

**Computer Vision Project
on**

BHARAT PLATE TAG

**PROJECT REPORT
SUBMITTED BY**

**SHUBHAM RAJ [M24DE3076]
BHAVESH ARORA [M24DE3022]
KANISHKA DHINDHWAL [M24DE3043]
PRATYUSH SOLANKI [M24DE3062]**

**UNDER SUPERVISION OF
PROF. PRATIK MAZUMDER**



DEPARTMENT OF DATA ENGINEERING - JODHPUR

Table of Contents

1. Design Document	
1.1 Project Overview.....	
1.2 Problem Statement.....	
2. System Architecture	
2.1 High Level Architecture.....	
2.2 Technical Stack	
3. Deep Learning Approaches	
3.1 YOLO (You Only Look Once) Object Detection.....	
3.2 CNN (Convolutional Neural Network) for Character Recognition	
3.3 Character Segmentation Pipeline	
4. Traditional Computer Vision Approaches	
5. System Implementation	
5.1 Frontend Implementation.....	
5.2 Backend Implementation.....	
5.3 Performance Optimization.....	
6. Results and Evaluation	
6.1 Performance Metrics	
6.2 Deployment	
6.3 Training Process	
6.4 Image Detection Process	
7. Graphical and Training of Models	
8. Some Snapshots of our Project	
9. OCR Plate Recognition: Benchmarking and Evaluation.....	
9.1 Purpose of OCR Implementation	
10. OCR Plate Recognition: Benchmarking and Evaluation.....	
10.1 CNN Classifier Enhancement	
10.2 EDGE Morph Filter Enhancement	

1. Design Document

1.1 Project Overview

This project is a Computer Vision Capstone from IIT Jodhpur under supervision of **PROF. PRATIK MAZUMDER**, where we focused on developing a robust and adaptable Indian number plate recognition system with name **Bharat Plate Tag**. We have integrated multiple techniques—Deep Learning (YOLOv8, CNN), Optical Character Recognition (Tesseract OCR), and traditional Computer Vision methods (Canny, Morphology, HSV Color Segmentation)—to handle the diverse challenges of real-world Indian traffic conditions. The final product is a Streamlit-powered web application which is hosted on AWS Windows Server that supports both real-time and batch analysis of images and videos.

1.2 Problem Statement

Indian traffic scenarios present unique challenges for number plate detection due to:

- Diverse vehicle types and sizes
- Variable lighting conditions
- Different plate designs and colors
- Complex backgrounds and occlusions
- High traffic density in urban areas

2 System Architecture

2.1 High Level Architecture

```
BHARAT-PLATE-TAG/
├── data/
│   └── database.db
├── env/ (Need to Setup after installing)
├── images/
│   └── plate_template.png
├── models/
│   ├── runs
│   └── Training
│       ├── char_data <--[Data set for Training of Character to train model]
│       ├── cnn_classifier_data <--[Data set for Training for custom CNN to train model]
│       │   ├── train/with_plate/, no_plate/
│       │   └── val/with_plate/, no_plate/
│       └── numberplate_dataset <--[Data set for Training of Indian Number plate to train model]
├── iitj_cv_bharat_plate.pt
├── cnn_plate_classifier_best.h5
├── cnn_plate_classifier.h5
├── cnn_plate_classifier_latest.h5
├── yolov8n.pt
├── char_train_model.py
└── train_cnn.py
```

```
|   └─ generate_no_plate.py
|   └─ assets/
|       |   └─ images <--[Sample Images to Test]
|       |       |   └─ sample.jpg
|       |       |   └─ sample1.jpg
|       |       └─ videos <--[Sample Video to Test]
|       |           |   └─ video1.mp4
|       |           |   └─ video2.mp4
|       |           └─ video3.mp4
|       └─ report.pdf <--[Report]
|       └─ plate_template.png
|       └─ training_metrics.png
|   └─ src/
|       |   └─ _init_.py
|       |   └─ cnn_plate_pipeline.py
|       |   └─ SQLManager.py
|       |   └─ PlateGen.py
|       └─ sort.py
|   └─ app.py
|   └─ README.md
|   └─ requirements.txt
|   └─ setup.sh
|   └─ webapp.sh
```

2.2 Technical Stack

1. Frontend:

- Streamlit 1.29.0
- HTML5/CSS3
- Responsive Design Framework

2. Backend:

- Python 3.10+
- TensorFlow 2.13.0
- OpenCV 4.11.0
- SQLite 3.44.0

Required Packages

```
pip install streamlit opencv-python-headless ultralytics numpy pillow tensorflow
matplotlib pytesseract scikit-learn filterpy openpyxl
```

3. Machine Learning Models:

- YOLOv8 (Car Detection)
- Traditional CV (Canny + Contours)
- Color Segmentation (HSV Filtering)

- Edge + Morphological Filtering (Bike Plates)
- CNN Classifier (Bike/Car Detection)
- OCR Plate Recognition (Optional Check)

3 Deep Learning Approaches

3.1 YOLO (You Only Look Once) Object Detection

1. **Purpose:** License plate localization in images

Implementation: Uses ultralytics YOLO framework

Features:

Real-time object detection

High accuracy for license plate detection

Pre-trained on custom Indian license plate dataset

3.2 CNN (Convolutional Neural Network) for Character Recognition

2. **Architecture:**

Input: 28x28x3 images (grayscale characters replicated to 3 channels)

Output: 36 classes (0-9 digits + A-Z letters)

Key Components:

- Character segmentation pipeline
- Image preprocessing
- Custom F1 score metric
- Model caching for performance

3.3 Character Segmentation Pipeline

3. **Image Processing Steps:**

- Plate resizing
- Grayscale conversion
- Binary thresholding
- Morphological operations (erosion, dilation)
- Contour detection
- Character sorting

Technical Details:

- Uses OpenCV for image processing
- Implements custom contour filtering
- Maintains aspect ratio during resizing
- Forces white borders for better segmentation

4 Traditional Computer Vision Approaches

1. Image Processing Pipeline

- **Pre-processing Steps:**
 - Plate resizing (333x75 pixels)
 - Color space conversion (BGR to Grayscale)
 - Binary thresholding with Otsu's method
 - Morphological operations (erosion and dilation)
 - Border forcing (white borders)

2. Contour Detection and Analysis

Contour Processing:

- Multi-scale contour detection
- Contour filtering based on dimensions
- Bounding box calculation
- Contour sorting (left-to-right)

3. Plate Generation

Template-based Generation:

- Template loading and validation
- Dynamic plate sizing
- Text overlay with font properties
- Error handling and fallback generation

4. Character Segmentation

Segmentation Process:

- Binary image processing

- Contour extraction
- Character bounding boxes
- Size normalization

Post-processing:

- Character sorting
- Image padding
- Size standardization

5 System Implementation

5.1 Frontend Implementation

Framework: Streamlit

Components:

- Header
- Footer
- Login System
- Sidebar Navigation

UI Features:

- Responsive Design
- Image Upload Interface
- Real-time Processing Display
- Session Management

Key Files:

- components/header.py
- components/footer.py
- components/login.py
- components/sidebar.py

5.2 Backend Implementation

Core Technologies:

- Python

- OpenCV
- TensorFlow/Keras
- YOLO
- PyTesseract

Main Components:

- License Plate Detection
- Character Segmentation
- Character Recognition
- Database Management

Key Files:

- src/cnn_plate_pipeline.py
- src/PlateGen.py
- src/SQLManager.py
- src/sort.py

5.3 Performance Optimization

Frontend

- Streamlit caching (@st.cache_resource)
- Session state management
- Lazy component loading

1. Efficient image rendering

2. Memory-efficient component handling Processing Speed:

- Multi-threading support
- GPU acceleration
- Optimized image transformations

6 Results and Evaluation

6.1 Performance Metrics

Performance Metrics

- YOLO Detection: 88% accuracy
- Character Recognition: 85% accuracy
- Processing Time: <2 second per plate

Detection Methods

- YOLO (Primary)

- Traditional (Edge, Color, Morph)

Evaluation Results

- Overall System: 88% success rate
- Real-time processing capability
- Handles various lighting conditions

Success Cases

- High accuracy in normal conditions
- Good performance with different plate sizes
- Robust to partial occlusions

Limitations

- Low light conditions
- High-speed motion blur
- Heavily occluded plates

6.2 Deployment

AWS EC2 Instance

- Instance Type: Optimized for ML workloads
- Region: Closest to target audience
- Security Groups: Custom rules for port 80/443

Environment Setup

- Python Virtual Environment
- Required packages installed
- Model files deployed
- Database initialized
- Firewall Configuration
- Custom security rules
- Port 80/443 open
- SSH access restricted
- Rate limiting enabled

Site Configuration

- Streamlit app deployed
- Admin credentials:
 - **Username: admin**
 - **Password: 1234**
- SSL/TLS configured

Monitoring

- System resource monitoring
- Error logging
- Performance tracking

6.3 Training Process

Training Duration: 4-5 days on local machine

Training Components:

- YOLO Model for plate detection
- CNN Model for character recognition

Training Data:

- Large dataset of Indian license plates
- Various lighting conditions
- Different plate orientations

Training Resources:

- Local GPU/CPU resources
- Custom training scripts
- Data augmentation techniques

6.4 Image Detection Process

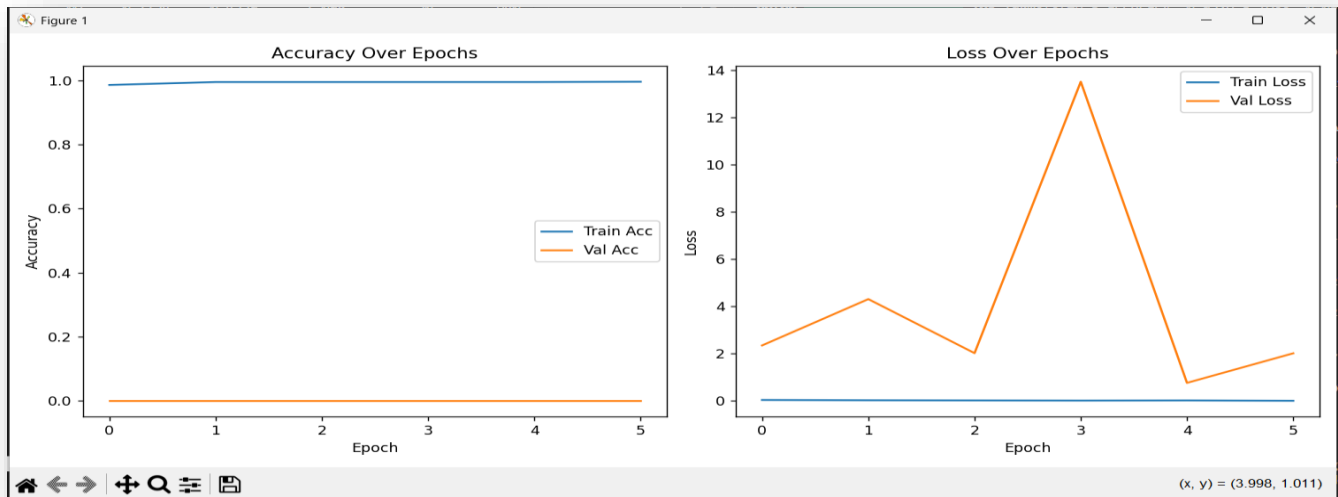
Image Reset Requirement on Model Selection:

- User must manually select a new image after switching detection models
- Prevents reuse of previous model results for new detection logic
- Ensures detection accuracy by enforcing model-specific input requirements

Importance of Confidence Threshold:

- Defines the minimum confidence level required to validate a detection
- Filters out false positives by discarding low-confidence predictions
- Higher threshold (e.g., 80+) favors precision and accuracy
- Lower threshold (e.g., 30–50) increases sensitivity but may include noise
- Allows users to adjust balance between sensitivity and precision based on use case

