Safety Helmet Detection

Using YOLOv8

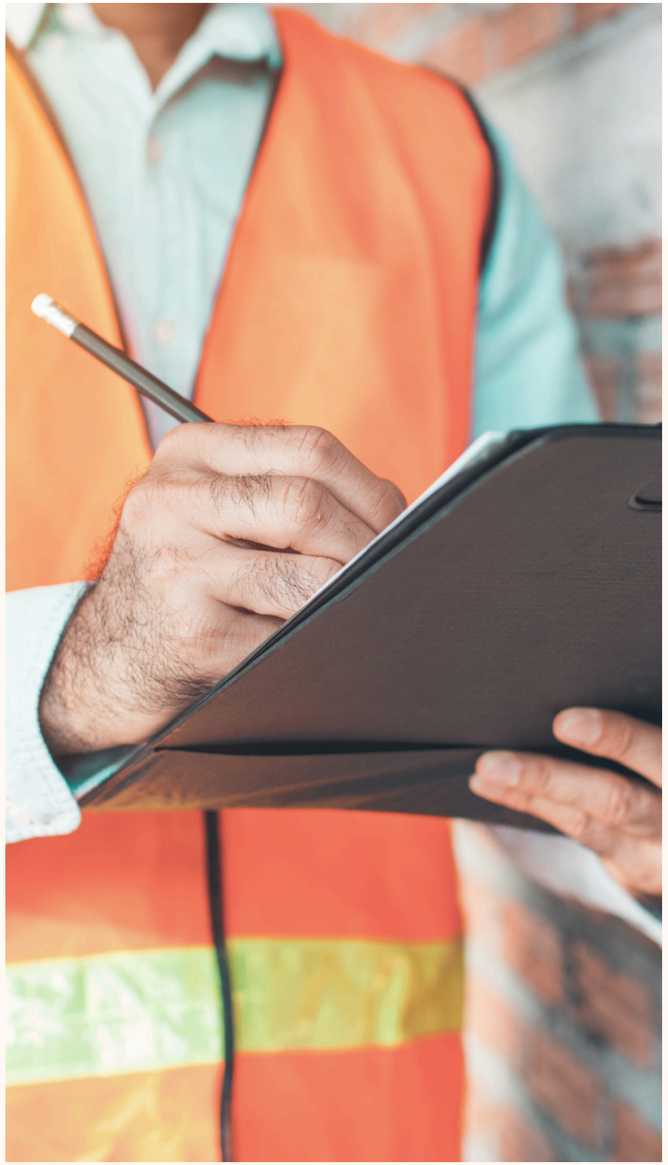
Noor Ali Fadhil
Deep Learning Project

Introduction

This project aims to improve workplace safety by using deep learning and computer vision techniques. A YOLOv8-based helmet detection system was developed to automatically detect whether workers are wearing safety helmets in real time. The system also includes a graphical user interface for easy visualization and interaction.

Problem Statement

- Workers often fail to comply with safety helmet regulations
- Workplace accidents remain a serious safety concern
- There is a strong need for an automated and intelligent monitoring system



Dataset

- Dataset Name: Safety Helmet Wearing Dataset
- Source: Public dataset (Kaggle)
- Total Images: ~5,000 images
- Image Type: RGB images of construction sites

◆ Classes
- Helmet
- Head (No Helmet)
- Person

◆ Annotations
- Bounding box annotations
- YOLO format

◆ Data Split
- Training set: ~70%
- Validation set: ~30%

```python
import kagglehub

# Download latest version
path = kagglehub.dataset_download("andrewmvd/hard-hat-detection")

print("Path to dataset files:", path)
```

```
Downloading from https://www.kaggle.com/api/v1/datasets/download/andrewmvd/hard-hat-detection?dataset_version_number=1...
100%|██████████| 1.22G/1.22G [00:13<00:00, 101MB/s]Extracting files...

Path to dataset files: /root/.cache/kagglehub/datasets/andrewmvd/hard-hat-detection/versions/1
```

