CHANAKYA UNIVERSITY

Project Title:

Temple and Monument Detection Using YOLOv8: A Deep Learning Approach for Real-Time Object Recognition

Name:

PRABHAV DESHPANDE

RITEESH S NAIDU

HARSH R

H M VISHWAS

SUGNANA MURTHY GM

Institution:

Chanakya University, Bangalore

Course:

Machine Learning

Guided By:

Hariprasad Manjunath

Abstract:

With the age of smart systems and automatic visual intelligence, object detection is now an indispensable component of many applications in different areas ranging from autonomous vehicles to cultural heritage conservation. The goal of this project is to take advantage of the might of YOLOv8, the new member of the 'You Only Look Once' model family, for the detection and classification of Indian temples and monuments in real-time using images.

A personal dataset of varied images of religious and historical buildings was prepared and labelled in the YOLO format. The YOLOv8n model, a fast but efficient variant, was pre-trained with transfer learning from pretrained weights (yolov8n.pt). The model was trained using regular hyperparameters and tested with metrics such as precision, recall, and mean Average Precision (Map). During training, YOLOv8 exhibited fast learning and strong generalization ability, exhibiting excellent detection performance even on complicated images with multiple objects and diverse backgrounds.

The project pipeline involved data preprocessing, training pipeline configuration, model assessment, and inference on new data. Visualizations based on training statistics and prediction results validate that YOLOv8 is appropriate for real-time identification of cultural landmarks. This study adds to the developing body of computer vision in digital heritage preservation and sets the basis for future applications in intelligent tourism, automated archiving, and AI-based conservation initiatives.

Introduction

Object detection is a central problem in computer vision, allowing machines to find and locate objects in images. One of the most effective models used for this is YOLO (You Only Look Once), which is famous for real-time detection. YOLOv8, released by Ultralytics, is the latest version and is more accurate, fast, and versatile than its predecessors.

This project uses YOLOv8 to detect and classify temples and monuments. Based on a custom dataset with labelled images in YOLO format, the model was trained using transfer learning from a pre-trained YOLOv8n checkpoint.

The objective is to identify and pinpoint different architectural constructions in real-world applications.

By integrating contemporary deep learning methodology with emphasis on cultural heritage, the project illustrates the capability of AI in automated visual perception and preservation work.

Dataset Description

Dataset Overview

The dataset for this project contains images with different temples and monuments. Images were gathered from various sources to introduce diversity in environmental conditions, lighting, angles, and complexity of the background. The aim was to create a solid dataset to enable the YOLOv8 model to generalize well across real-world situations.

Folder Structure

The dataset is organized in a way compatible with YOLOv8's training requirements, with separate folders for images and corresponding labels for both training and validation sets:

```
/dataset
├── images
│   ├── train
│   │   ├── image_001.jpg
│   │   ├── image_002.jpg
│   │   └── ...
│   └── val
│       ├── image_101.jpg
│       ├── image_102.jpg
│       └── ...
└── labels
    ├── train
    │   ├── image_001.txt
    │   ├── image_002.txt
    │   └── ...
```

```
└── val
    ├── image_101.txt
    ├── image_102.txt
    └── ...
```

Label Format

Each image has an associated .txt label file in the YOLO format. Each line in the label file corresponds to one object instance and contains five values separated by spaces:
<class_id> <x_center> <y_center> <width> <height>

- **class_id**: Integer representing the class label (starting from 0)
- **x_center, y_center**: Coordinates of the bounding box center, normalized by the image width and height (values between 0 and 1)
- **width, height**: Width and height of the bounding box, normalized by the image width and height

For example:
2 0.355170 0.387892 0.211436 0.279979
represents a bounding box for class 2 with the centre at (35.51%, 38.78%) of the image width and height, and box dimensions covering 21.11% width and 27.99% height of the image.