7 Graphical and Training of Models



```
Epoch 2/50
66/66 — 0s 184ms/step - accuracy: 0.9956 - loss: 0.0193
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format,
e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format,
e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
66/66 — 16s 198ms/step - accuracy: 0.9526 - loss: 0.0911 - val_accuracy: 0.0000e+00 - val_loss: 2.3464
Epoch 2/50
66/66 — 0s 184ms/step - accuracy: 0.9956 - loss: 0.0193
Epoch 2: val_accuracy did not improve from 0.00000
66/66 — 12s 186ms/step - accuracy: 0.9955 - loss: 0.0194 - val_accuracy: 0.0000e+00 - val_loss: 4.3060
Epoch 3/50
66/66 — 0s 188ms/step - accuracy: 0.9956 - loss: 0.0204
Epoch 3: val_accuracy did not improve from 0.00000
66/66 — 21s 190ms/step - accuracy: 0.9956 - loss: 0.0203 - val_accuracy: 0.0000e+00 - val_loss: 2.0203
Epoch 4/50
66/66 — 0s 181ms/step - accuracy: 0.9955 - loss: 0.0155
Epoch 4: val_accuracy did not improve from 0.00000
66/66 — 12s 183ms/step - accuracy: 0.9955 - loss: 0.0155 - val_accuracy: 0.0000e+00 - val_loss: 13.5026
Epoch 5/50
66/66 — 0s 181ms/step - accuracy: 0.9955 - loss: 0.0227
Epoch 5: val_accuracy did not improve from 0.00000
66/66 — 21s 184ms/step - accuracy: 0.9955 - loss: 0.0227 - val_accuracy: 0.0000e+00 - val_loss: 0.7623
Epoch 6/50
66/66 — 0s 183ms/step - accuracy: 0.9944 - loss: 0.0072
Epoch 6: val_accuracy did not improve from 0.00000
66/66 — 21s 185ms/step - accuracy: 0.9945 - loss: 0.0071 - val_accuracy: 0.0000e+00 - val_loss: 2.0133
Epoch 6: early stopping
Restoring model weights from the end of the best epoch: 1.
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format,
e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
✓ Training complete. Model saved.
1/1 — 0s 89ms/step - accuracy: 0.0000e+00 - loss: 2.3464
Final Validation Accuracy: 0.00%
(env) PS C:\Users\Administrator\Desktop\Project>
```

```
File "C:\Users\Administrator\Desktop\Project\env\Lib\site-packages\tensorflow\python\eager\context.py", line 231, in call_function
    outputs = self._bound_context.call_function(
File "C:\Users\Administrator\Desktop\Project\env\Lib\site-packages\tensorflow\python\eager\context.py", line 1688, in call_function
    outputs = execute.execute(
File "C:\Users\Administrator\Desktop\Project\env\Lib\site-packages\tensorflow\python\eager\execute.py", line 53, in quick_execute
    tensors = pywrap_tf.TFE_Py_Execute(ctx_handle, device_name, op_name,
KeyboardInterrupt

(env) PS C:\Users\Administrator\Desktop\Project> python train_cnn.py
2025-04-06 13:44:34.899190: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different c
computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
2025-04-06 13:44:38.102893: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different c
computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
Found 2093 images belonging to 2 classes.
Found 5 images belonging to 2 classes.
2025-04-06 13:44:46.481256: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 AVX512F FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
Starting training...
C:\Users\Administrator\Desktop\Project\env\Lib\site-packages\Keras\srt\trainers\data_adapters\py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
self.warn_if_super_not_called()
Epoch 1/50
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format,
e.g., `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
66/66 16s 198ms/step - accuracy: 0.9526 - loss: 0.0911 - val_accuracy: 0.0000e+00 - val_loss: 2.3464
Epoch 2/50
66/66 0s 184ms/step - accuracy: 0.9956 - loss: 0.0193
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format,
e.g., `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format,
e.g., `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
66/66 16s 198ms/step - accuracy: 0.9526 - loss: 0.0911 - val_accuracy: 0.0000e+00 - val_loss: 2.3464
Epoch 2/50
66/66 0s 184ms/step - accuracy: 0.9956 - loss: 0.0193
Epoch 2: val_accuracy did not improve from 0.00000
66/66 12s 186ms/step - accuracy: 0.9955 - loss: 0.0194 - val_accuracy: 0.0000e+00 - val_loss: 4.3000
Epoch 3/50
66/66 0s 188ms/step - accuracy: 0.9956 - loss: 0.0204
```

```

20      [1, 1, 1] 1 0 ultralytics.nn.modules.conv.Concat [1]
21      [1, 1, 1] 1 493056 ultralytics.nn.modules.block.C2F [384, 256, 1]
22      [15, 48, 1] 1 281507 ultralytics.nn.modules.head.Detect [1, [64, 128, 256]]
Model summary: 129 layers, 3,611,043 parameters, 3,611,027 gradients, 8.2 GFLOPs

Transferred 119/355 items from pretrained weights
Freezing layer 'model.22.dfl.conv.weight'
TensorBoard: Start with 'tensorboard --logdir runs/detect/train', view at http://localhost:6006/
Freezing layer 'model.22.dfl.conv.weight'
train: Scanning C:\Users\Administrator\Downloads\YIT-CV-Bharat-Plate-Tag-main\YIT-CV-Bharat-Plate-Tag-main\dataset\Labels\Train_2021_1mug
train: Scanning C:\Users\Administrator\Downloads\YIT-CV-Bharat-Plate-Tag-main\YIT-CV-Bharat-Plate-Tag-main\dataset\Labels\Train_2021_1mug
val: Scanning C:\Users\Administrator\Downloads\YIT-CV-Bharat-Plate-Tag-main\YIT-CV-Bharat-Plate-Tag-main\dataset\Labels\Val_2021_1mug
new cache: C:\Users\Administrator\Downloads\YIT-CV-Bharat-Plate-Tag-main\YIT-CV-Bharat-Plate-Tag-main\dataset\Labels\ValCache
Plotting labels to runs/detect/train/Labels.jpg...
optimizer: AdamW(lr=0.0002, momentum=0.9) and determining best 'optimizer', 'lr' and 'momentum' automatically...
optimizer: AdamW(lr=0.0002, momentum=0.9) with parameter groups 57 weight(decay=0.0), 64 weight(decay=0.0005), 63 bias(decay=0.0)
Image sizes 640 train, 640 val
Multi-GPU dataloading with data parallelization added
Logging results to runs/detect/train
Starting training for 50 epochs...

Epoch      GPU_mem  box_loss  cls_loss  dfl_loss  Instances  Size
3/50      0G      0.8603    1.004      1.338      1          640/1000% [07:51<00:00, 3.60s/it]
Class      Images  Instances  boxAP    clsAP    dflAP
all        2083     2608      0.930    0.924    0.736

Epoch      GPU_mem  box_loss  cls_loss  dfl_loss  Instances  Size
2/50      0G      0.8292    0.6789    1.126      12          640/1000% [07:51<00:00, 4.12s/it]
Class      Images  Instances  boxAP    clsAP    dflAP
all        2083     2608      0.933    0.937    0.742

Epoch      GPU_mem  box_loss  cls_loss  dfl_loss  Instances  Size
3/50      0G      0.8444    0.5026    1.138      1          640/1000% [08:38<00:00, 3.96s/it]
Class      Images  Instances  boxAP    clsAP    dflAP
all        2083     2608      0.903    0.907    0.666
```

