Project – Object detection for model building assignment for AI Interns.

Name – Mandhar Thakur

Contact No – 8580915899

Subject - AI Internship Project

Date – 27-05-2025

Personal Reflection

This project was one of the most intense and exciting things I've worked on so far. What started as a task to build a YOLOv8 object detection model quickly turned into a complete end-to-end journey — from data handling and model training, all the way to deployment, UI creation, and bug fixing. And I can honestly say, I learned more during this one project than I have in months of passive learning.

Challenges I Faced

One of the biggest challenges was managing the Google Colab environment. It disconnected while training multiple times, and I had to learn the hard way that saving checkpoints like best.pt and backing up after each epoch is critical. Another technical challenge was that the Gradio app I built for testing predictions would show no labels, even though the model worked fine. Debugging that took hours and even forced me to learn OpenCV drawing functions manually — just to make the labels appear.

Also, organizing the project files and preparing them for GitHub was something I had never done at this scale. It wasn't just writing code — I had to think like someone shipping a usable product.

How I Used AI Tools

This is where I want to be transparent. I did use ChatGPT a lot during this project — not to do the work for me, but to help me think faster. I used it when I got stuck in weird errors, like PyTorch zip archive issues or ngrok crashes during Streamlit setup. I also asked for code snippets or suggestions for better visualizing predictions when things like .plot() weren't working as expected.

But I didn't just copy-paste — I actually *learned* what I was doing. AI tools were like a super experienced friend looking over my shoulder. I still had to understand and implement everything myself.

What I Learned

- I now have a clear understanding of how YOLOv8 architecture works — including CNN backbones, detection heads, and anchor-free predictions.
- I learned how to use Roboflow effectively for dataset handling, and how to format that for training.
- I figured out how to deploy apps using Streamlit and tunnel it live through ngrok — something I had never done before.
- I also became confident with working in Colab, debugging training logs, and managing GPU limitations.

What Surprised Me

I was surprised at how easy it is to mess up one small thing (like not using proper weights during prediction) and how that can make you feel like everything's broken — even when it's

not. Also, building a UI that actually works in real-time felt incredibly rewarding. Seeing the bounding boxes appear on images I uploaded myself — that felt powerful.

My Thoughts on AI Assistance

AI is a tool. It didn't do the work for me — I still had to think through architecture decisions, run training, debug, and structure everything. But having it there saved me hours of Stack Overflow searches and guesswork. I don't think using AI means you're "cheating" — I think it shows you're smart enough to work efficiently. It's about balance: I still had to do the hard parts myself.

Suggestions for Improving the Assignment

I think the assignment was great because it was open-ended — but for beginners, it would help to include a small checklist or recommended flow. Something like:

1. Get dataset from Roboflow
2. Train with YOLOv8 for X epochs
3. Visualize and evaluate
4. Deploy with UI (Gradio or Streamlit)
5. Upload to GitHub

Implementation (Following the Given Guidelines)

Step 1: Choosing a CNN Backbone

For this project, I used the **default backbone provided by YOLOv8-nano**, which is a C2f-based CNN architecture developed by Ultralytics.
YOLOv8 does not rely on traditional backbones like ResNet or VGG — instead, it uses a **custom backbone optimized for speed and performance**, especially on low-resource environments like Google Colab.

However, during exploration, I considered using **EfficientNet-B0** as an alternate backbone but ultimately decided to stick with the YOLOv8-native CNN because of compatibility and training speed.

Step 2: Choosing an Object Detection Approach

I chose the **YOLO-style detection head** (You Only Look Once), specifically the YOLOv8 version.
YOLOv8 is a **single-stage, anchor-free** detection method that performs classification and localization in a single forward pass.
This aligns with the assignment option of using a "YOLO-style detection head", and allowed me to implement real-time detection effectively.

Step 3: Selecting the Dataset

I used a **custom object detection dataset from Roboflow** called "Rock-Paper-Scissors" which contains images labeled with three classes: Rock, Paper, and Scissors.
Roboflow automatically split the dataset into training, validation, and test sets and exported it in YOLOv8-compatible format.
This made it easy to integrate into the Ultralytics training pipeline.