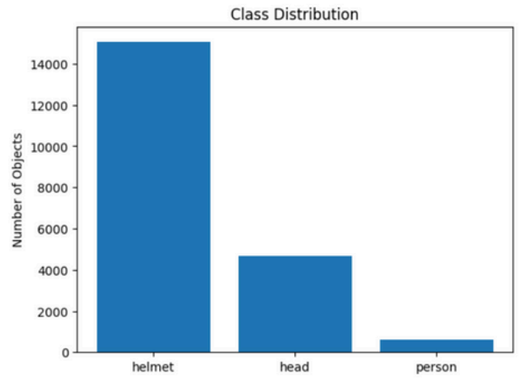
```python
import os

train_images = len(os.listdir("/content/helmet_yolo/images/train"))
val_images   = len(os.listdir("/content/helmet_yolo/images/val"))

print("Train images:", train_images)
print("Validation images:", val_images)
```

```
Train images: 4000
Validation images: 1000
```

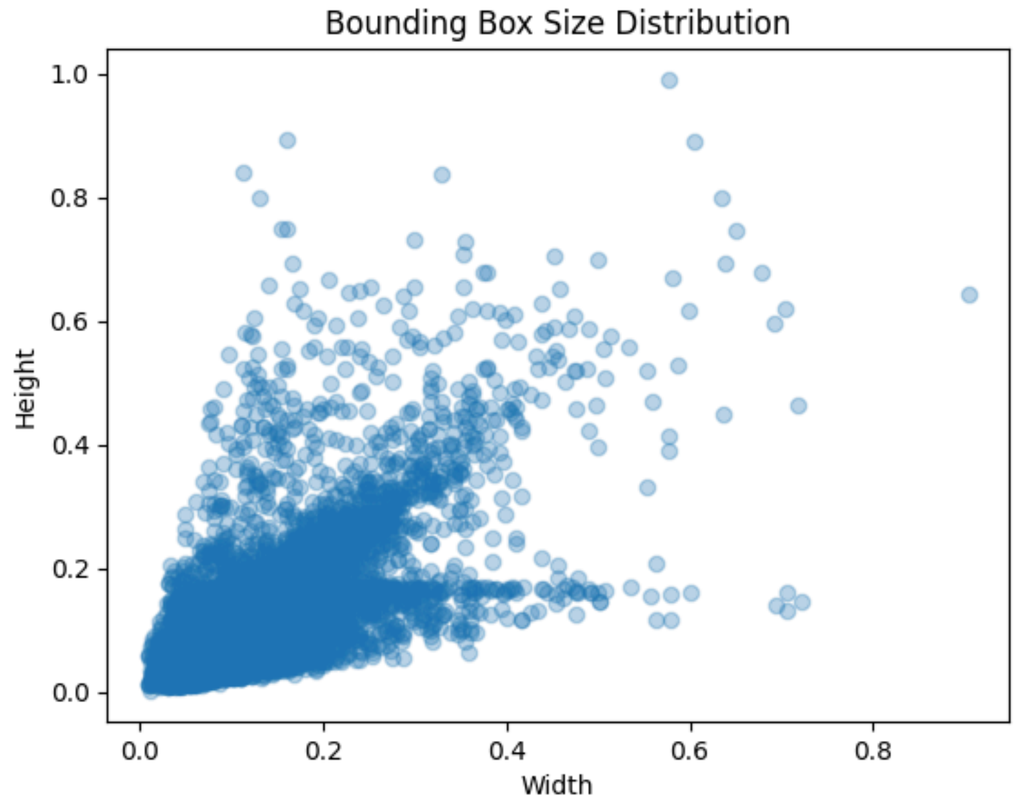
```python
from collections import Counter

label_dir = "/content/helmet_yolo/labels/train"
class_counts = Counter()

classes = {0:"helmet", 1:"head", 2:"person"}

for file in os.listdir(label_dir):
    with open(os.path.join(label_dir, file)) as f:
        for line in f:
            cls = int(line.split()[0])
            class_counts[cls] += 1

for k,v in class_counts.items():
    print(classes[k], ":", v)
```

```
helmet : 15050
head : 4686
person : 597
```

```python
import matplotlib.pyplot as plt

labels = [classes[k] for k in class_counts.keys()]
values = class_counts.values()

plt.bar(labels, values)
plt.title("Class Distribution")
plt.ylabel("Number of Objects")
plt.show()
```



```python
widths, heights = [], []

for file in os.listdir(label_dir):
    with open(os.path.join(label_dir, file)) as f:
        for line in f:
            _, x, y, w, h = map(float, line.split())
            widths.append(w)
            heights.append(h)

plt.scatter(widths, heights, alpha=0.3)
plt.xlabel("Width")
plt.ylabel("Height")
plt.title("Bounding Box Size Distribution")
plt.show()
```




```python
import cv2
from ultralytics.utils.plotting import Annotator
from google.colab.patches import cv2_imshow

img_path = "/content/helmet_yolo/images/train"
lbl_path = "/content/helmet_yolo/labels/train"

sample_img = os.listdir(img_path)[0]
img = cv2.imread(os.path.join(img_path, sample_img))
h, w, _ = img.shape

annotator = Annotator(img)

with open(os.path.join(lbl_path, sample_img.replace(".png", ".txt"))) as f:
    for line in f:
        cls, x, y, bw, bh = map(float, line.split())
        x1 = int((x - bw/2) * w)
        y1 = int((y - bh/2) * h)
        x2 = int((x + bw/2) * w)
        y2 = int((y + bh/2) * h)
        annotator.box_label([x1, y1, x2, y2], {0:"helmet",1:"head",2:"person"}[int(cls)])

cv2_imshow(img)
```

