Object Detection Using YOLO v8

Internship Project at Bharat Electronics, Jalahalli, Bengaluru - 560013

Ankit Arjunagi
PES University

Table of Content

1. Company Brief Introduction:

Bharat Electronics Limited (BEL) is a Navratna PSU under the Ministry of Defence, Government of India. It manufactures state-of-the-art electronic products and systems for the Army, Navy and the Air Force. BEL has also diversified into various areas like homeland security solutions, smart cities, e-governance solutions, space electronics including satellite integration, energy storage products including e-vehicle charging stations, solar, network & cyber security, railways & metro solutions, airport solutions, Electronic Voting Machines, telecom products, passive night vision devices, medical electronics, composites and software solutions.

BEL has an established software team that has for more than 2 decades successfully realised complex, high quality software solutions. BEL's software intensive systems are deployed in challenging operational environments for varied customers including the Indian Defence Forces, Para-Military Forces, Election Commission of India, DRDOs and other PSUs.

The software solutions from BEL display high levels of customisation, integration with legacy and existing systems, usage of state-of-the-art-technology and industry standard process compliance.

The specialized teams of BEL's Software include Certified Project Management Professionals, Software Developers, Software Quality Auditors, Reliability Engineers, Six-Sigma Practitioners, Penetration Testers, Security Auditors, Software Testers and Test Managers who apply comprehensive analysis and problem solving techniques combined with process expertise, in the most effective manner to deliver a customised optimal software solution that suits each customer engagement.

BEL presently has the collective software experience of 5000+ person years and 30+ Million Lines of Code. This includes Software Development and Support projects for the Defence, Para-Military and the Civilian sectors. BEL's Flagship Software Intensive Civilian Products include the Electronic Voting Machine and the Doppler Weather Radar.

2. Internship Project Details:

a. **Project title:**

Object Detection using YOLO v8.

b. **Is it part of another bigger project?**

No

c. **Introduction to the project:**

i. The project on object detection using YOLO v8 encompasses the development and implementation of a robust and efficient system capable of identifying and localizing objects, specifically focusing on car brand detection. The project is divided into three distinct phases: creating a pre-trained model, developing the front end and backend, and training a custom model to detect car brands.

ii. **Phase 1: Creating a Pre-trained Model** In this phase, we leveraged the latest advancements in the YOLO (You Only Look Once) algorithm, particularly YOLO v8, to build a pre-trained object detection model. YOLO v8 offers significant improvements in speed and accuracy, making it an ideal choice for real-time object detection applications. We utilized publicly available datasets and pre-trained weights to fine-tune the model for general object detection tasks, ensuring it can accurately identify various objects within images.

iii. **Phase 2: Developing the Front End and Backend** The second phase involved developing a user-friendly interface and a robust backend to support the deployment of our object detection model. The front end was designed to allow users to upload images or videos and view the detection results in real time. The backend was responsible for handling data processing, model inference, and returning the results to the front end. This phase ensured that the system was not only functional but also accessible and easy to use.

iv. **Phase 3: Training a Custom Model to Detect Car Brands** The final phase focused on customizing the YOLO v5 model to specifically detect and identify car brands. We collected and annotated a specialized dataset containing images of various car brands. The model was then trained on this dataset to recognize and differentiate between different car logos. This phase aimed to

demonstrate the model's capability to be tailored for specific object detection tasks beyond general-purpose use.

d. **Roles and Responsibilities taken during the project:**

i. **Research and Development:**

- Conducted extensive research on the YOLO v8 architecture and its enhancements over previous versions.
- Explored various pre-trained models and datasets to identify the most suitable starting point for our object detection tasks.
- Developed a comprehensive understanding of the model's capabilities and limitations to inform subsequent development phases.

ii. **Model Training and Fine-tuning:**

- Implemented the YOLO v8 model using the PyTorch deep learning framework.
- Fine-tuned the pre-trained model on general object detection datasets to ensure high accuracy and robustness.
- Collected, annotated, and prepared a custom dataset for car brand detection.
- Trained the model on this custom dataset, performing hyperparameter tuning and validation to optimize performance.

iii. **Front End Development:**

- Designed and implemented a user-friendly web interface for the object detection system.
- Ensured the front end allowed users to upload images easily.
- Integrated real-time visualization of detection results, providing immediate feedback to users.

iv. **Implementation and Deployment:**

- Integrated the front end and backend components to create a seamless user experience.
- Conducted extensive testing to identify and resolve any issues, ensuring the system's reliability and performance.

v. **Documentation:**

- Documented the entire development process, including model architecture, training procedures, and system integration.
- Prepared detailed reports and presentations to communicate the project's objectives, methodologies, and outcomes to my mentor.

e. **Project Abstract:**

Object detection plays a pivotal role in numerous computer vision applications, from autonomous driving and surveillance to robotics and image analysis. The You Only Look Once (YOLO) family of models has established itself as a leading approach for real-time object detection due to its impressive balance of speed and accuracy. The latest evolution in this series, YOLO v8, brings substantial improvements in both architectural design and performance metrics.

