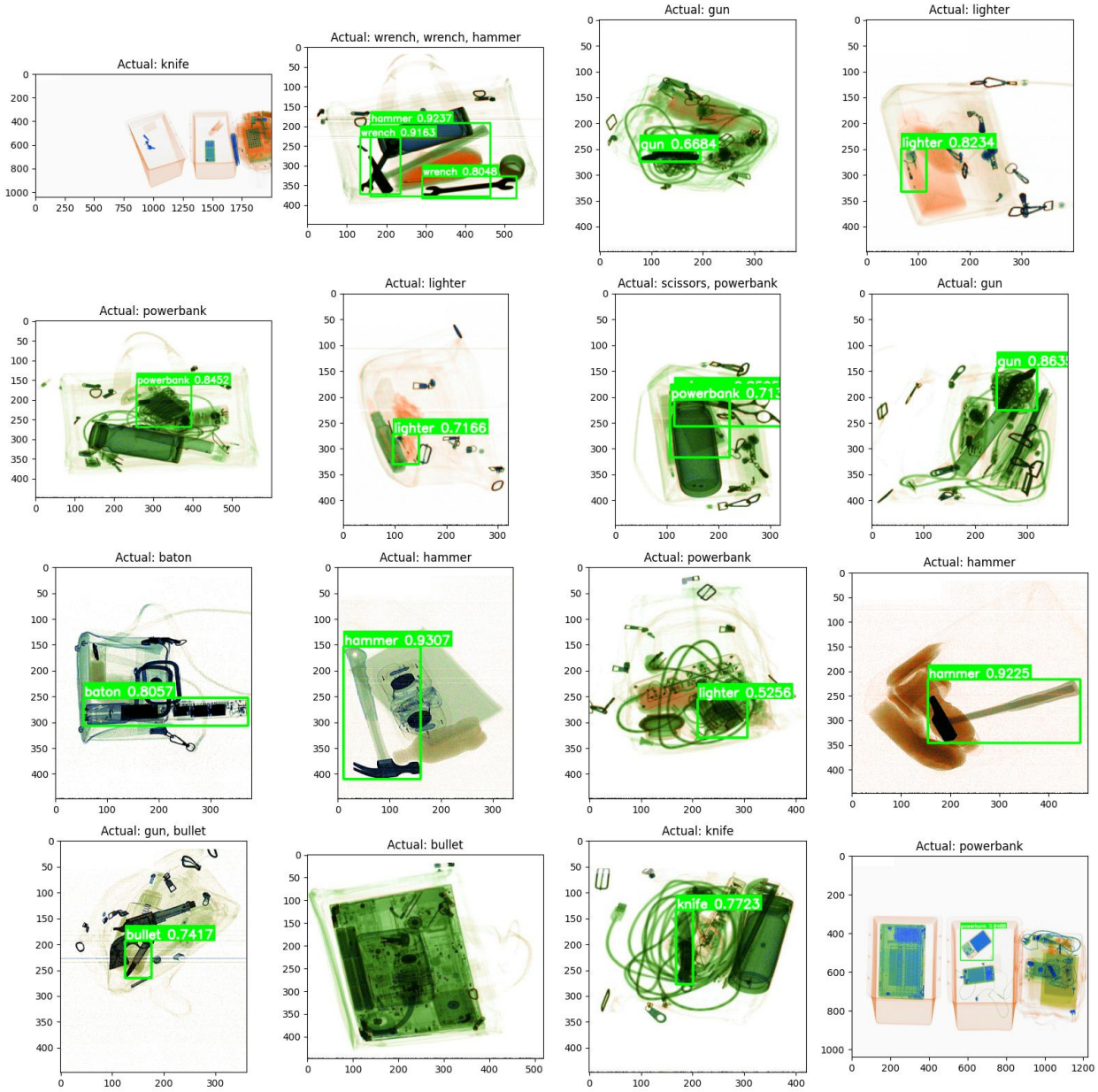
ก่อนนำข้อมูลมาใช้ ได้มีการแปลง annotation จากรูปแบบ COCO ([x_min, y_min, width, height]) ให้อยู่ในรูปแบบ YOLO format (<class_id> <x_center> <y_center> <width> <height>) เพื่อให้เข้ากับโมเดล YOLOv11s จากนั้นจึงแบ่งข้อมูลออกเป็น Training / Validation / Test sets และทำ undersampling บางส่วนของภาพที่ไม่มีวัตถุ เพื่อเพิ่มความสมดุลของข้อมูลและลดเวลาในการประมวลผล

โดยสรุป ชุดข้อมูล PIDray มีจุดเด่นคือความใกล้เคียงกับสภาพแวดล้อมจริงในสนามบินและสถานีขนส่ง ความหลากหลายของฉาก และระดับความยากที่ท้าทาย ซึ่งเหมาะสำหรับใช้ประเมินความสามารถของโมเดลตรวจจับวัตถุ (Object Detection) อย่าง YOLOv11s ได้อย่างครอบคลุม