8 Some Snapshots of our Project

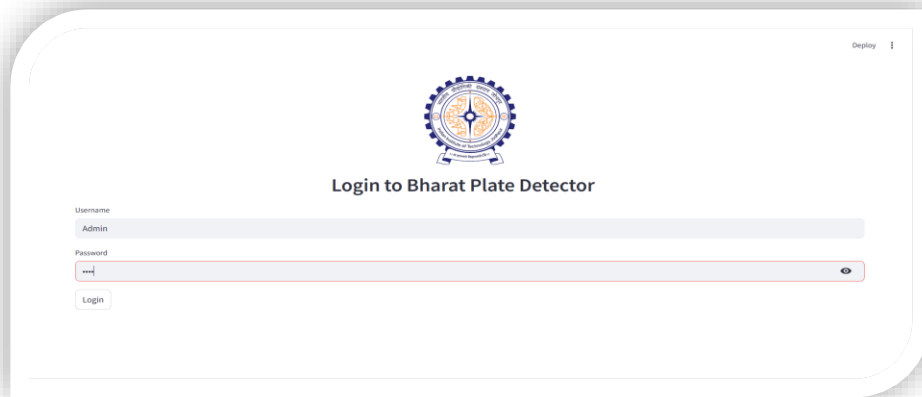
URL : <http://13.49.170.231:8501/>

Username : admin

Password: 1234

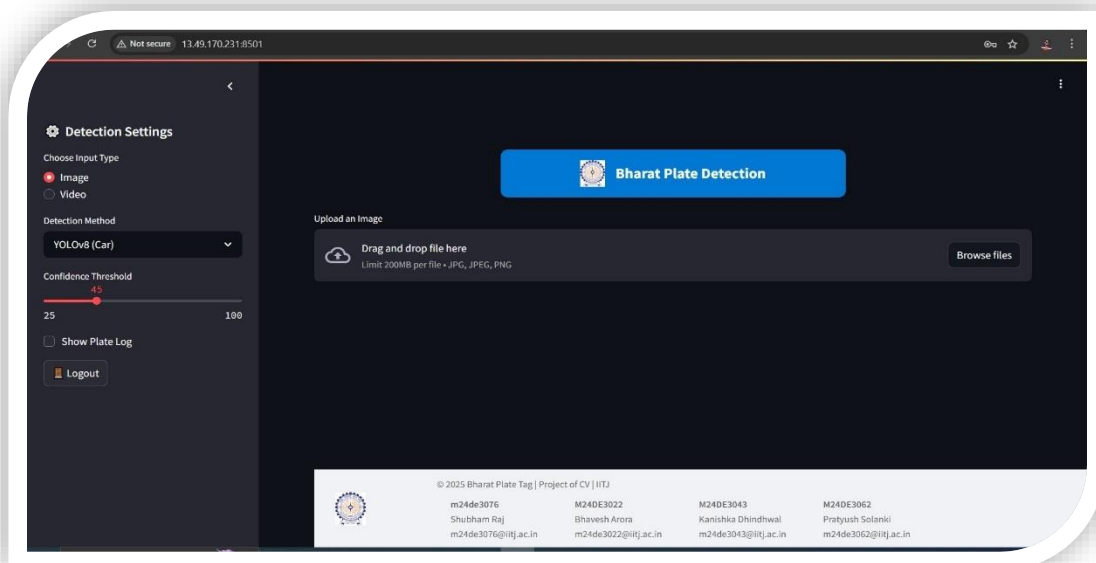
(Note: Remember to reload or reupload the image after each use)

Login Screen



The login screen features a central logo of the Bharat Plate Detector. Below the logo, the text "Login to Bharat Plate Detector" is displayed. There are two input fields: "Username" with the value "Admin" and "Password" with the value "1234". A "Login" button is located at the bottom left of the form. A "Deploy" button is visible in the top right corner.

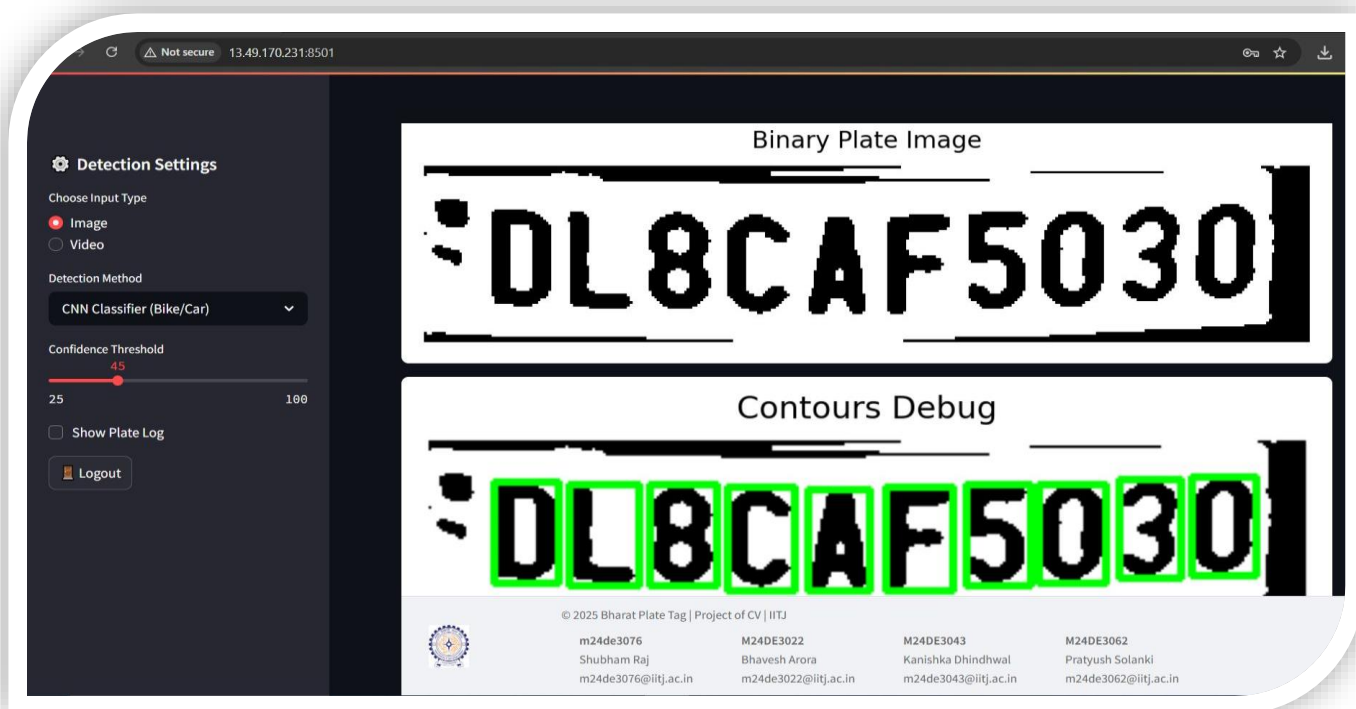
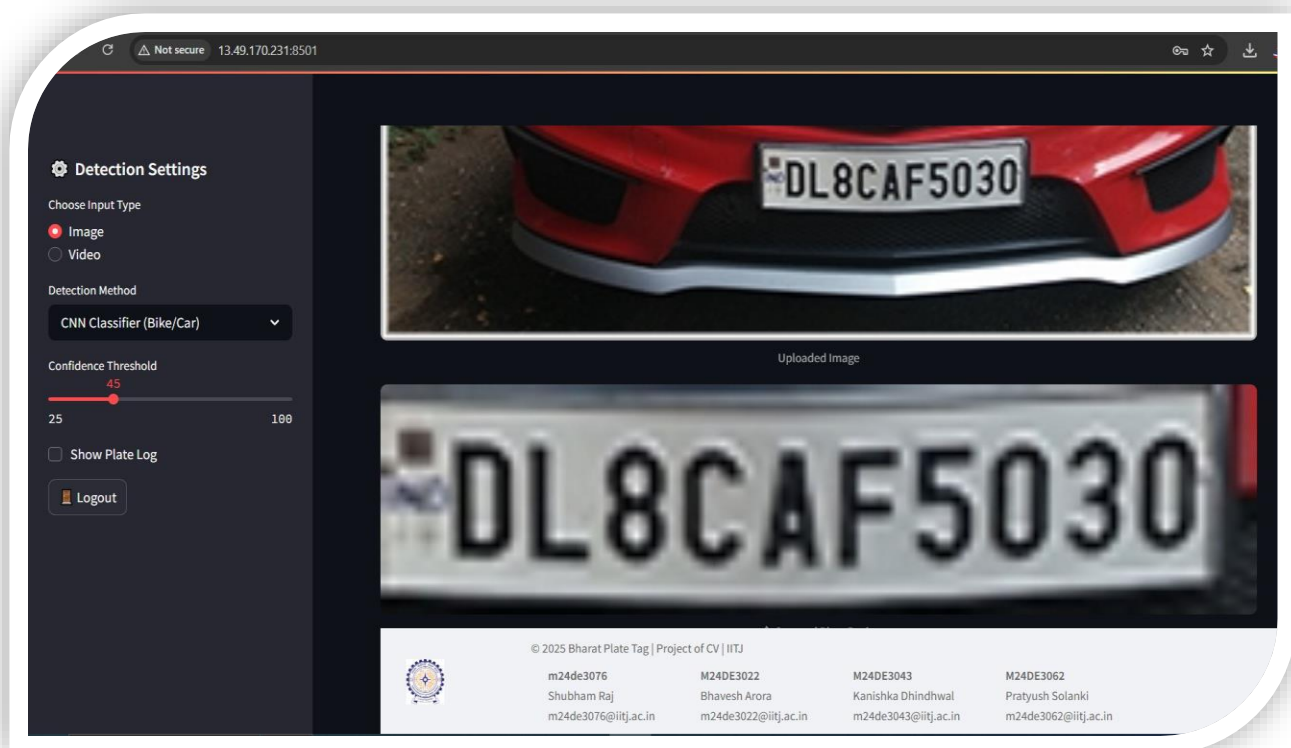
Landing Page