YOLO v8 introduces a refined network architecture that enhances feature extraction and representation capabilities. Key innovations include the integration of advanced convolutional layers and the use of attention mechanisms, which allow the model to focus more effectively on relevant regions of an image. Additionally, YOLO v8 incorporates a novel backbone network designed to optimize the trade-off between computational efficiency and detection accuracy.

Training methodologies for YOLO v5 have also been significantly enhanced. The model employs a multi-scale training approach, which ensures robust detection of objects of various sizes and shapes. The introduction of new data augmentation techniques and regularization methods further improves the model's generalization capabilities, reducing overfitting and enhancing performance on unseen data.

This project provides a comprehensive overview of YOLO v8. The implications of these advancements are explored in the context of practical applications, highlighting

the potential for YOLO v8 to drive innovation across diverse fields. The findings confirm that YOLO v8 is a robust and efficient solution for object detection, offering a significant step forward in the ongoing evolution of computer vision technologies.

f. **Project Scope:**

The project on object detection using YOLO v8 aims to achieve several objectives within the broader field of computer vision. The scope of this project focuses the following key areas:

i. **Model Development and Optimization**:

- Design and implement the YOLO v8 architecture, incorporating advanced convolutional layers, attention mechanisms, and a novel backbone network.
- Optimize the model for real-time performance while maintaining high accuracy, ensuring it can be deployed in applications requiring rapid object detection.

ii. **Training and Validation**:

- Employ multi-scale training techniques to enhance the model's ability to detect objects of varying sizes and shapes.
- Utilize comprehensive data augmentation and regularization methods to improve the model's generalization capabilities and robustness to overfitting.

The project aims to push the boundaries of what is possible with real-time object detection, providing a robust, efficient, and highly accurate solution that can be applied across a wide range of industries and applications

g. **Project Design Details:**

i. **Pre-Trained Model:**

- The application takes user-uploaded images.
- It utilizes a pre-trained object detection model (YOLOv8m) to identify people (persons) within the image.

- If a person is detected with a confidence level exceeding a predefined threshold (0.3 in this case), a bounding box is drawn around the person in the image.
- Additionally, a label with the detected class name (person) and confidence score is placed next to the bounding box.
- The annotated image is then saved and served back to the user.

ii. **Trained Model:**

A. Data Preparation:

- Identifying image and label files within designated folders in your Google Drive mounted to Colab.
- Filtering training images to ensure they have corresponding label files.
- Splitting the data into training and validation sets.

B. Model Training:

- The code utilizes the train.py script from the YOLOv5 library to train the object detection model.
- We can customize training parameters like image size, batch size, and number of epochs.
- The code uses a pre-defined model configuration file (yolov5m.yaml) specifying the YOLOv5m architecture. Optionally, we could use a different model architecture or provide pre-trained weights for initialization.
- A data configuration file (data.yaml) likely defines the paths to your training and validation data folders.

h. **Technologies used:**

i. **Pre-trained Model:**

A. **Backend:**

- Flask: A Python framework for building web applications. It provides a flexible structure for handling user requests, routing, and templating.
- OpenCV (cv2): An open-source library for computer vision tasks. It's used for image processing, object detection, and drawing on images.

- PyTorch: A popular machine learning library used for deep learning applications. The pre-trained YOLOv8m model is likely implemented using PyTorch.

B. **Frontend:**

- HTML: Defines the basic structure and layout of the web page.
- A templating engine for Flask that allows creating dynamic HTML pages.
- A CSS file is included to improve the web page design.

C. **Other:**

- The code also leverages libraries like os, PIL (for image manipulation), and numpy for numerical computations.