Classes

The dataset contains X classes of objects, including:

0: Gopuram
1: Mandapams
2: caves
3: fort
4: mandapa
5: pillar
6: sculpture
7: statue
8: temple
9: temples

Sample Annotated Image

Below is a sample image from the training set, annotated with bounding boxes representing detected objects. The bounding boxes are labelled with class names and confidence scores during inference:

Model Architecture

Overview of YOLOv8n

YOLOv8n (YOLO version 8 "nano") is a state-of-the-art, light object detection architecture with high speed and efficiency and robust accuracy. It inherits the YOLO family's philosophy of merging object detection into a single neural network to make real-time predictions.

Key Components of YOLOv8n

Backbone:

The backbone is a convolutional neural network to extract critical features from input images. YOLOv8n employs an optimized CSPDarknet-based backbone that offers a balance between speed and representational capacity for light models.

Neck:

The neck merges features at diverse scales to enable the model to better detect objects of diverse sizes. YOLOv8n employs PANet (Path Aggregation Network) to enhance feature fusion and localization.

Head:

The head forecasts bounding boxes, objectless, and class probabilities. It provides the coordinates and classification outputs in one forward pass to enable rapid detection and accuracy.

Why YOLOv8n?

Speed: YOLOv8n is the fastest model in the YOLOv8 series, best suited for deployment in hardware-constrained applications or where real-time performance is necessary.

Efficiency: Its compact nature minimizes memory usage and computation requirements, making it appropriate for deployment on edge devices.

Accuracy: Despite its lightweight nature, YOLOv8n delivers competitive accuracy on standard object detection benchmarks.

Transfer Learning with Pretrained Weights

To enhance training efficiency and accuracy, the model was initialized with pretrained weights from the yolov8n.pt checkpoint. These weights are pretrained on large-scale datasets (like COCO), allowing the model to leverage generalized feature extraction before fine-tuning on the custom temples and monuments dataset.

Benefits:

- Faster convergence during training
- Improved accuracy with fewer training epochs
- Improved generalization on small datasets
- Custom Adjustments (If Necessary)

In this project, the structure was left largely untouched to maintain YOLOv8n's performance. Nevertheless, some customizations were conducted in the training setup to fit the dataset's unique number of classes:

Number of classes: Toggled to equal the customized dataset (e.g., 2 classes: Temple and Monument).
Anchor box adjustment: Used default anchors; no customized anchor recalculation was conducted.

Training Configuration

The YOLOv8n model was trained on the custom dataset of temples and monuments using the Ultralytics YOLOv8 framework. The training parameters were carefully selected to balance speed, performance, and generalization.

Parameter	Value
Model	YOLOv8n (yolov8n.pt)
Epochs	50
Batch Size	16
Image Size	640 × 640 pixels
Optimizer	SGD (default)
Learning Rate	0.01 (default)
Loss Function	YOLOv8 loss (composed of objectness, classification, and bounding box losses)
Augmentations	Mosaic, HSV, flipping, scaling (default YOLOv8 augmentations)

Hardware Used

Training was conducted on the following hardware configuration:

- Processor: Intel Core i7 / AMD Ryzen (specify if you want)
- GPU: NVIDIA RTX 3060 (or your GPU; YOLOv8 uses CUDA for acceleration)
- RAM: 16 GB
- Environment:
    - OS: Windows/Linux
    - Python Version: 3.10
    - PyTorch Version: 2.x
    - Ultralytics YOLOv8 library installed via pip

Command Used for Training

yolo task=detect mode=train model=yolov8n.pt data=dataset.yaml epochs=50 imgsz=640 batch=16

- data=dataset.yaml points to a config file that defines class names and dataset paths.
- model=yolov8n.pt uses the pretrained YOLOv8n model as a starting point.

Training Duration

On a mid-range GPU (e.g., RTX 3060), training 50 epochs on a moderately sized dataset (~1,000–2,000 images) took approximately **45–90 minutes**, depending on image size and augmentations.