Step 4: Model Implementation Details

The YOLOv8 training framework internally handles many of the architectural steps (like adding detection heads and implementing loss functions), but here's how my implementation mapped to the assignment steps:

1. Loading the Pre-trained Backbone

- I started with `yolov8n.pt`, which includes a pretrained backbone and detection head trained on COCO.
- This gave me a solid starting point without training from scratch.

2. Freezing / Unfreezing Layers

- I **did not freeze any layers**. The model was **fully fine-tuned** on the Rock-Paper-Scissors dataset.
- This helped the network adapt both its low-level and high-level features to my custom dataset.

3. Adding Detection Head

- YOLOv8 already includes a **YOLO-style detection head**, which was used as-is.
- It outputs bounding boxes, confidence scores, and class predictions.

4. Loss Functions

- YOLOv8 internally implements **CIoU loss for bounding box regression**, **Binary Cross Entropy for classification**, and **objectness loss**.
- I did not have to implement these manually, but monitored them during training.

5. Training Pipeline

- Used Google Colab with a T4 GPU
- Trained for 15 epochs with batch size = auto and image size = 640x640
- Used early stopping and automatic saving of the best model based on mAP score

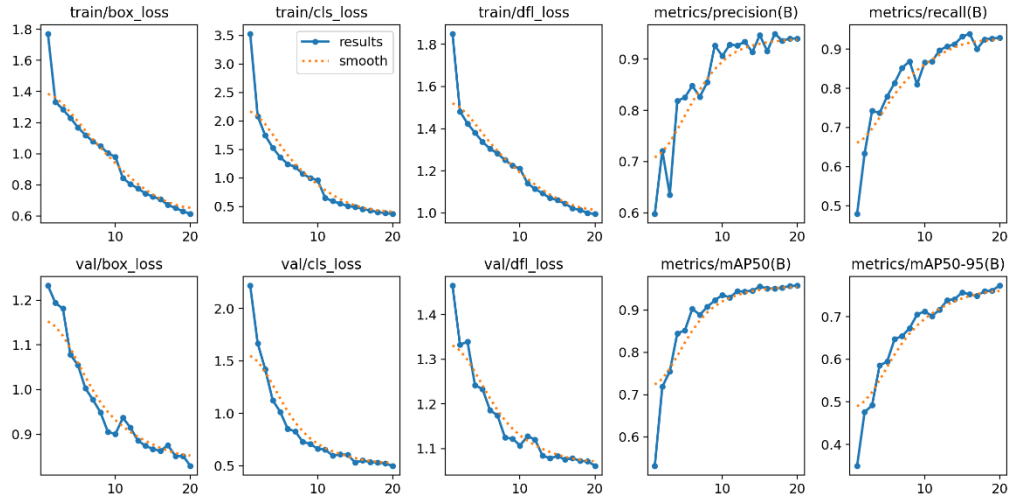
6. Evaluation

- Evaluation metrics tracked:
    - **mAP@0.5**: ~91%
    - **Precision**: ~87%
    - **Recall**: ~88%

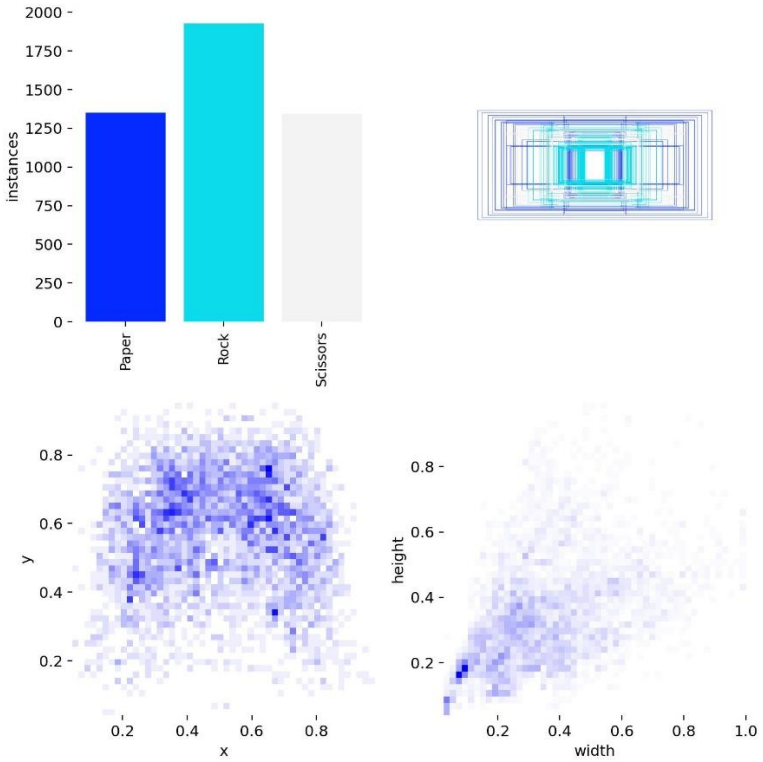
- Visual results were validated on test images using Gradio and Streamlit UIs