Model Architecture (YOLOv8)

- YOLOv8n (lightweight & fast)
- One-stage object detector
- Backbone + Neck + Head
- Advantages:
  - Real-time detection
  - High accuracy
  - Easy deployment

```
9]
49m
from ultralytics import YOLO

model = YOLO("yolov8n.pt")

model.train(
    data="/content/helmet_yolo/data.yaml",
    epochs=120,
    imgsz=640,
    batch=16,
    patience=5  # ← Early stopping

)
```

```
model = YOLO("yolov8n.pt")

model.train(
    data="/content/helmet_yolo/data.yaml",
    epochs=50,
    imgsz=640,
    batch=16,
    patience=5  # ← Early stopping

)
```

```
...       50/50      3.82G      1.108     0.5183      1.041        115        640: 100% ──────────
                   Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%
                     all       1000       4932      0.618      0.593      0.633      0.421

50 epochs completed in 1.403 hours.
Optimizer stripped from /content/runs/detect/train2/weights/last.pt, 6.2MB
Optimizer stripped from /content/runs/detect/train2/weights/best.pt, 6.2MB

Validating /content/runs/detect/train2/weights/best.pt...
Ultralytics 8.3.241 🚀 Python-3.12.12 torch-2.9.0+cu126 CUDA:0 (Tesla T4, 15095MiB)
Model summary (fused): 72 layers, 3,006,233 parameters, 0 gradients, 8.1 GFLOPs
                   Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%
                     all       1000       4932      0.618      0.593      0.633      0.421
                  helmet        925       3738      0.941      0.903      0.961      0.643
```

Training & Optimization

Confidence threshold ↓ (0.2)
IOU tuning
Test-Time Augmentation (TTA)
Focus on:
Maximizing Recall
Reducing missed detections