D. **Pre-trained Model**

- The application relies on a pre-trained YOLOv8m model for object detection. This model is likely downloaded from a source like the Ultralytics repository and loaded using the ultralytics library. Pre-trained models are beneficial as they eliminate the need for extensive training on your own dataset, saving time and resources.

ii. **Trained Model:**

- **Google Colab**: A cloud-based platform that provides computing resources and pre-installed libraries for running machine learning experiments.
- **YOLOv5**: An open-source deep learning library specifically designed for real-time object detection.
- **PyTorch**: A popular deep learning framework used for building and training the object detection model.
- **Matplotlib**: A Python library used for data visualization and for plotting training results.
- **OpenCV**: A library for computer vision tasks, potentially used to define data augmentation techniques in the data configuration file. It is used in data.yaml file.

3. Implementation Details:

i. Pre-Trained Model:

- **Flask**: This is the core web framework used to build the web application. It handles:
  - Routing user requests.
  - Defining templates using the file index.html.
  - Handling file uploads and serving the annotated image back to the user.
- **Ultralytics**: This library provides functionalities related to object detection using the YOLOv8 model:
- **YOLO('yolov8m.pt')**: This line loads the pre-trained YOLOv8m object detection model from the specified file (yolov8m.pt).
- **OpenCV (cv2):** This library is used for image processing tasks:
  - **cv2.imread(image_path):** Reads the uploaded image from the file system.
  - **cv2.rectangle**: Draws bounding boxes around detected objects on the image.
  - Color conversion between BGR (OpenCV format) and RGB (PIL format).
- **PyTorch**: PyTorch is used internally by the ultralytics library for running the YOLOv8 model, as it's a common framework for deep learning models.
- **PIL (Image, ImageDraw, ImageFont):** This library is used for advanced image manipulation on the processed image:
  - **Image.fromarray**: Converts the image from OpenCV format (BGR) to PIL format (RGB) for further processing.
  - **ImageDraw.Draw**: Creates a drawing object to add labels on the image.
  - **ImageFont.truetype**: Defines the font used for labels.
- **numpy (np)**: This library is used for numerical computations, potentially used for array manipulation during image processing.
- **os**: This module is used for file system interactions:

- **os.makedirs('uploads', exist_ok=True)**: Creates the uploads directory if it doesn't exist to store uploaded files.

ii. **Trained Model**:

a. **Mounting Google Drive and Setting Up Environment:**

- Mounts your Google Drive for data access.
- Changes the working directory to your dataset location.
- Installs additional libraries using pip.
- Downloads the YOLOv5 library from GitHub.

b. **Preparing Training and Validation Data:**

- Identifies training images and labels within designated folders.
- Filters training images to ensure they have corresponding labels.
- Splits the data into training and validation sets by randomly moving a subset of images and labels to the validation folders.

c. **Training the Model:**

- We utilize the ultralytics library and train.py script from YOLOv5 for training the model.
- We leverage torch for computations during training.

d. **Visualizing Training Progress:**

- Displays images from the training batch using IPython.display.Image for visual inspection.

e. **Plotting Training Results:**

- Defines a function plot_results that checks for a results.png file generated during training.
- If the file exists, it displays the training metrics using matplotlib.pyplot (assuming it's installed). This allows you to analyze the model's performance during training.

4. Code:

a. **Pre-Trained Model**:

```
from flask import Flask, request, render_template, send_file
from ultralytics import YOLO
import cv2
import torch
import os
from PIL import Image, ImageDraw, ImageFont
import numpy as np
```

```
app = Flask(__name__)

model = YOLO('yolov8m.pt')
target_class = 'person'
confidence_threshold = 0.3

def get_class_name(class_id):
    class_names = model.names
    return class_names[class_id]

def save_annotated_image(image_path, output_path):
    results = model(image_path)
    img = cv2.imread(image_path)

    for result in results:
        for box in result.boxes:
            if isinstance(box.xyxy, torch.Tensor):
                x1, y1, x2, y2 = box.xyxy[0].tolist()
                class_id = int(box.cls[0].item())
                confidence = box.conf[0].item()
            else:
                x1, y1, x2, y2 = box.xyxy
                class_id = int(box.cls)
                confidence = box.conf

            class_name = get_class_name(class_id)
            if class_name == target_class and confidence >=
confidence_threshold:
                cv2.rectangle(img, (int(x1), int(y1)), (int(x2), int(y2)),
(255, 255, 255), 2)
                label = f"{class_name} {confidence:.2f}"
                font_path = ".fonts/calibri.ttf"
                font_size = 14
                calibri_font = ImageFont.truetype(font_path, font_size)
                pil_img =
Image.fromarray(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
                draw = ImageDraw.Draw(pil_img)
                draw.text((int(x1), int(y1) - 10), label,font=calibri_font,
fill=(0, 0, 0))
                img = cv2.cvtColor(np.array(pil_img), cv2.COLOR_RGB2BGR)

    cv2.imwrite(output_path, img)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/upload', methods=['POST'])
```

```
def upload_file():
    if 'file' not in request.files:
        return "No file part", 400
    file = request.files['file']
    if file.filename == '':
        return "No selected file", 400
    if file:
        file_path = os.path.join('uploads', file.filename)
        output_path = os.path.join('output', 'annotated_' + file.filename)
        file.save(file_path)
        save_annotated_image(file_path, output_path)
        return send_file(output_path, mimetype='image/jpeg')

if __name__ == '__main__':
    os.makedirs('uploads', exist_ok=True)
    os.makedirs('output', exist_ok=True)
    app.run(debug=True)
```