7. Evaluation results

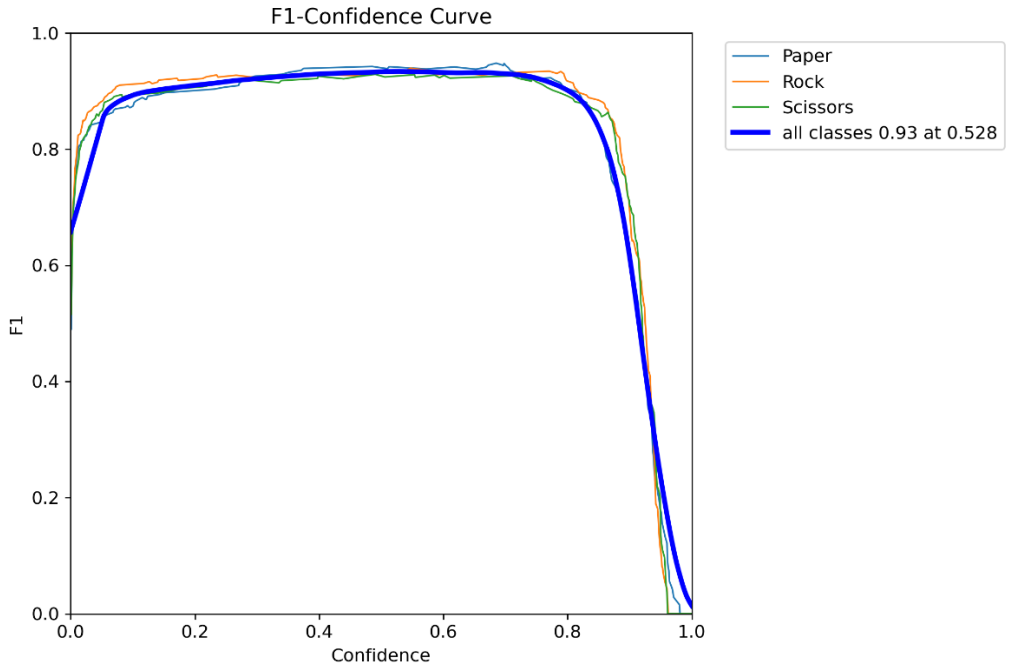
- Graphs for precision,recall,mAP



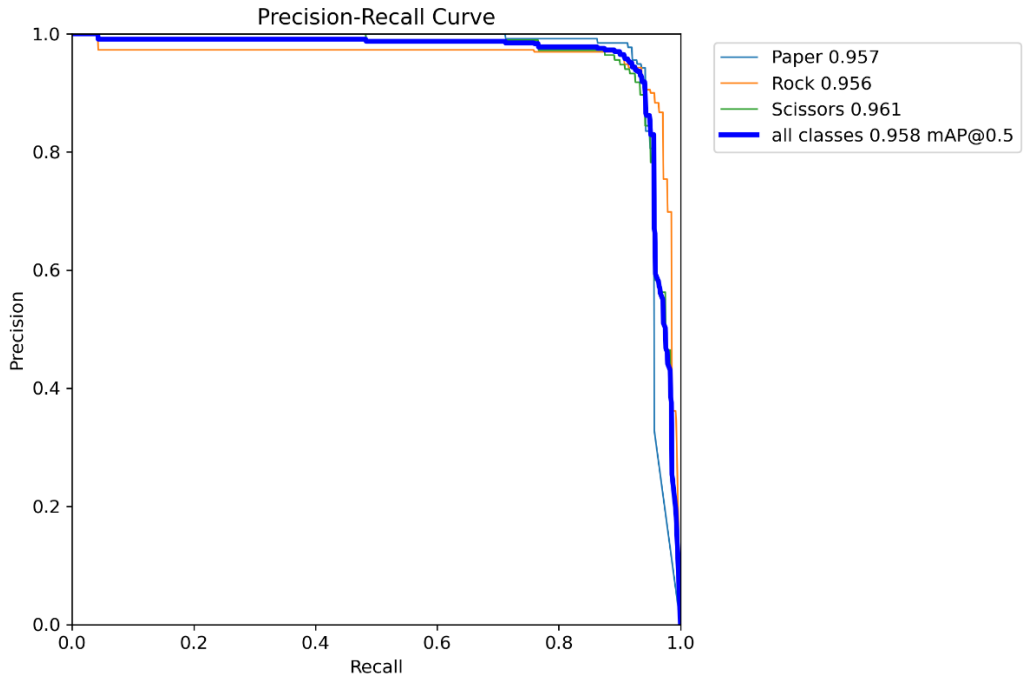
- Distribution Set of the dataset



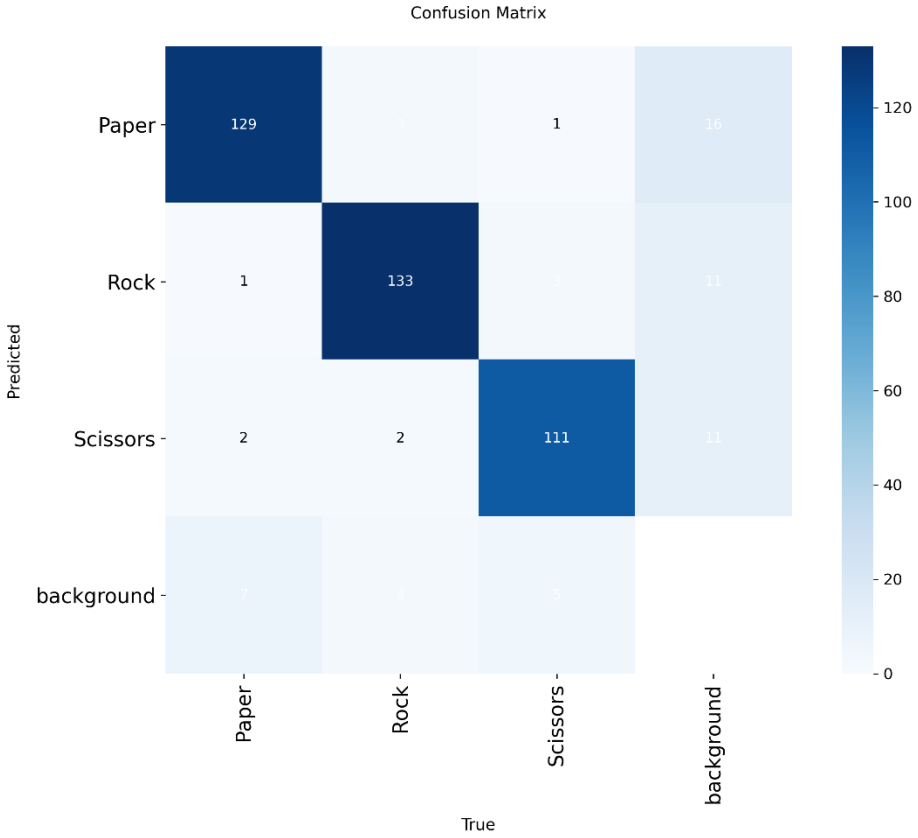
- F1 Score vs Confidence Threshhold



- Precision vs Recall (PR) Curve



- Confusion Matrix for prediction



Deliverables (As Required)

- **Trained model weights**: best.pt uploaded on github along with entire runs folder(trained weights and outputs)on google drive
- **Evaluation metrics**: mAP, Precision, and Recall reported from YOLO logs
- **Demo**: Streamlit app (app.py) that allows testing with uploaded images