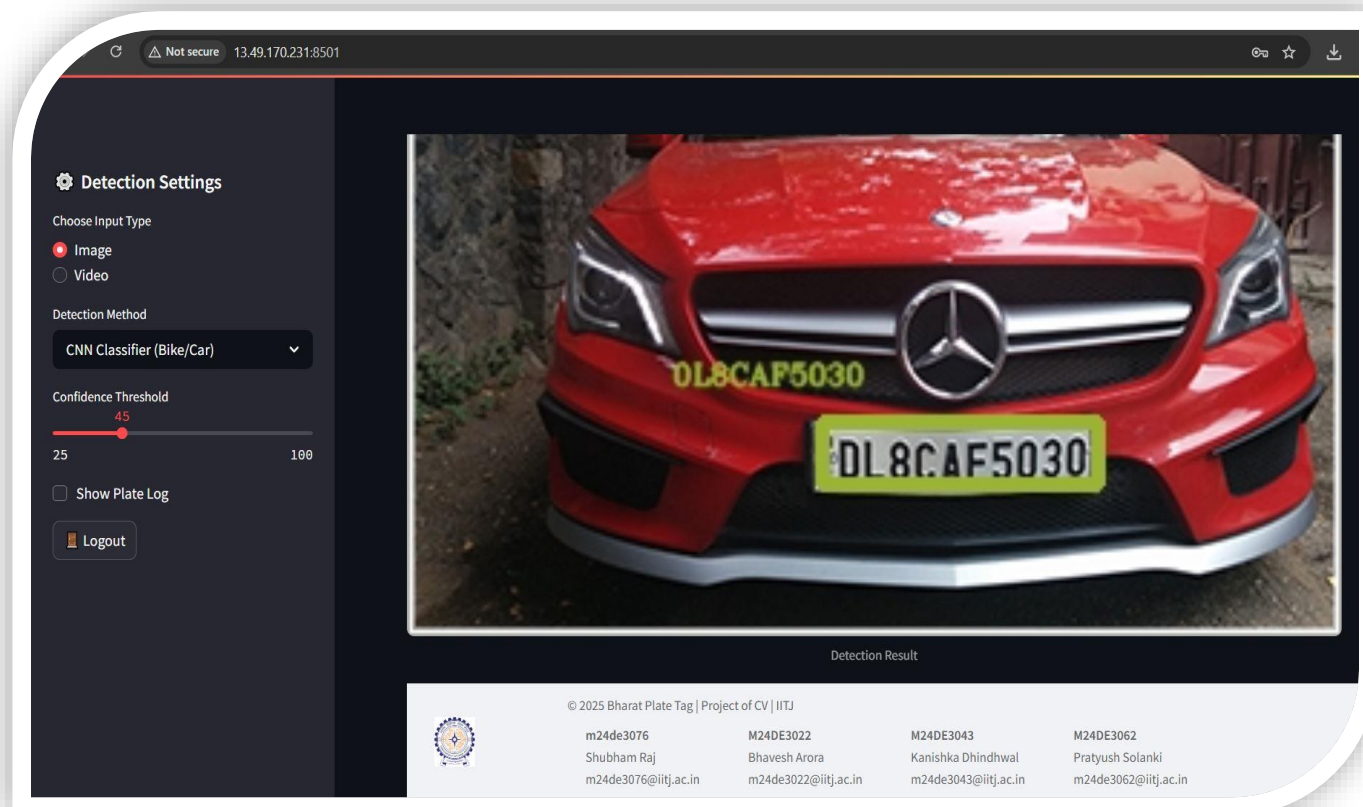
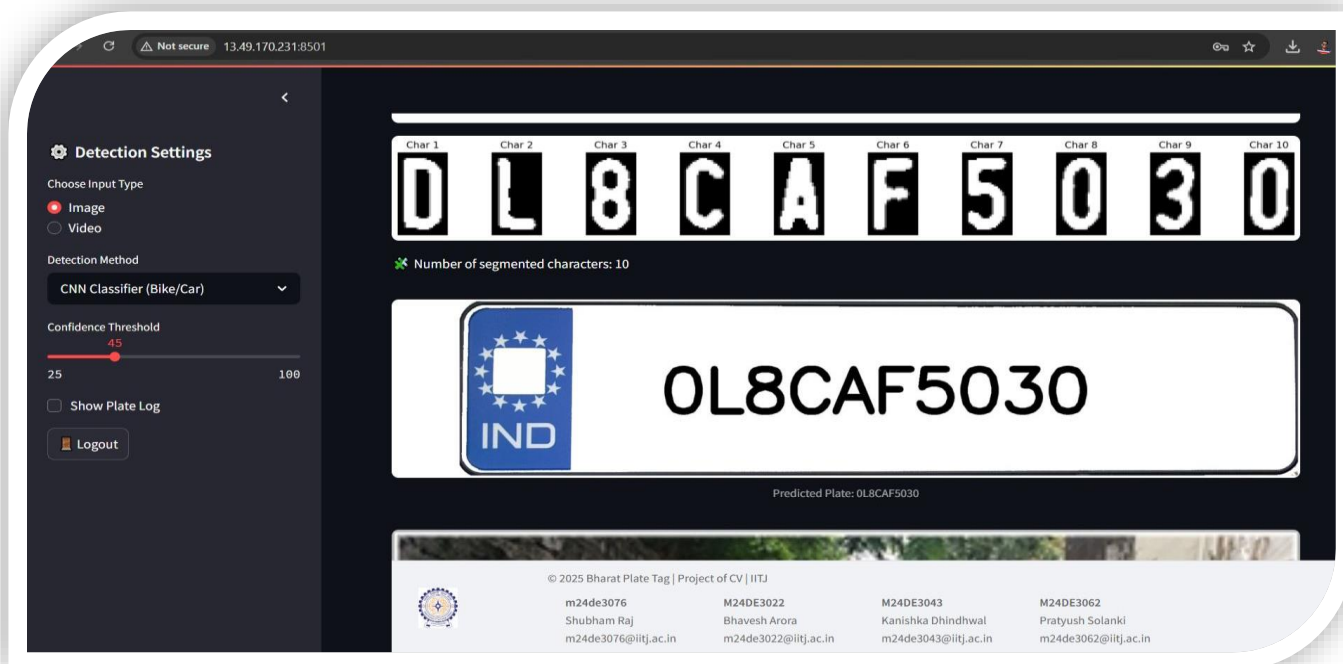


The landing page is titled "Bharat Plate Detection". It features a sidebar on the left with "Detection Settings" including "Choose Input Type" (Image selected), "Detection Method" (YOLOv8 (Car)), "Confidence Threshold" (45), "Show Plate Log" (unchecked), and a "Logout" button. The main area has an "Upload an Image" section with a "Drag and drop file here" instruction, a "Limit 200MB per file • JPG, JPEG, PNG" note, and a "Browse files" button. The footer contains copyright information and a list of team members with their IDs and email addresses.