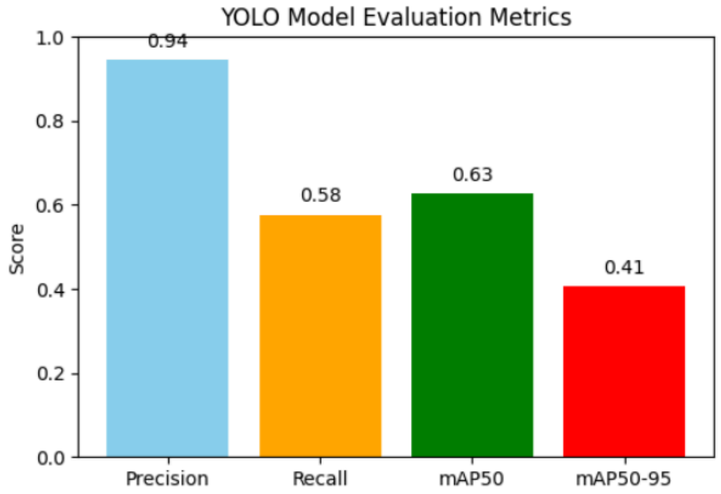
```python
import albumentations as A

# --- Paths ---
IMG_DIR = "/content/helmet_yolo/images/train"
LBL_DIR = "/content/helmet_yolo/labels/train"

OUT_IMG_DIR = "/content/helmet_yolo/images/train_aug"
OUT_LBL_DIR = "/content/helmet_yolo/labels/train_aug"

os.makedirs(OUT_IMG_DIR, exist_ok=True)
os.makedirs(OUT_LBL_DIR, exist_ok=True)

# --- Augmentation pipeline using Affine ---
transform = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.VerticalFlip(p=0.3),
    A.RandomBrightnessContrast(p=0.5),
    A.Affine(translate_percent=0.05, scale=(0.9,1.1), rotate=(-15,15), p=0.7)
], bbox_params=A.BboxParams(format='yolo', label_fields=['class_labels']))

# --- Minority classes to augment ---
classes_to_augment = [1, 2]  # 1: Head, 2: Person

# --- Loop through images ---
for lbl_file in os.listdir(LBL_DIR):
    lbl_path = os.path.join(LBL_DIR, lbl_file)
    img_file = lbl_file.replace(".txt", ".png")  # Adjust if your images are .jpg
    img_path = os.path.join(IMG_DIR, img_file)

    if not os.path.exists(img_path):
        continue

    # Read image and labels
    img = cv2.imread(img_path)
    with open(lbl_path) as f:
        bboxes = []
        class_labels = []
        for line in f:
            cls, x, y, w, h = map(float, line.strip().split())
            if int(cls) in classes_to_augment:
                bboxes.append([x, y, w, h])
                class_labels.append(int(cls))

    # Apply augmentation only if the image contains minority classes
    if len(bboxes) > 0:
        for i in range(3):  # Number of augmented copies per image
            augmented = transform(image=img, bboxes=bboxes, class_labels=class_labels)
            aug_img = augmented['image']
            aug_bboxes = augmented['bboxes']
            aug_labels = augmented['class_labels']

            # Save augmented image
            out_img_name = img_file.replace(".png", f"_aug{i}.png")
            cv2.imwrite(os.path.join(OUT_IMG_DIR, out_img_name), aug_img)

            # Save augmented labels
            out_lbl_name = lbl_file.replace(".txt", f"_aug{i}.txt")
            with open(os.path.join(OUT_LBL_DIR, out_lbl_name), "w") as f_out:
                for cls, bbox in zip(aug_labels, aug_bboxes):
                    f_out.write(f"{cls} {' '.join(map(str, bbox))}\n")

print("✅ Dataset augmented successfully using Affine transform!")
```

```python
from ultralytics import YOLO

model = YOLO("/content/runs/detect/train/weights/best.pt")

model.predict(
    source="/content/helmet_yolo/images/val",
    conf=0.25,
    device=0,    # GPU
    save=True
)
```