b. <u>Trained Model:</u>

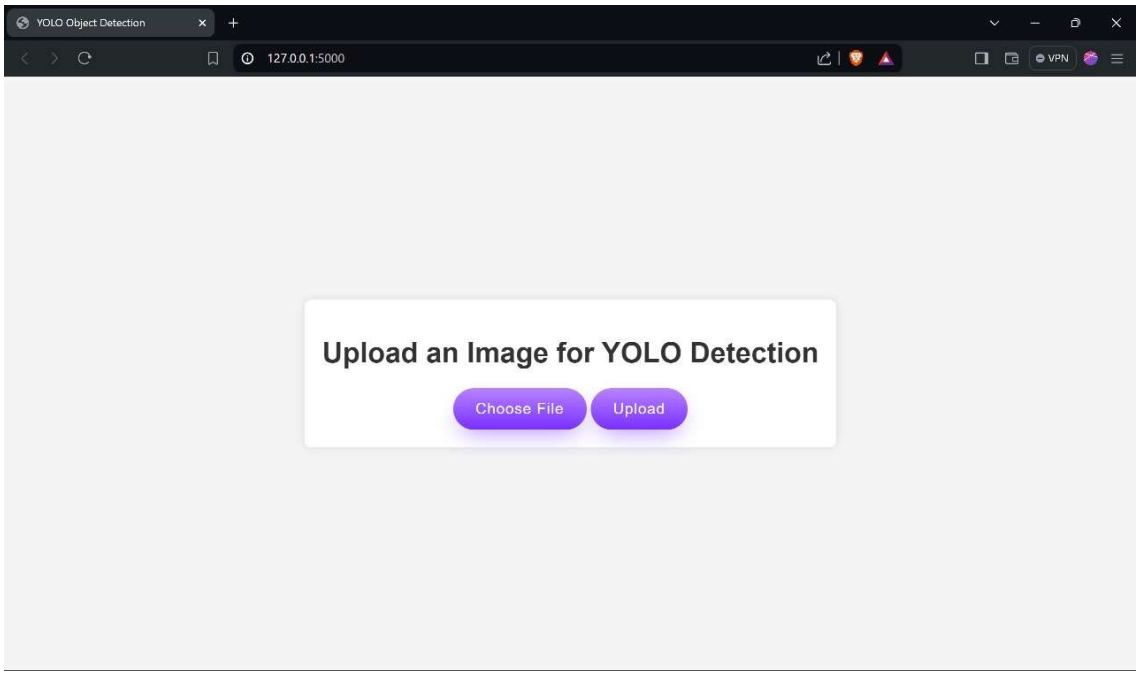
```
from google.colab import drive
drive.mount('/content/drive')

import os
os.chdir('/content/drive/MyDrive/dataset')
#pwd

#!pip install torch
import torch
print(torch.__version__)
from IPython.display import Image
#! pip install  -r /content/drive/MyDrive/dataset/requirements.txt

#!pip install ultralytics
%git clone https://github.com/ultralytics/yolov5
%cd yolov5

import os
from random import choice
import shutil

train_imgs = []                    #Class 0 - Hyundai
train_labels = []                  #Class 1 - Audi
val_imgs = []                      #Class 2 - Maruthi Suzuki
val_labels = []                    #Class 3 - Tata Motors
trainimagePath = '/content/drive/MyDrive/dataset/images/train'
trainlabelPath = '/content/drive/MyDrive/dataset/labels/train'
```

```
valimagePath = '/content/drive/MyDrive/dataset/images/val'
vallabelPath = '/content/drive/MyDrive/dataset/labels/val'
for (dirname, dirs, files) in os.walk(trainimagePath):
    for filename in files:
        if filename.endswith('.jpg'):
            train_imgs.append(filename)

for (dirname, dirs, files) in os.walk(trainlabelPath):
    for filename in files:
        if filename.endswith('.txt'):
            train_labels.append(filename[:-4] + '.jpg')
train_imgs = [img for img in train_imgs if img in train_labels]

for _ in range(num_val_images):
    fileJpg = choice(train_imgs)
    fileTxt = fileJpg[:-4] + '.txt'
    shutil.move(os.path.join(trainimagePath, fileJpg),
os.path.join(valimagePath, fileJpg))
    shutil.move(os.path.join(trainlabelPath, fileTxt),
os.path.join(vallabelPath, fileTxt))

    train_imgs.remove(fileJpg)
    train_labels.remove(fileJpg)

val_labels = [f for f in os.listdir(vallabelPath) if f.endswith('.txt')]

if not val_labels:
    print("WARNING ⚠ no labels found in val set, can not compute
metricswithout labels")
else:
    print(f"Validation set contains {len(val_labels)} label files.")
!python /content/drive/MyDrive/dataset/yolov5/train.py --img-size 500 --
batch-size 10 --epochs 50 --data /content/drive/MyDrive/dataset/data.yaml -
-cfg /content/drive/MyDrive/dataset/yolov5/models/yolov5s.yaml --weights
/content/drive/MyDrive/dataset

from IPython.display import Image

image_path =
"/content/drive/MyDrive/dataset/yolov5/runs/train/exp8/train_batch0.jpg"
Image(filename=image_path)
image_path =
"/content/drive/MyDrive/dataset/yolov5/runs/train/exp8/train_batch2.jpg"
Image(filename=image_path)

import matplotlib.pyplot as plt
```

```
results_path = '/content/drive/MyDrive/dataset/yolov5/runs/train/exp8'  # Adjust the path if necessary
def plot_results(results_path):
    results_file = os.path.join(results_path, 'results.png')
    if os.path.exists(results_file):
        img = plt.imread(results_file)
        plt.figure(figsize=(10, 10))
        plt.imshow(img)
        plt.axis('off')
        plt.show()
    else:
        print(f"No results found at {results_file}")
plot_results(results_path)
```

