**Assignment 2**

**By Abhishek Sharma**

**Task 1**

**#importing important libraries**

**from google.colab import files**

**uploaded = files.upload() # Choose the file manually**

**import numpy as np**

**import cv2**

**import matplotlib.pylab as plt**

img=cv2.imread('computer.jpg')

img2=cv2.imread('computer.jpg')

img=cv2.cvtColor(img,cv2.COLOR\_BGR2RGB)

plt.imshow(img)

plt.axis('off')

plt.title("original image")

plt.show()



def show\_hsv\_channels(image):

hsv = cv2.cvtColor(image, cv2.COLOR\_RGB2HSV)

h, s, v = cv2.split(hsv)

titles = ['Hue', 'Saturation', 'Value']

images = [h, s, v]

plt.figure(figsize=(10, 3))

for i in range(3):

plt.subplot(1, 3, i + 1)

plt.imshow(images[i], cmap='gray')

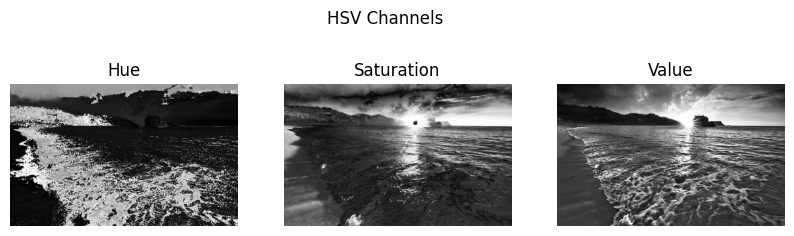
plt.title(titles[i])

plt.axis('off')

plt.suptitle("HSV Channels")

plt.show()

show\_hsv\_channels(img)



def show\_histogram\_equalization(image):

gray = cv2.cvtColor(image, cv2.COLOR\_RGB2GRAY)

equalized = cv2.equalizeHist(gray)

plt.figure(figsize=(8, 4))

plt.subplot(1, 2, 1)

plt.imshow(gray, cmap='gray')

plt.title('Original Grayscale')

plt.axis('off')

plt.subplot(1, 2, 2)

plt.imshow(equalized, cmap='gray')

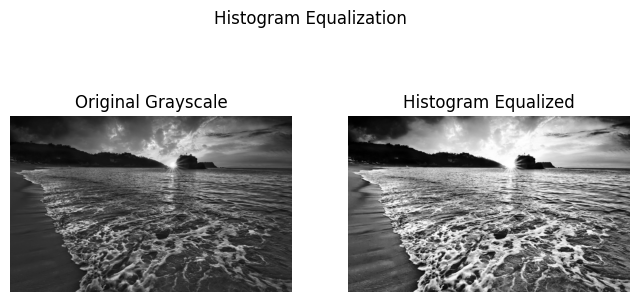
plt.title('Histogram Equalized')

plt.axis('off')

plt.suptitle("Histogram Equalization")

plt.show()

show\_histogram\_equalization(img)



def show\_binary\_inversion(image, threshold=128):

gray = cv2.cvtColor(image, cv2.COLOR\_RGB2GRAY)

\_, binary = cv2.threshold(gray, threshold, 255, cv2.THRESH\_BINARY\_INV)

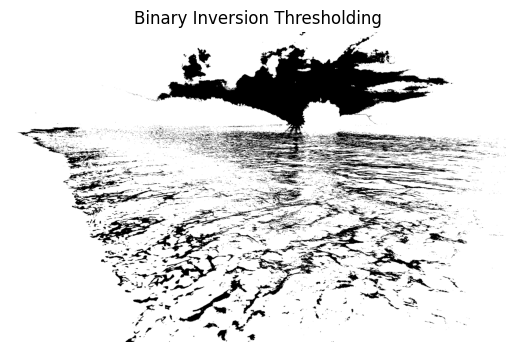
plt.imshow(binary, cmap='gray')

plt.title("Binary Inversion Thresholding")

plt.axis('off')

plt.show()

show\_binary\_inversion(img)



def show\_posterized\_image(image):

gray = cv2.cvtColor(image, cv2.COLOR\_RGB2GRAY)