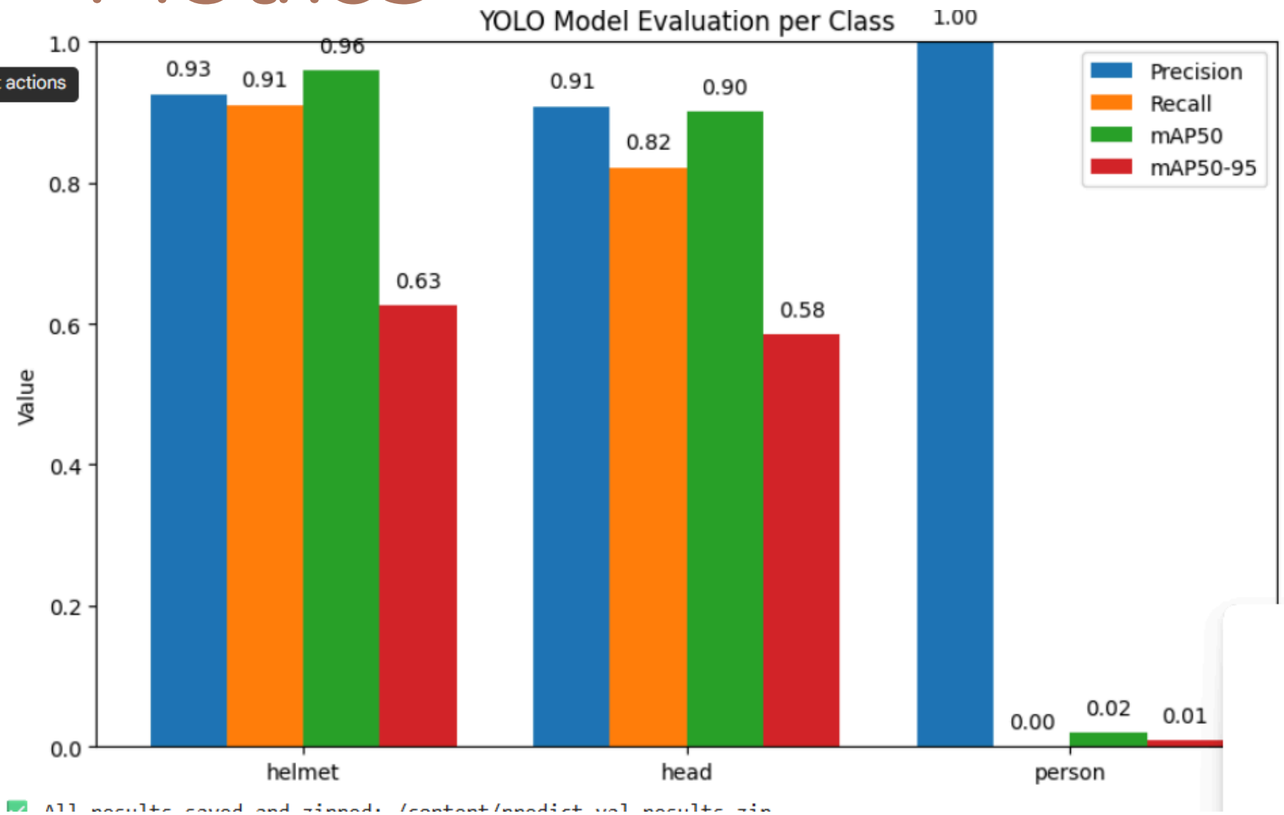
Evaluation Metrics

YOLO Model Evaluation (Overall):

	Metric	Value
0	Precision	0.9448
1	Recall	0.5771
2	mAP50	0.6261
3	mAP50-95	0.4067



output actions



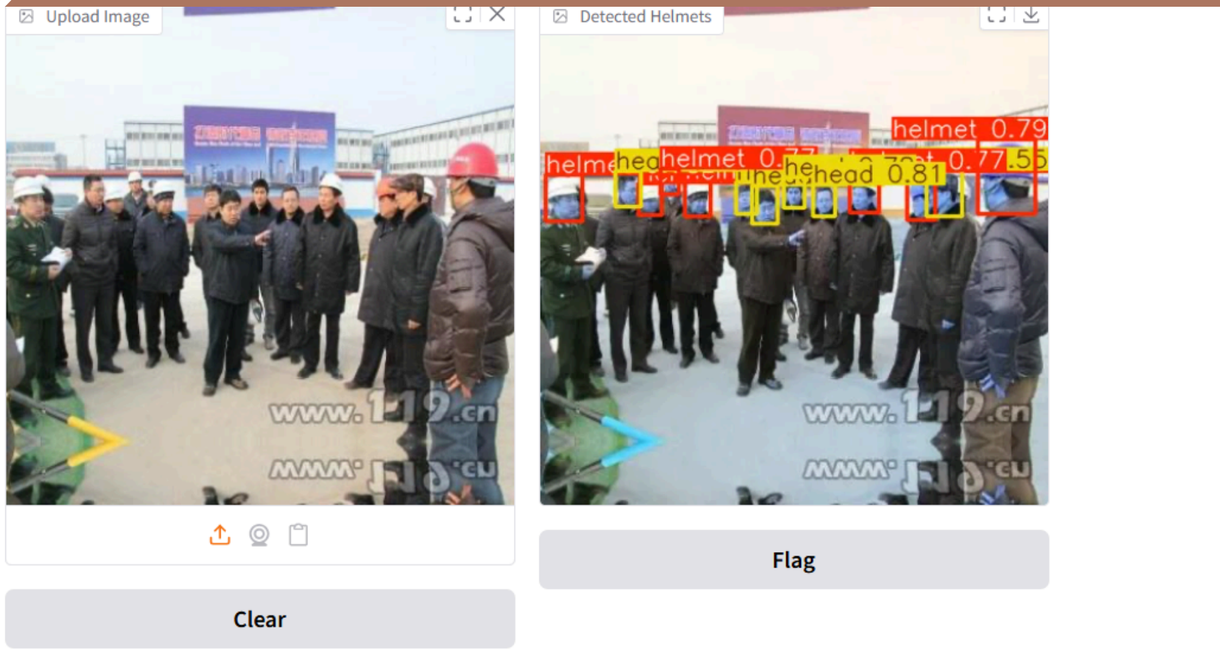

```
Ultralytics 8.3.241 🚀 Python-3.12.12 torch-2.9.0+cpu CPU (AMD EPYC 7B12)
Model summary (fused): 72 layers, 3,006,233 parameters, 0 gradients, 8.1 GFLOPs
val: Fast image access ✅ (ping: 0.0±0.0 ms, read: 35.2±14.1 MB/s, size: 266.5 KB)
val: Scanning /content/helmet_yolo/labels/val.cache... 1000 images, 0 backgrounds, 0 corrupt: 100% ━━━━━━━━━━━━ 1000/1000 241.
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ━━━━━━━━━━━━ 63/63 3.5s/it 3:38
                   all       1000       5169      0.961        0.6       0.65      0.438
                helmet        912       3916      0.948      0.922       0.97      0.654
                  head        185       1099      0.936      0.877       0.94      0.638
                person         36        154          1          0     0.0389     0.0224
Speed: 4.3ms preprocess, 200.0ms inference, 0.0ms loss, 0.5ms postprocess per image
Results saved to /content/runs/detect/val7
```