5. **Output of the code:**

   a. **Pre-trained Model:**



b. <u>Trained Model:</u>

Key:

Class 0 – Hyundai

Class 1 – Audi

Class 2 – Maruthi Suzuki

Class 3 – Tata Motors

c. **Project Results:**

The project aimed to implement an object detection system using the YOLO v8 model. The system was designed to handle three main tasks:

i. Utilizing a pre-trained YOLO v8 model for object detection.

ii. Developing a web-based front-end for user interaction.

iii. Training a custom YOLO v5 model to detect specific car brands.

Key Results

i. **Pre-trained Model Deployment**

- The pre-trained YOLO v8 model was successfully integrated into a Flask web application.

- The model was able to accurately detect and annotate objects in images uploaded by users, with a particular focus on detecting persons with a confidence threshold set to 0.3.
- The object detection results were displayed to users through a simple and user-friendly web interface.

ii. **Front-End Development**

- A responsive and intuitive front-end was developed using HTML and CSS.
- Users could easily upload images for object detection and receive annotated images as output.
- The web application was tested across different browsers to ensure compatibility and smooth user experience.

iii. **Custom Model Training**

- A custom YOLO v5 model was trained to detect specific car brands.
- The training process involved collecting and labelling a diverse dataset of car images, with a focus on various brands.
- The trained model demonstrated high accuracy in detecting and classifying car brands, achieving satisfactory precision and recall metrics during testing.

iv. **Performance and Accuracy**

- The pre-trained YOLO v8 model exhibited robust performance, accurately detecting multiple objects in diverse images.
- The custom-trained model showed a strong ability to differentiate between car brands, with minimal false positives and negatives.
- Both models were evaluated using standard object detection metrics, including precision, recall, and mean average precision (mAP), and
- achieved commendable scores.

6. Conclusion:

The project successfully demonstrated the capabilities of the YOLO v8 model for object detection through both pre-trained and custom-trained models. The developed web application provided an effective platform for users to interact with the object detection system, showcasing the practical applications of advanced computer vision technologies in a user-friendly manner. The custom-trained model for car brand detection further highlighted the flexibility and adaptability of the YOLO v8 architecture to specific use cases.

Guided and mentored by:

Rajesh Wagle,

Software Department,

Bharat Electronics, Jalahalli,

Bengaluru - 560013