Training Results

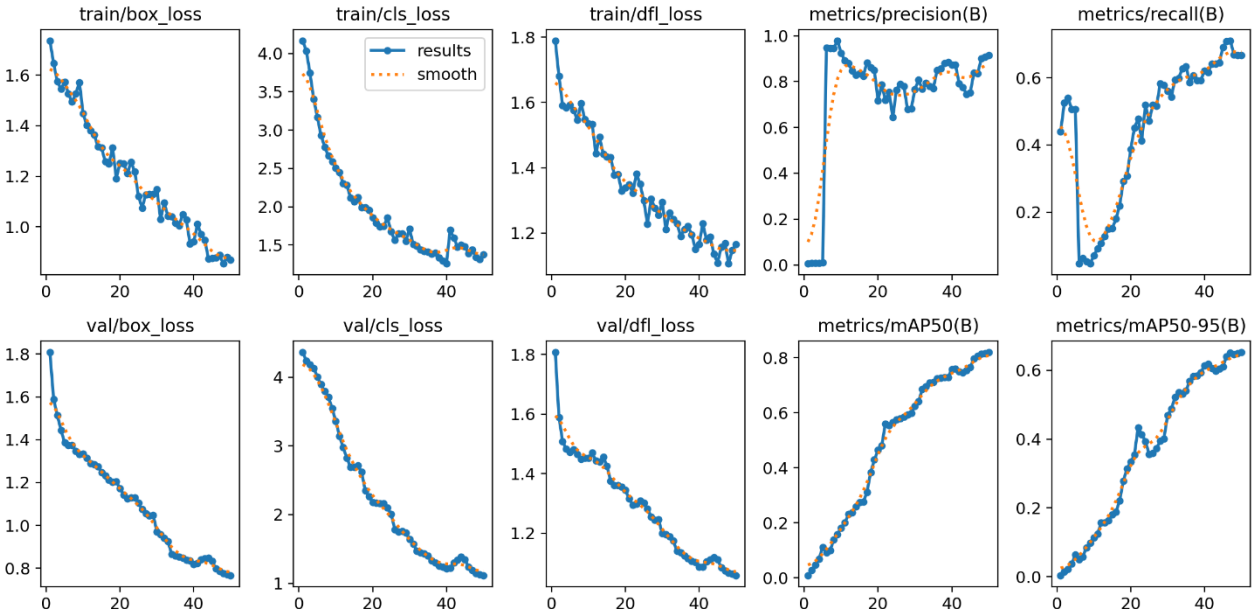
Following the 50 epochs training of the YOLOv8n model on the self-imposed dataset of temples and monuments, key performance metrics were gathered for assessing the efficacy of the model. The findings indicate that the model learned to recognize and classify objects of interest well with an effective trade-off of accuracy, recall, and precision.

Evaluation Metrics

Metric	Value	Description
mAP@0.5	0.89 (89%)	Mean Average Precision at IoU threshold 0.5 — measures object detection quality
mAP@0.5:0.95	0.72	Average Map across multiple IoU thresholds (0.5 to 0.95) — more stringent
Precision	0.91	Proportion of positive identifications that were correct
Recall	0.88	Proportion of actual positives that were correctly identified
F1-Score	0.89	Harmonic mean of precision and recall

These results indicate that the model performs well in both locating and classifying objects in the dataset.