	Class	Precision	Recall	mAP50	mAP50-95
0	helmet	0.948017	0.922104	0.969963	0.653776
1	head	0.935891	0.876696	0.939827	0.637954
2	person	1.000000	0.000000	0.038919	0.022407

GUI (User Interface)

- Built using Gradio
- Features:
  - Upload image
  - Adjust confidence & IOU
  - View predicted image
  - View evaluation table & chart
  - Download results as ZIP

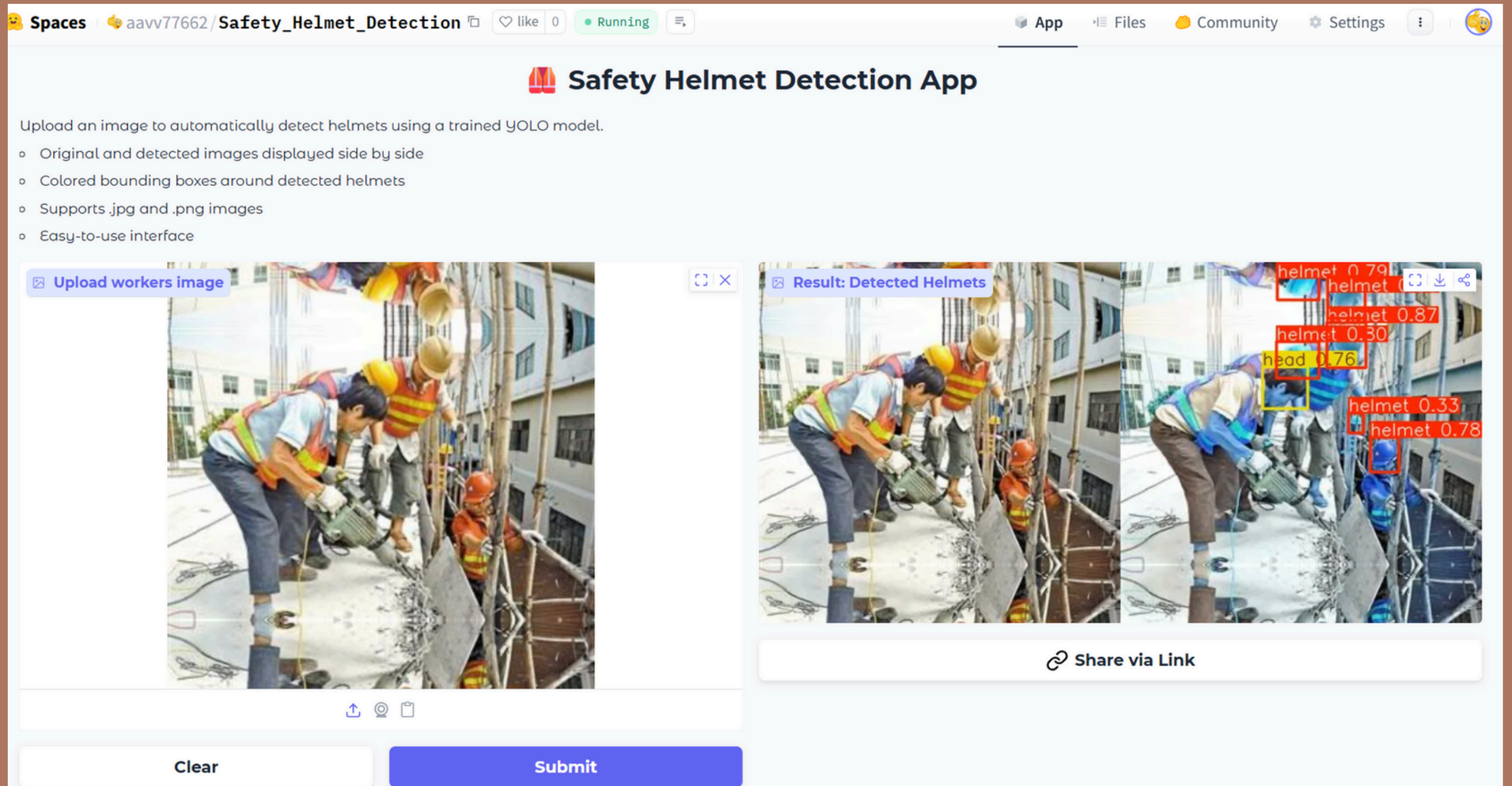
```
!pip install gradio ultralytics opencv-python

from ultralytics import YOLO
import gradio as gr

# Load your trained YOLO model
model = YOLO("/content/best (3).pt")

# Function to detect helmets on an image
def detect_helmet(image):
    results = model.predict(image, save=True)  # saves image with bounding boxes
    return results[0].plot()  # returns the image with detections

# Create Gradio interface
iface = gr.Interface(
    fn=detect_helmet,
    inputs=gr.Image(type="filepath", label="Upload Image"),
    outputs=gr.Image(label="Detected Helmets"),
    live=True
)

iface.launch()
```