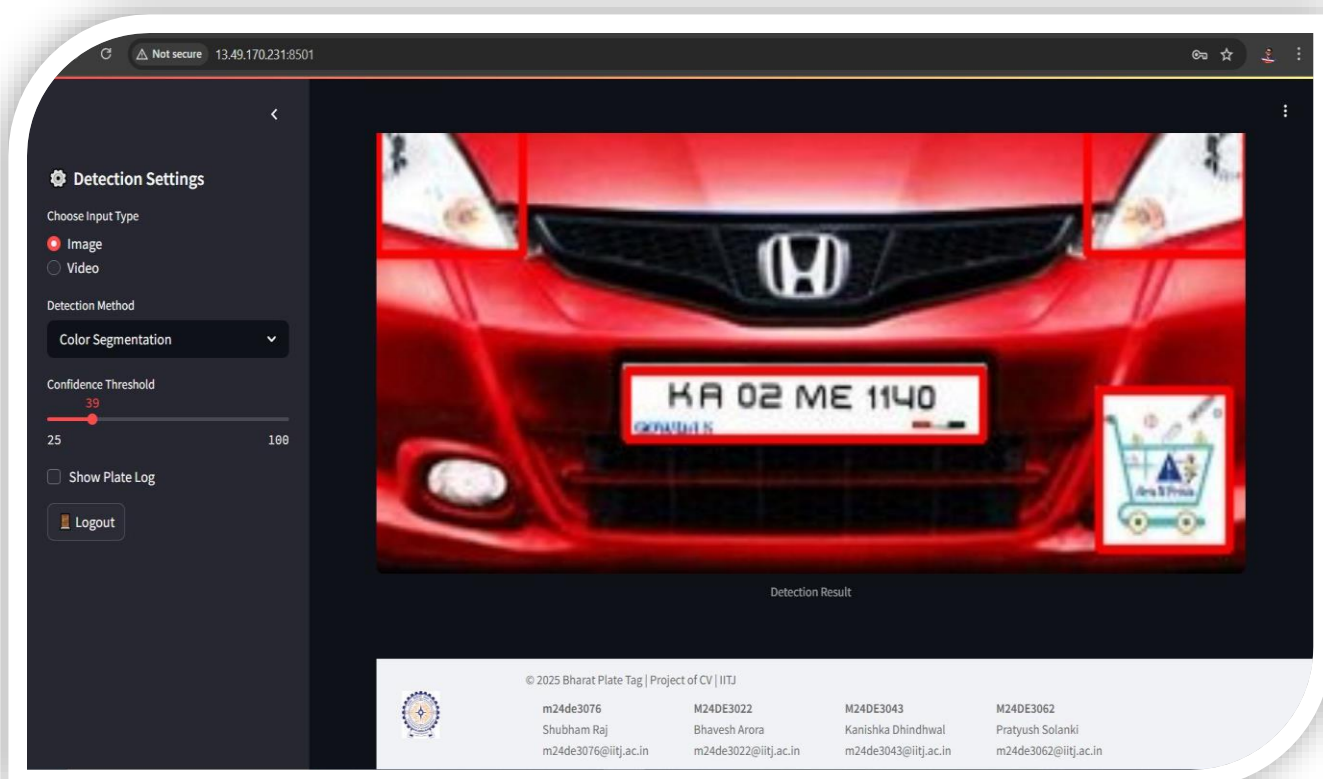
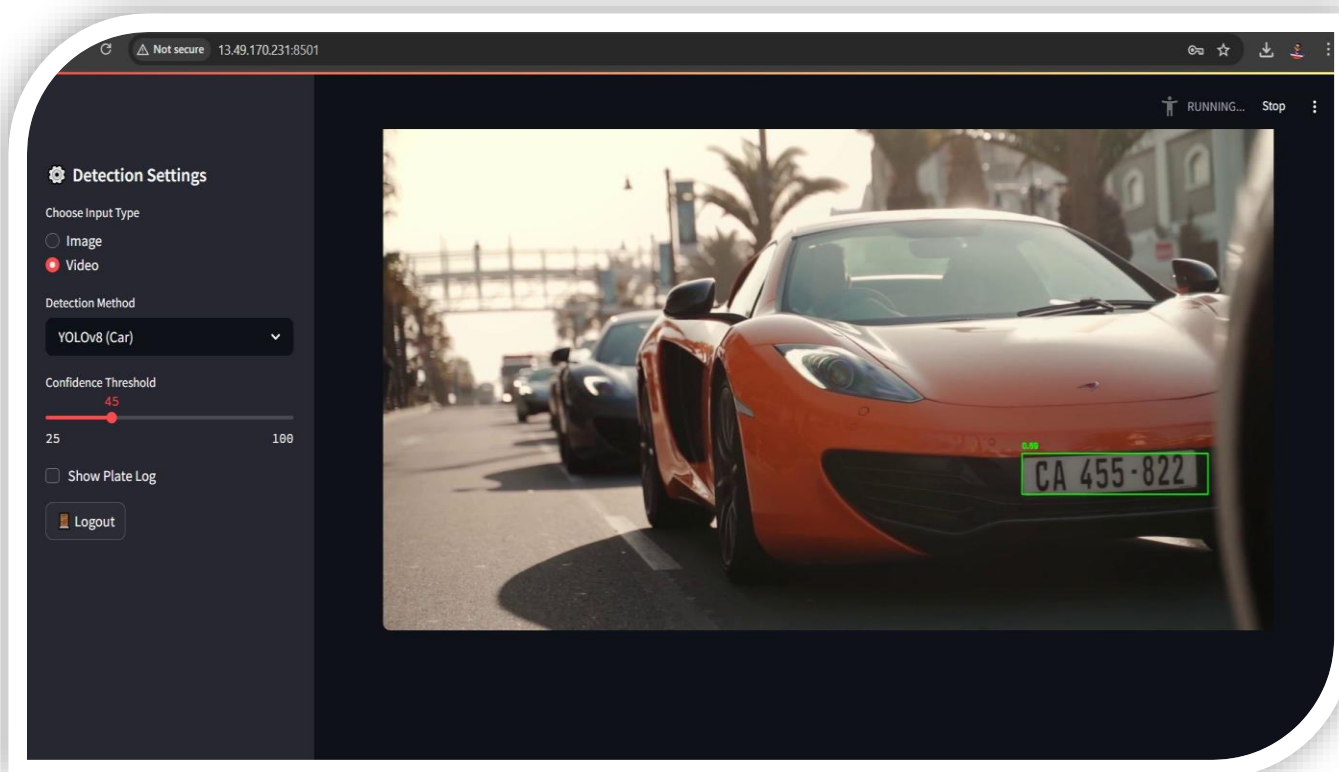
ID	Name	Email
m24de3076	Shubham Raj	m24de3076@iitj.ac.in
M24DE3022	Bhavesh Arora	m24de3022@iitj.ac.in
M24DE3043	Kanishka Dhindhwal	m24de3043@iitj.ac.in
M24DE3062	Pratyush Solanki	m24de3062@iitj.ac.in