posterized = (gray // 64) \* 64

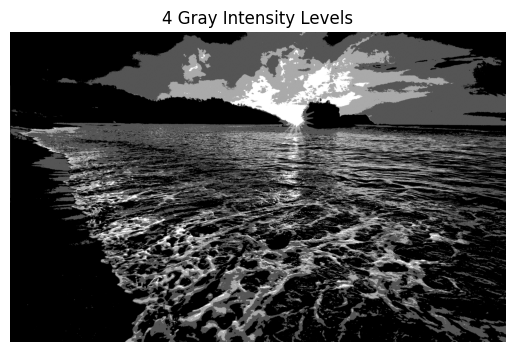
plt.imshow(posterized, cmap='gray')

plt.title("4 Gray Intensity Levels")

plt.axis('off')

plt.show()

show\_posterized\_image(img)



def show\_laplacian\_scharr(image):

gray = cv2.cvtColor(image, cv2.COLOR\_RGB2GRAY)

lap = cv2.Laplacian(gray, cv2.CV\_64F)

scharr = cv2.Scharr(gray, cv2.CV\_64F, 1, 0) + cv2.Scharr(gray, cv2.CV\_64F, 0, 1)

plt.figure(figsize=(8, 4))

plt.subplot(1, 2, 1)

plt.imshow(np.uint8(np.absolute(lap)), cmap='gray')

plt.title("Laplacian")

plt.axis('off')

plt.subplot(1, 2, 2)

plt.imshow(np.uint8(np.absolute(scharr)), cmap='gray')

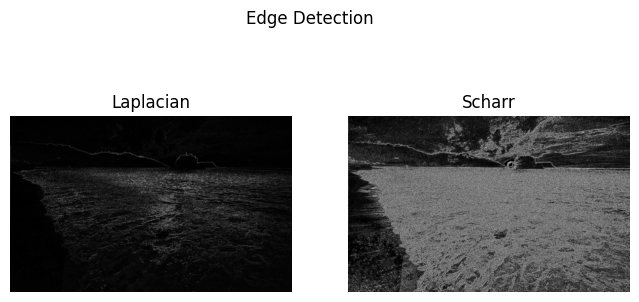
plt.title("Scharr")

plt.axis('off')

plt.suptitle("Edge Detection")

plt.show()

show\_laplacian\_scharr(img2)



**Task 2**

Set up YOLOv5 environment

Apply the following steps:

Run the following commands in terminal

Step1- git clone https://github.com/ultralytics/yolov5

Step2- cd yolov5

Step3- python -m venv yolov5env

Step4- yolov5env\Scripts\activate

step5 -pip install -r requirements.txt

Step6- Upload the image to train the model on the that

python detect.py --source data/images.jpg --weights yolov5s.pt

Here are some original images and images after detection

Original Image



Image after applying model



Original image 2



Image after applying model

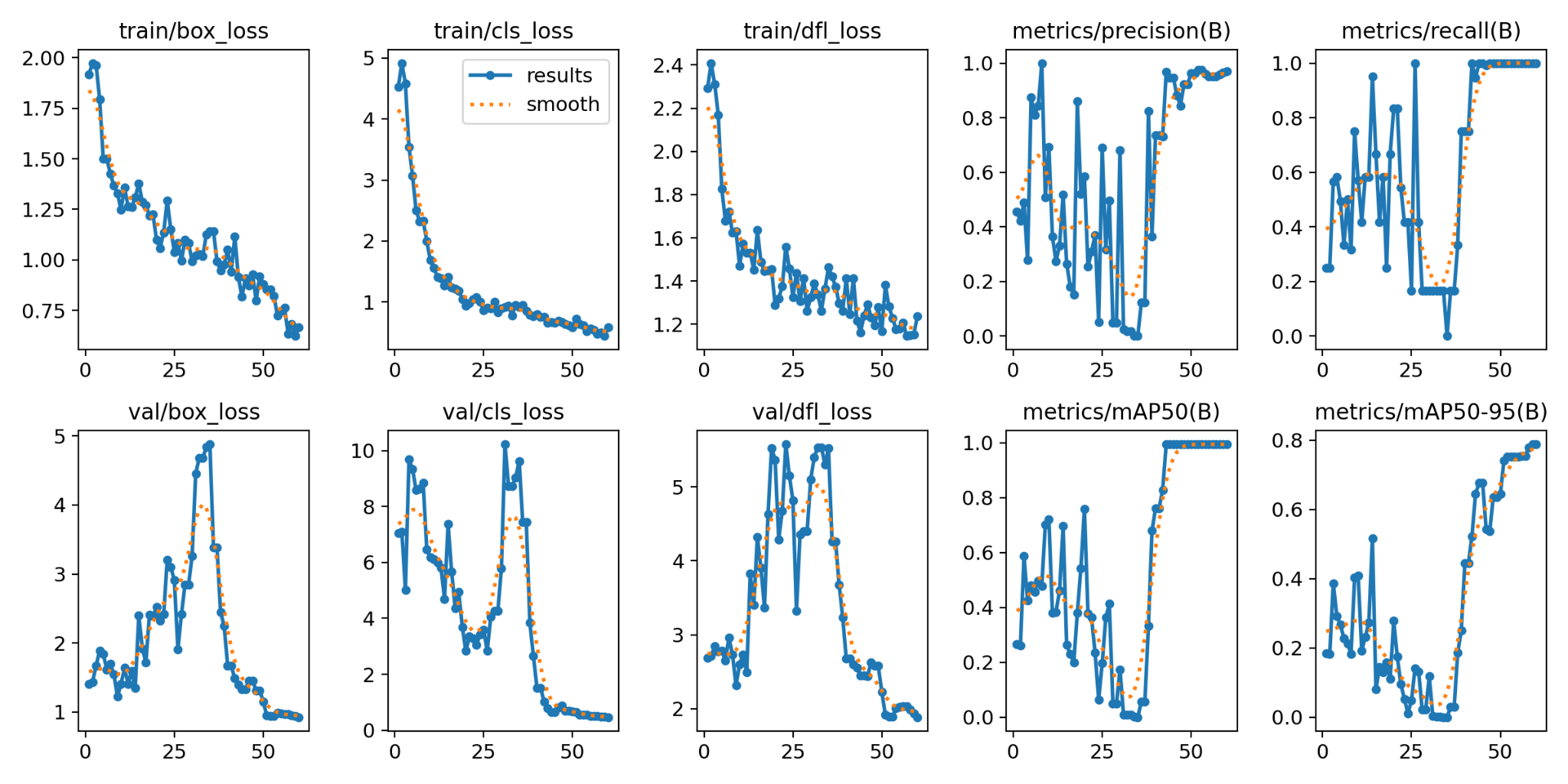