## 9. อธิบายการประเมินผลโมเดล

### 9.1 การประเมินเชิงคุณภาพ (Qualitative Evaluation)

การประเมินเชิงคุณภาพทำโดยการสุ่มภาพจากชุด Test Set มาทดลองทำนายผล ซึ่งจะแสดง bounding box และชื่อคลาสของวัตถุที่ตรวจพบพร้อมค่าความมั่นใจ (confidence score) ดังภาพด้านล่าง




*ภาพกลุ่มตัวอย่าง 16 รูปที่ลองทำการประเมินเชิงคุณภาพ*

จากภาพจะเห็นว่าโมเดลสามารถตรวจจับวัตถุได้อย่างถูกต้องและแม่นยำเป็นส่วนใหญ่ แม้ในกรณีที่วัตถุมีการซ้อนทับหรือมีสิ่งของอื่นบดบังบางส่วน

## 9.2 การประเมินเชิงปริมาณ (Quantitative Evaluation)

ในส่วนนี้ได้ทำการประเมินผลโดยใช้ Mean Average Precision (mAP) ซึ่งเป็น metric มาตรฐานในงาน Object Detection

- mAP50หมายถึงค่าเฉลี่ยของความแม่นยำเมื่อ IoU ≥ 0.5
- mAP50-95 หมายถึงค่าเฉลี่ยของ mAP ที่คำนวณจาก IoU ตั้งแต่ 0.5 ถึง 0.95 (ช่วงละเอียดกว่าและเข้มงวดกว่า)

9.2.1. ภาพรวมของผลการทดสอบ (Overall Performance)

โมเดลถูกทดสอบกับชุดข้อมูล 4 กลุ่ม ได้แก่ easy, hard, hidden และ overall โดยผลลัพธ์สรุปในตารางด้านล่าง

| Dataset | Images | Instances | mAP50 | mAP75 | mAP50-95 |
|---|---|---|---|---|---|
| easy | 987 | 987 | 83.76 | 81.5 | 77.32 |
| hard | 825 | 950 | 87.52 | 82.06 | 76.34 |
| hidden | 506 | 506 | 76.1 | 72.61 | 65.42 |
| overall | 2318 | 2443 | 84.66 | 81.16 | 75.63 |

*ตารางแสดงภาพรวมของการทดสอบ*

จากตารางจะเห็นว่าโมเดลทำผลงานได้ดีในกลุ่ม easy และ hard โดยมีค่า mAP50 เฉลี่ยอยู่ที่ 84.66% และค่า mAP50–95 ที่ 75.63% ขณะที่กลุ่ม hidden ซึ่งเป็นกรณีที่วัตถุถูกซ่อนหรือบังบางส่วน มีความยากสูง จึงได้ค่าความแม่นยำต่ำกว่าที่ประมาณ 65.4%

9.2.2 ประสิทธิภาพรายคลาส (Per-Class Performance)

เพื่อวิเคราะห์ความสามารถของโมเดลในแต่ละประเภทของวัตถุต้องห้าม ได้มีการคำนวณ Precision, Recall, F1-score และค่า mAP แยกตามคลาส ดังตารางด้านล่าง

| Class | Images | Instances | Precision | Recall | F1 | mAP50 | mAP50-95 |
|---|---|---|---|---|---|---|---|
| baton | 83 | 86 | 80.43 | 87.21 | 83.68 | 90.12 | 81.36 |
| pliers | 261 | 272 | 94.64 | 95.59 | 95.11 | 97.23 | 89.96 |
| hammer | 260 | 270 | 97.52 | 87.38 | 92.17 | 94.1 | 89.06 |
| powerbank | 294 | 311 | 86.18 | 89.71 | 87.91 | 94.5 | 80.68 |
| scissors | 259 | 293 | 76.97 | 89.08 | 82.58 | 90.41 | 79.68 |
| wrench | 217 | 226 | 94.4 | 96.9 | 95.63 | 97.72 | 92.15 |
| gun | 151 | 165 | 81.05 | 28.51 | 42.19 | 54.2 | 43.08 |
| bullet | 129 | 130 | 76.24 | 70 | 72.99 | 76.94 | 67.39 |
| sprayer | 113 | 116 | 95.41 | 71.66 | 81.85 | 87.95 | 77.39 |
| handcuffs | 139 | 139 | 98.61 | 98.56 | 98.58 | 99.08 | 93.69 |
| knife | 214 | 237 | 70.24 | 38.83 | 50.02 | 58.59 | 48.24 |
| lighter | 198 | 198 | 72.85 | 68.69 | 70.71 | 75.02 | 64.9 |