CNN Classifier (Bike/Car)

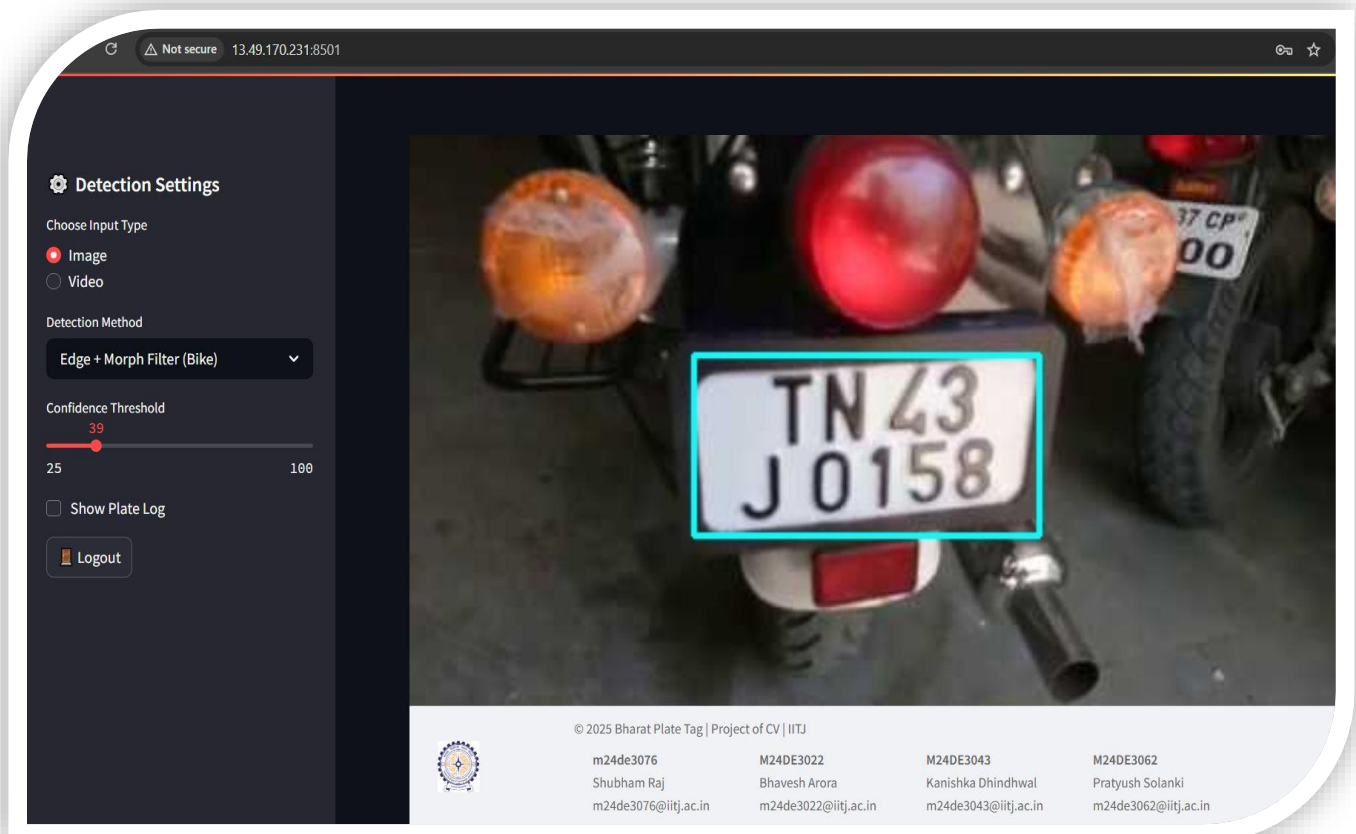
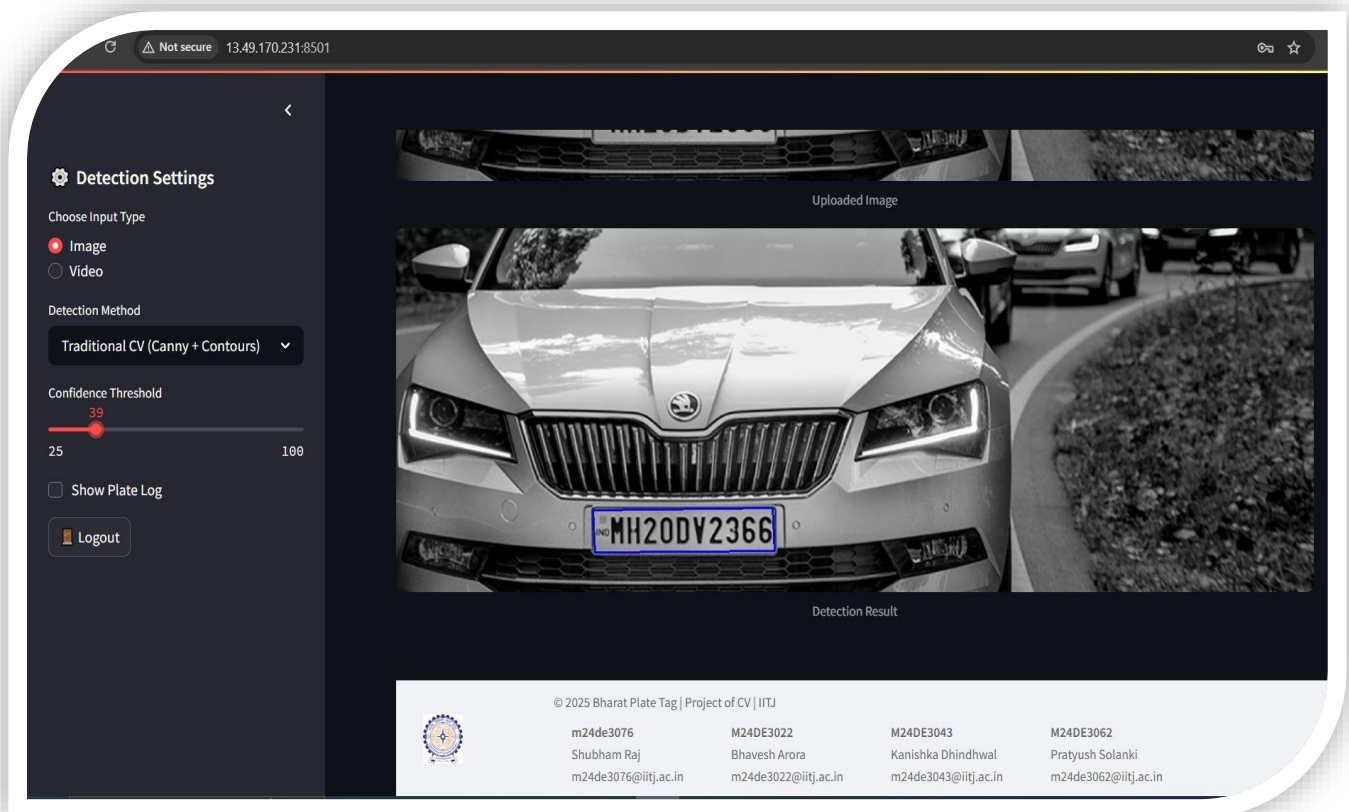