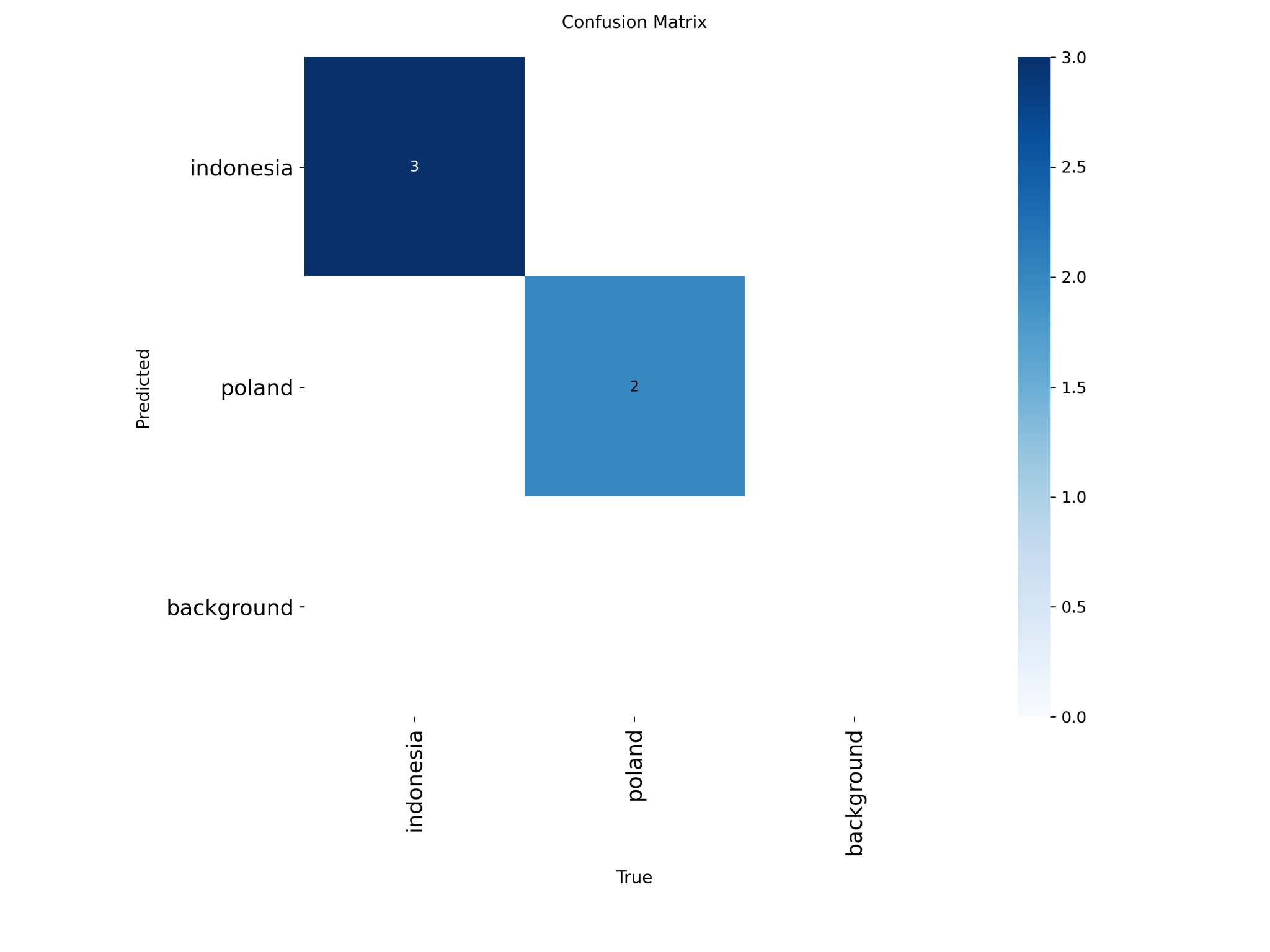
Task 3

Creating a yolo model to detect the poland and indonesia national flag

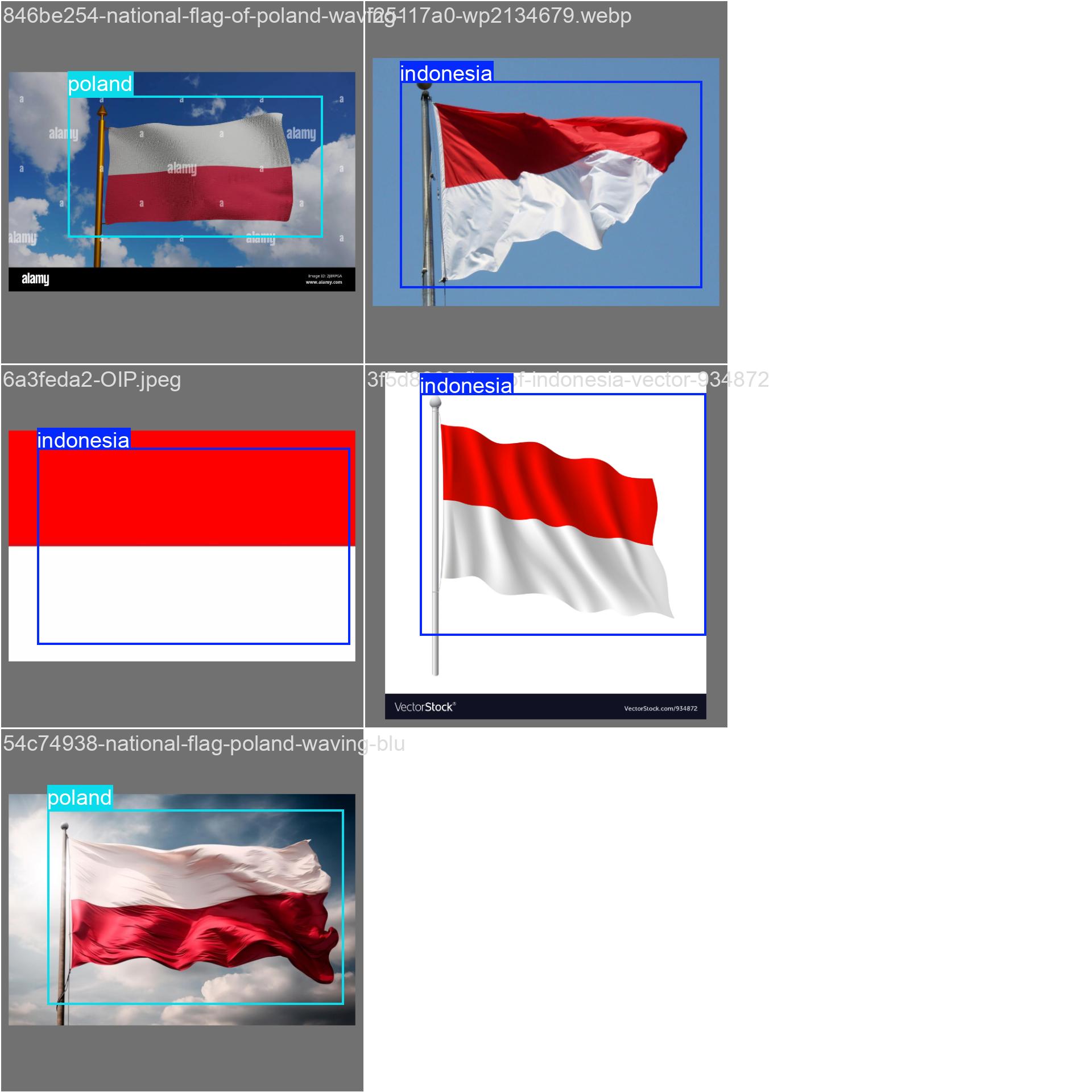
Follow the instruction given in the video linked in the assignment to create the model using yolo