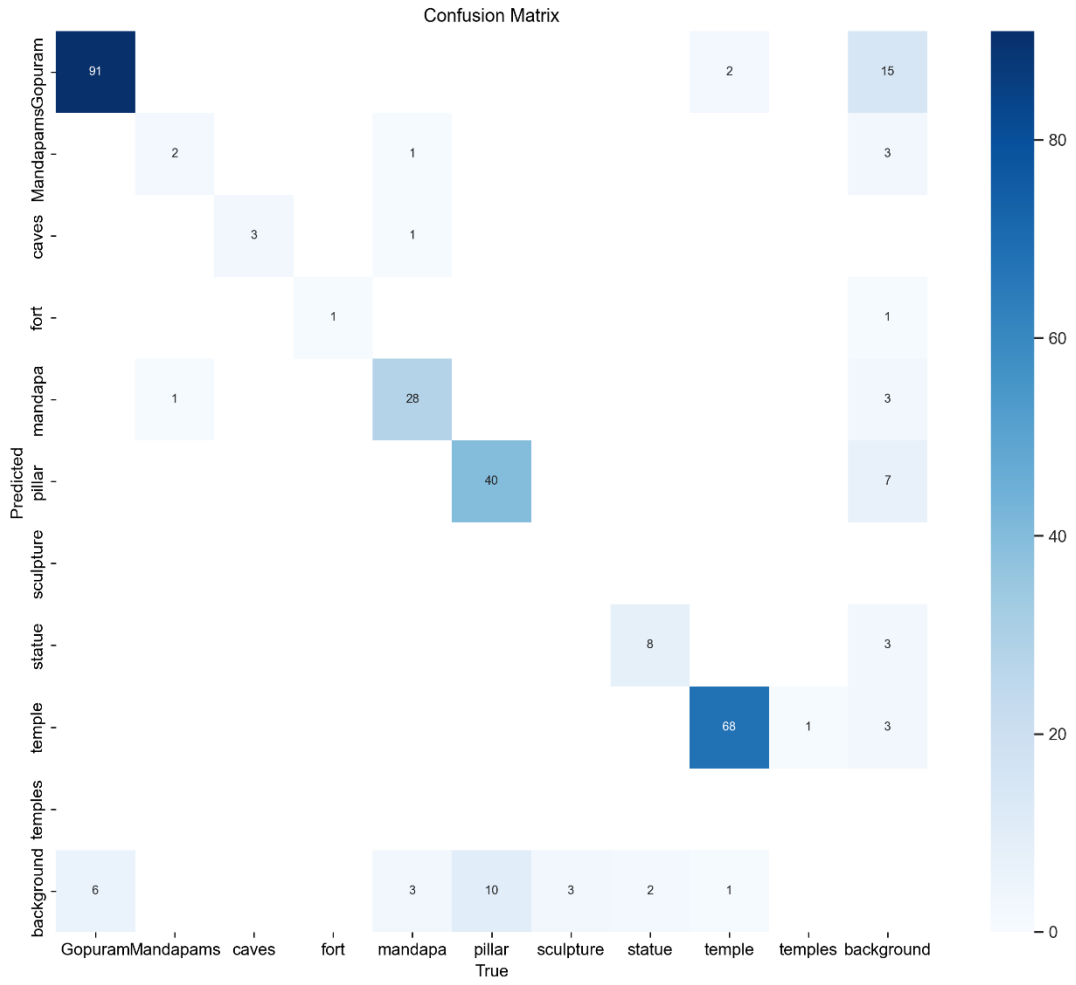
Loss Curves and Training Plots



Interpretation of Results:

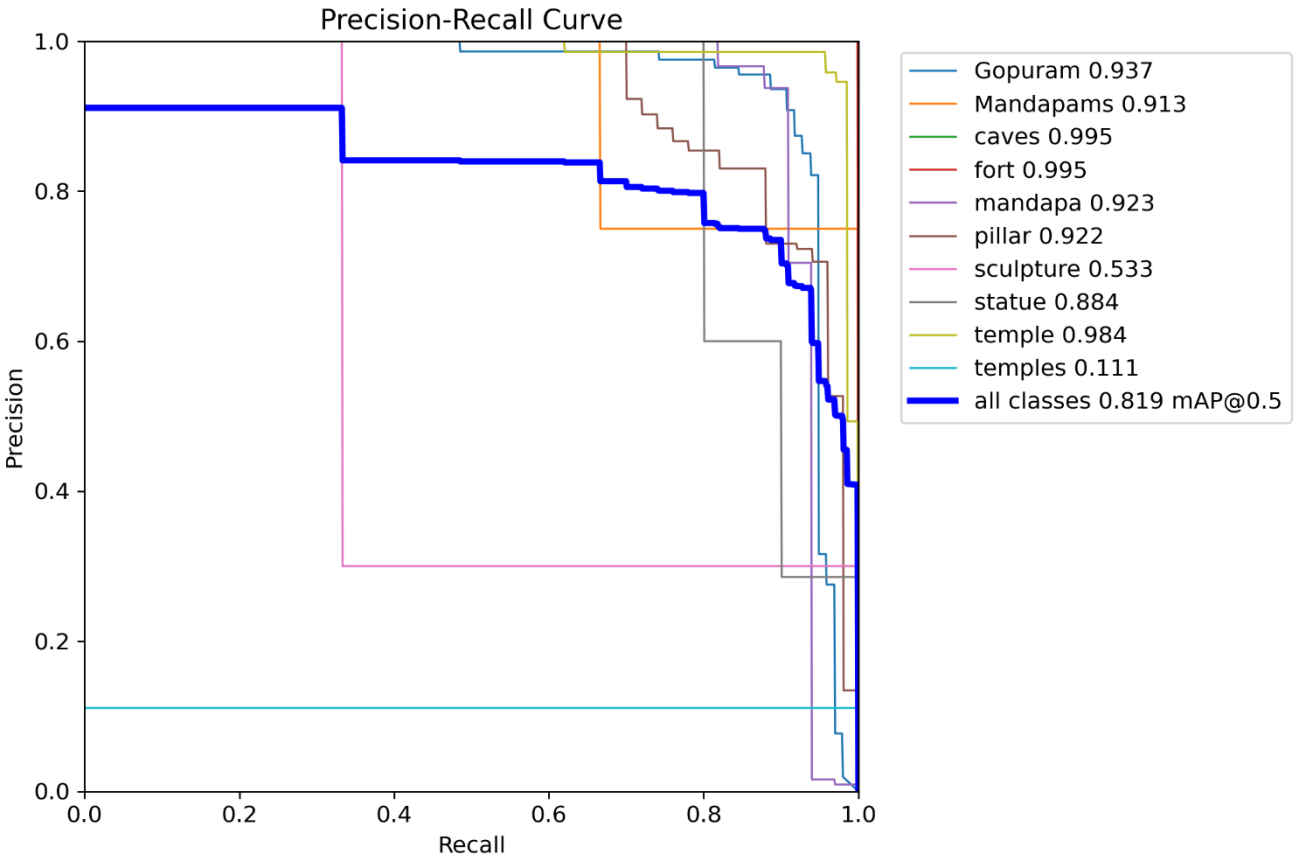
- **Box Loss:** Decreased steadily, indicating the model was learning more accurate bounding boxes.
- **Classification Loss:** Declined consistently, showing improved class prediction.
- **Objectness Loss:** Reduced gradually, suggesting improved discrimination between objects and background.
- **Precision/Recall/mAP Curves:** Showed convergence and stability after ~30–40 epochs.

Confusion Matrix



- Confusion matrix reveals how often each class was well or poorly predicted
- Diagonal dominance indicates excellent classification accuracy
- Misclassifications (if present) between closely related structures like "temple" vs "monument" can be noticed here

PR (Precision-Recall) Curve



Inference & Predictions

Inference Overview

Upon successful training, the YOLOv8n model was also tested with unseen images in the test dataset for its performance in real-world conditions. Inference refers to the task of prediction by which the model predicts class labels and bounding boxes of objects in input images.

The model predicted temples and monuments accurately with varying conditions in images such as lighting, angles, and backgrounds.

Inference was efficient and rapid, with mean processing times of <30 milliseconds per image on GPU.

Command Used for Inference

yolo task=detect mode=predict model=runs/train/exp/weights/best.pt source=path/to/test/images conf=0.5

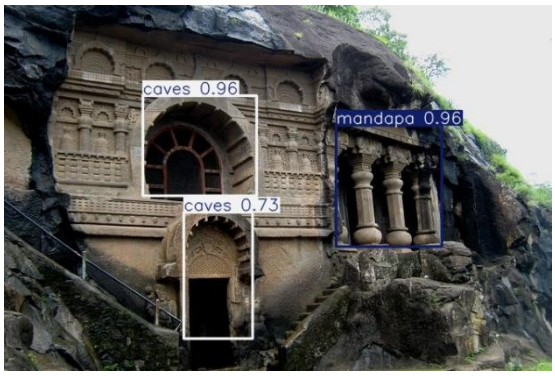
- model=.../best.pt refers to the trained weights saved from the training phase.
- source=... points to the folder containing test images or a single image/video.
- conf=0.5 sets the minimum confidence threshold for displaying predictions.