*ตารางแสดงผลการทดสอบแยกตามคลาส*

ผลลัพธ์แสดงให้เห็นว่าโมเดลสามารถตรวจจับคลาสที่มีลักษณะชัดเจน เช่น handcuffs, pliers, wrench ได้ดีมาก โดยมีค่า Precision และ Recall มากกว่า 95% ในขณะที่คลาสที่มีรูปร่างเล็กหรือมักซ่อนอยู่ในวัตถุอื่น เช่น gun และ knife มีค่า Recall ต่ำกว่า เนื่องจากการตรวจจับทำได้ยากกว่าในภาพ X-ray

โดยสรุป โมเดล YOLOv11s ที่ผ่านการปรับ fine-tune บน PIDray dataset แสดงให้เห็นถึงความสามารถในการตรวจจับวัตถุต้องห้ามได้แม่นยำและสอดคล้องกับระดับความยากของข้อมูลจริงในสนามบิน

## 10. บทความอ้างอิงและงานที่เกี่ยวข้อง

### 10.1 ชุดข้อมูลและแนวคิดพื้นฐาน

[1] L. Zhang, L. Jiang, R. Ji, H. Fan, “A Large-scale X-ray Benchmark for Real-World Prohibited Item Detection (PIDray),” *arXiv preprint arXiv:2211.10763*, 2022.

- Link: https://arxiv.org/pdf/2211.10763
- Reference หลักสำหรับ Dataset (PIDray) ที่เราใช้ train และทดสอบโมเดลทั้งหมดในโครงงานนี้

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 779-788, 2016.

- Link: https://arxiv.org/pdf/1506.02640.pdf
- ใช้อ้างอิงถึงแนวคิดดั้งเดิมของสถาปัตยกรรม YOLO (You Only Look Once) ซึ่งเป็นพื้นฐานของโมเดลที่เราเลือกใช้

[3] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement,” *arXiv preprint arXiv:1804.02767*, 2018.

- Link: https://arxiv.org/pdf/1804.02767.pdf
- ใช้อ้างอิงการพัฒนาที่สำคัญของสถาปัตยกรรม YOLO (เช่น การใช้ multi-scale features) ซึ่งเป็นแนวคิดที่ถูกนำมาพัฒนาต่อในโมเดล YOLOv11s ที่เราใช้

### 10.2 เฟรมเวิร์ก ไลบรารี และเอกสารอ้างอิง

[4] G. Jocher, A. Chaurasia, and J. Qiu, “YOLO by Ultralytics (Version 8.x),” [Software], 2023.

- Link: https://github.com/ultralytics/ultralytics
- นี่คือ Library/Framework หลัก (Ultralytics) ที่เราใช้ในการ train และ fine-tuning โมเดล YOLOv11s ของเรา (ตามที่ระบุใน *train.ipynb*)

[5] Ultralytics, “YOLO Data Augmentation Guide,” [Online Documentation], 2024.

- Link: https://docs.ultralytics.com/guides/yolo-data-augmentation/
- ใช้อ้างอิงเทคนิค Data Augmentation (เช่น mosaic, mixup) ที่เรากำหนดค่าใน augmentation_params ในไฟล์ *train.ipynb*

[6] S. Tiangolo (S. Ramírez), “FastAPI,” [Software], 2018.

- Link: https://fastapi.tiangolo.com/
- ใช้อ้างอิง FastAPI Framework ที่เราใช้สร้าง Backend API สำหรับการ deploy โมเดล (ตามที่ระบุใน Endpoint prohibited-item-detection-api)

[7] E. You, et al., “Vite: Next Generation Frontend Tooling,” [Software], 2020.

- Link: https://vitejs.dev/
- ใช้อ้างอิง Vite ซึ่งเป็นเครื่องมือ Frontend build tool ที่เราใช้สำหรับพัฒนา Website (x-ray-detection.vercel.app)

## 11. สัดส่วนของงาน

| หมวดงานหลัก | ประเภทงานย่อย | สัดส่วนงาน (%) | ผู้รับผิดชอบ |
|---|---|---|---|
| พัฒนาโมเดล | เตรียมชุดข้อมูล | 10 | ธัญชนก |
| | ฝึกโมเดล | 10 | ชวัลวิทย์ |
| | ประเมินประสิทธิภาพของโมเดล | 10 | ชวัลวิทย์ |
| Backend | ออกแบบ API | 5 | ชวัลวิทย์ |
| | พัฒนา Backend | 20 | ชวัลวิทย์ |
| | Deployment | 5 | ชวัลวิทย์ |
| Frontend | ออกแบบ UX/UI | 5 | ธัญชนก |
| | พัฒนา Frontend | 20 | ธัญชนก |
| | Deployment | 5 | ธัญชนก |
| รายงาน | | 10 | ธัญชนก |
| รวมทั้งหมด | | 100 | |