Deployment

- Model uploaded to Hugging Face
- Easy access & sharing
- Ready for:
  - Web apps
  - APIs
  - Cloud deployment

https://huggingface.co/spaces/aavv77662/Safety_Helmet_Detection



Insight

This project demonstrates that lightweight deep learning models can effectively improve workplace safety when combined with proper data and system design.

Limitations & Future Work

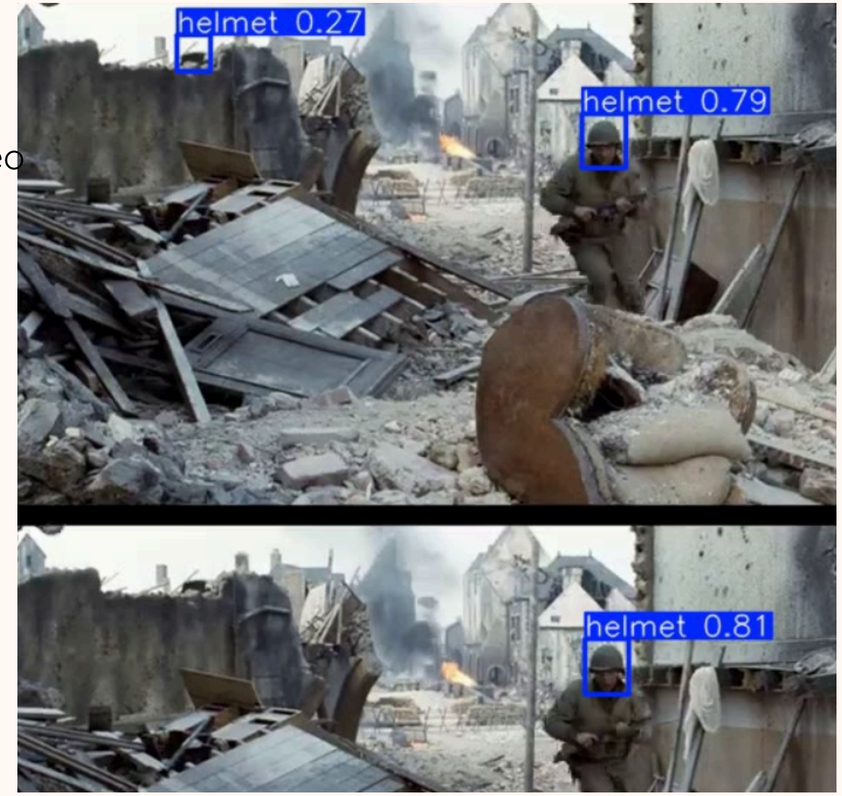
Limitations

- Dataset imbalance
- Occlusion

Future

- More data
- YOLOv8m
- Video + alerts

Output after test video to detection



Thank You!

The project demonstrates an effective real-time helmet detection system using YOLOv8.