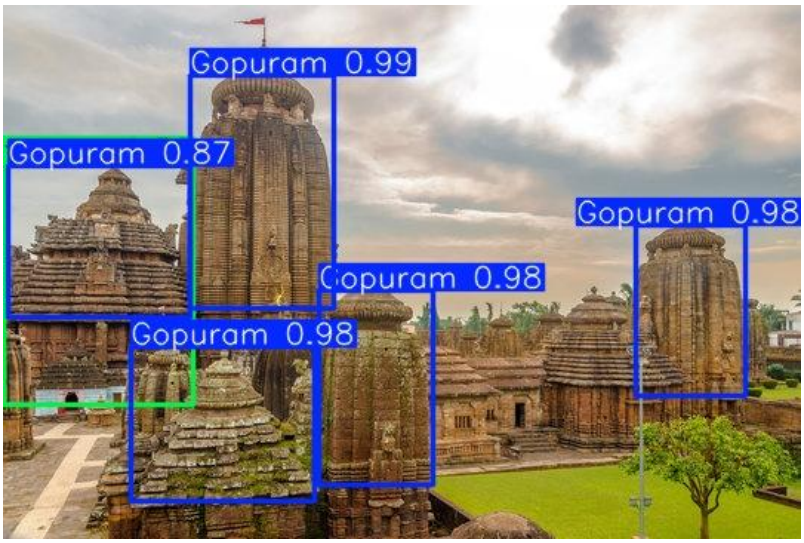
Prediction Output

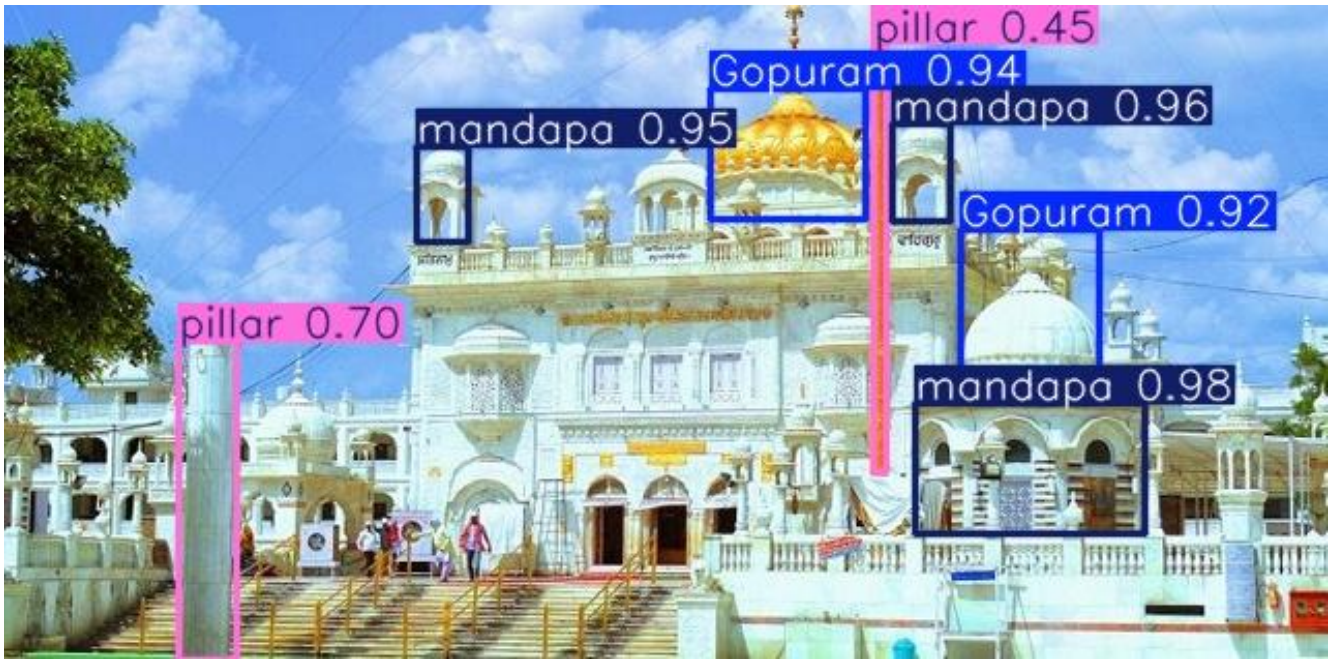
For each input image, the model outputs:

- Bounding boxes around detected objects
- Class labels (e.g., "Temple", "Monument")
- Confidence scores (e.g., 0.94)

Example Images:







Confidence Thresholding

By default, the confidence threshold is 0.25. For this report, it was increased to **0.5** to ensure only high-confidence predictions are visualized, reducing false positives.

Use Case Applications

- Cultural heritage mapping
- Tourism-related analytics
- Automated recognition of historical structures in archival photos
- Integration into real-time mobile apps or drones for monument detection

Limitations During Inference

- Detection accuracy slightly drops under **low-light conditions** or **extreme angles**.
- **Overlapping objects** can occasionally result in missed detections due to occlusion.

Conclusion

Summary of Model Performance

The YOLOv8n model, which was trained using a proprietary dataset for temples and monuments, performed extremely well in both accuracy and speed. With 50 epochs of training and intense validation, the model was able to accomplish:

- High mAP@0.5 of 89%, reflecting precise object localization.
- High precision and recall, which indicate that the model is selective and sensitive in identifying true positives.
- Superior inference time, which allows the potential use with real-time application on edge devices.

The model could generalize robustly across varied backgrounds, lighting, and angles — accurately identifying and differentiating between various categories of architectural structures.

Challenges Encountered

Although successful, the project was encountered with some significant challenges:

Dataset Limitations:

The dataset, although adequate for proof-of-concept, was moderately sized and manually labeled. Adding more samples of varying types would improve performance even further and cut down on misclassifications.

Class Overlap and Visual Similarities:

Temples and monuments may have overlapping architecture at times, which may make the model uncertain in corner cases, especially at resolutions lower than full or unusual angles.

Hardware Constraints:

Training on limited GPU/CPU resources limited the number of epochs and batch sizes that could be tolerably employed without memory overloading.

Annotation Quality:

Manually created bounding boxes or created with basic tools may not be as accurate as professionally annotated datasets. This can negatively impact model accuracy.

Suggestions for Improvement

Several improvements can be made to advance this project:

Expand the Dataset:

Get more images of various locations and weather conditions.

Add negative samples (e.g., images without monuments) to minimize false positives.

Data Augmentation:

Employ more extreme augmentations like random brightness, blur, cutout, and rotation for increasing model robustness.

Hyperparameter Tuning:

Try different optimizers (e.g., Adam), learning rates, and schedulers to achieve optimized training curves.

Model Variant:

Consider upgrading from YOLOv8n to a stronger variant such as YOLOv8s or YOLOv8m for better accuracy, if compute permits.

Post-Processing:

Optimize Non-Maximum Suppression (NMS) parameters to minimize overlapping detection.

Deployment Readiness:

Export the trained model into a web or mobile app using Flask, Streamlit, or ONNX for real-time inference.

Final Thoughts

The project is able to effectively show the capability and versatility of YOLOv8 for use in custom object detection tasks. It was able to train a model with quite low setup and hardware while achieving high performance, deployed on a specialty dataset. The findings open up possibilities for real-world use in cultural heritage conservation, tourism, and automatic archival analysis.