* Results

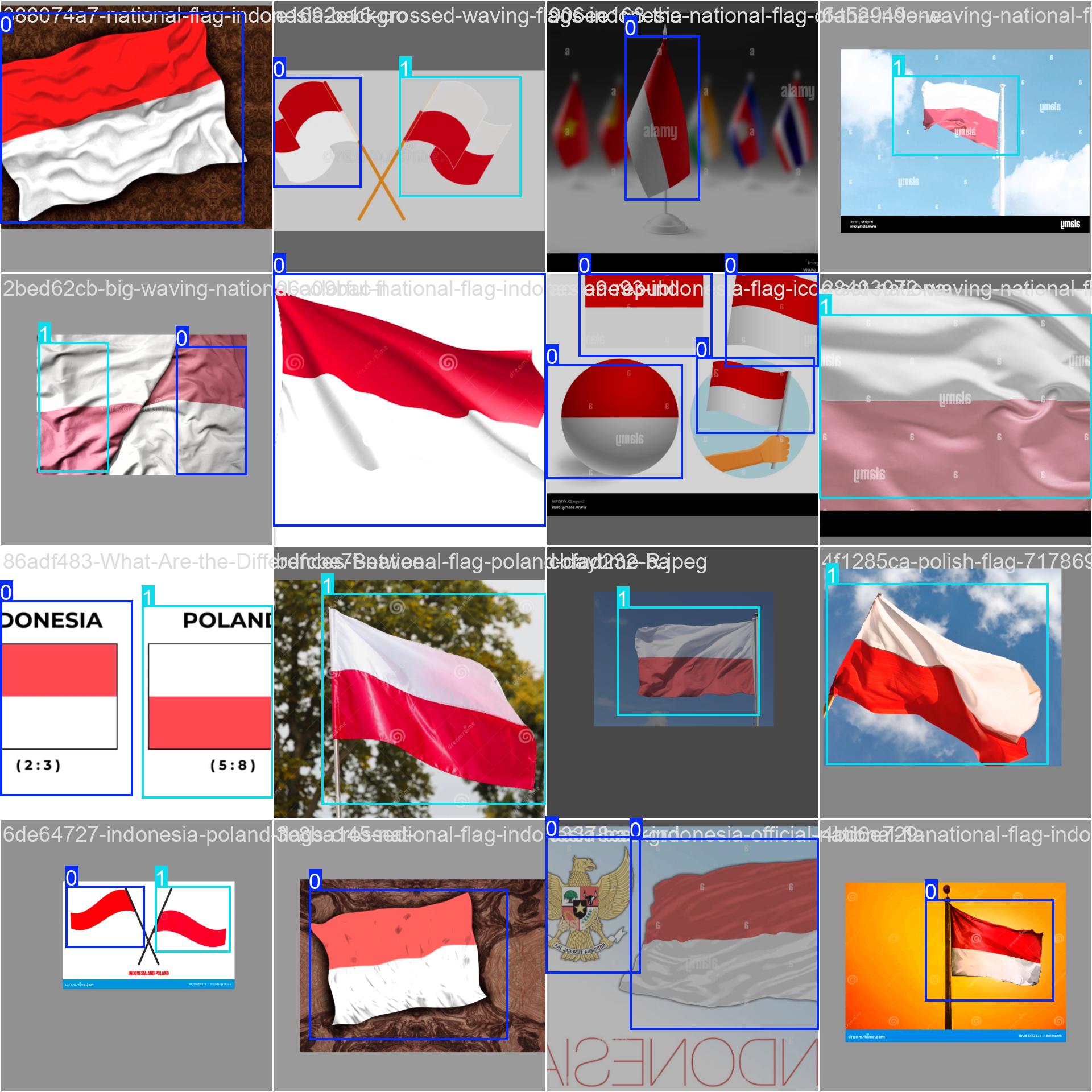




**Some training datasets**



**Some testing datasets**

****

**Task 4**

**Summary of YOLOv5 research paper**

This research paper presents a detailed analysis of **YOLOv5**, a state-of-the-art object detection model developed by Ultralytics. YOLOv5 stands out for its **speed, accuracy, and efficiency**, making it a preferred choice for real-time and resource-constrained applications like edge computing and IoT.

### **1. Objective of the Study**

The main goal of the study is to:

* Examine **YOLOv5’s internal architecture**, training methods, and evolution.
* Compare its performance to other object detection models.
* Analyze the **trade-offs between model size, accuracy, and speed** among its variants (n, s, m, l, x).
* Highlight YOLOv5's transition from the Darknet framework to **PyTorch**, and how it has improved usability, flexibility, and development.

### **2. Architecture Overview**

YOLOv5 follows the traditional YOLO architecture with three main components:

* **Backbone**: Extracts features from the image using CSP (Cross Stage Partial) networks.
* **Neck**: Uses PANet for feature fusion at different scales.
* **Head**: Generates final predictions for object bounding boxes and classes.

This design allows the model to detect small, medium, and large objects effectively in real-time.

### **3. Training Techniques**

Two key training methodologies are emphasized:

* **Data Augmentation**: Including **mosaic augmentation**, which stitches four images into one to improve detection of small or rare objects.
* **Loss Function**: A combination of classification, objectness, and localization losses using **Binary Cross-Entropy** and **Complete IoU (CIoU)**.

These techniques enhance generalization, especially in real-world conditions.

### **4. PyTorch Transition**

Shifting from the **Darknet (C-based)** to **PyTorch** allowed:

* Easier integration of new features.
* More user-friendly training and deployment.
* Support for modern development tools and hardware acceleration.

### **5. Supporting Tools and Formats**

YOLOv5 supports:

* A simple **TXT annotation format** with YAML config files.
* Popular labeling tools like **Roboflow, LabelImg, CVAT**, with easy format conversion.
* Integration with platforms like **Weights & Biases, ClearML**, and **Deci** for tracking, deployment, and optimization.

### **6. Innovations and Impact**

The paper highlights the following YOLOv5 innovations:

* **CSP Backbone**: Reduces redundancy and improves training efficiency.
* **PA-Net Neck**: Better feature aggregation.
* **16-bit precision**: Faster inference on supported GPUs.
* **Flexible model variants**: Support various use cases.

These make YOLOv5 one of the most efficient and adaptable models for real-time object detection.

### **Conclusion**

YOLOv5 is not just an improvement in object detection but also a significant **step forward in model accessibility and usability**. Its PyTorch implementation, modular design, and scalable architecture make it suitable for a wide range of real-world applications—from **smart manufacturing** to **renewable energy** monitoring.

By combining speed, accuracy, and ease of deployment, YOLOv5 sets a **high benchmark** for modern object detection systems.