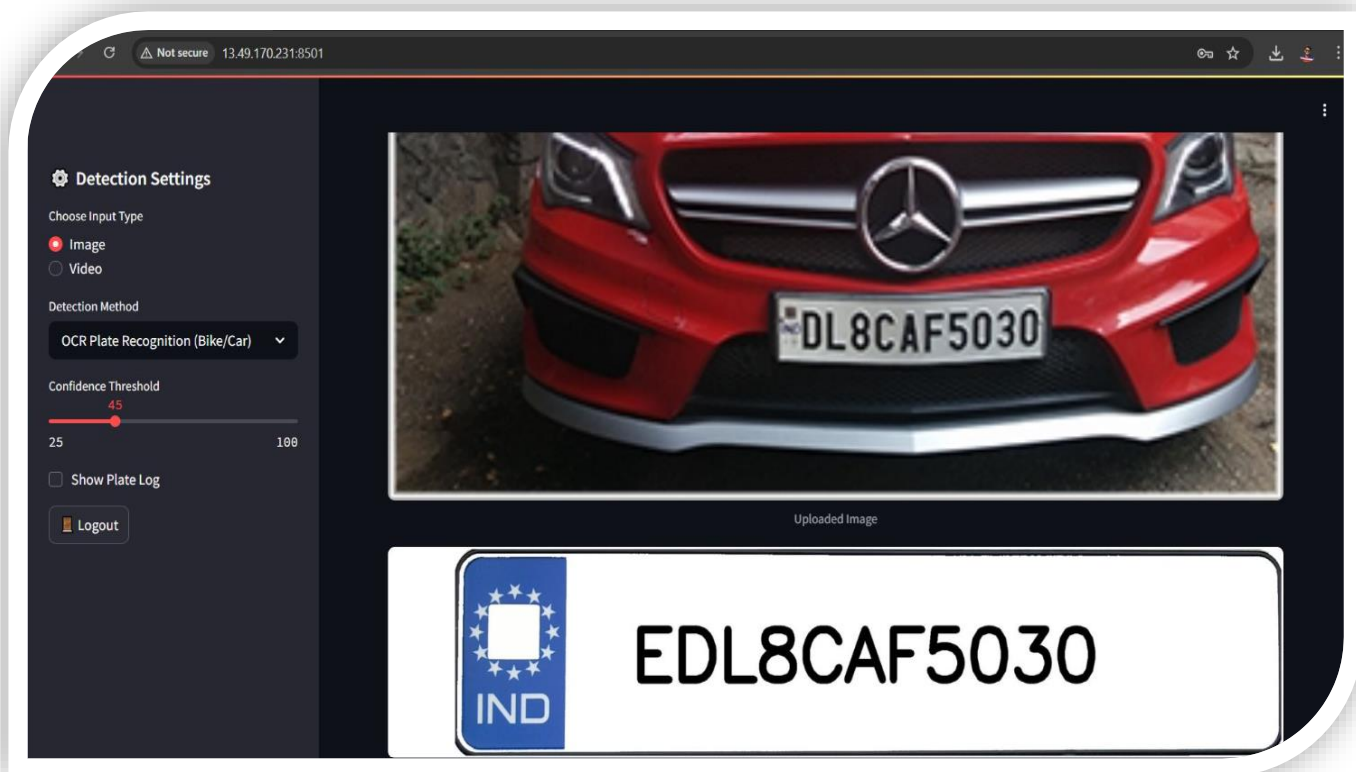
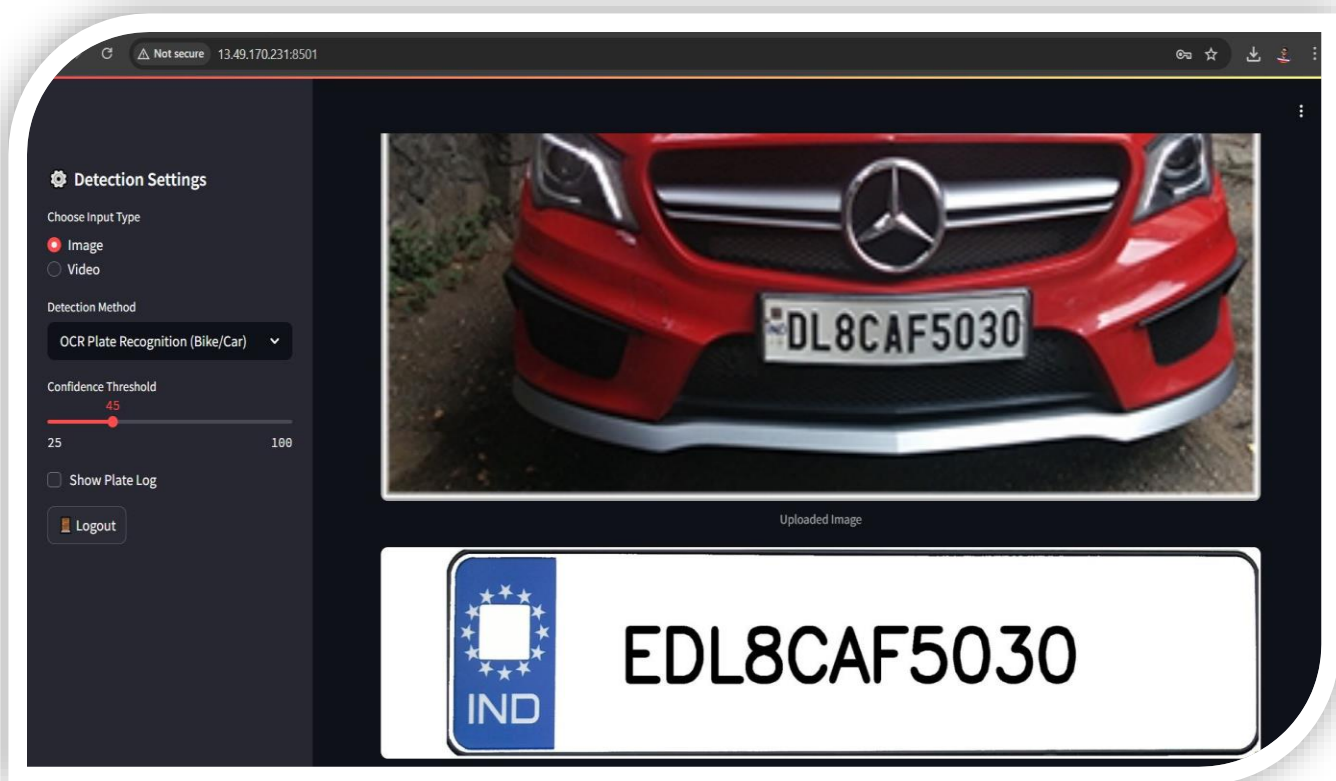
YOLOv8(Car) - Video

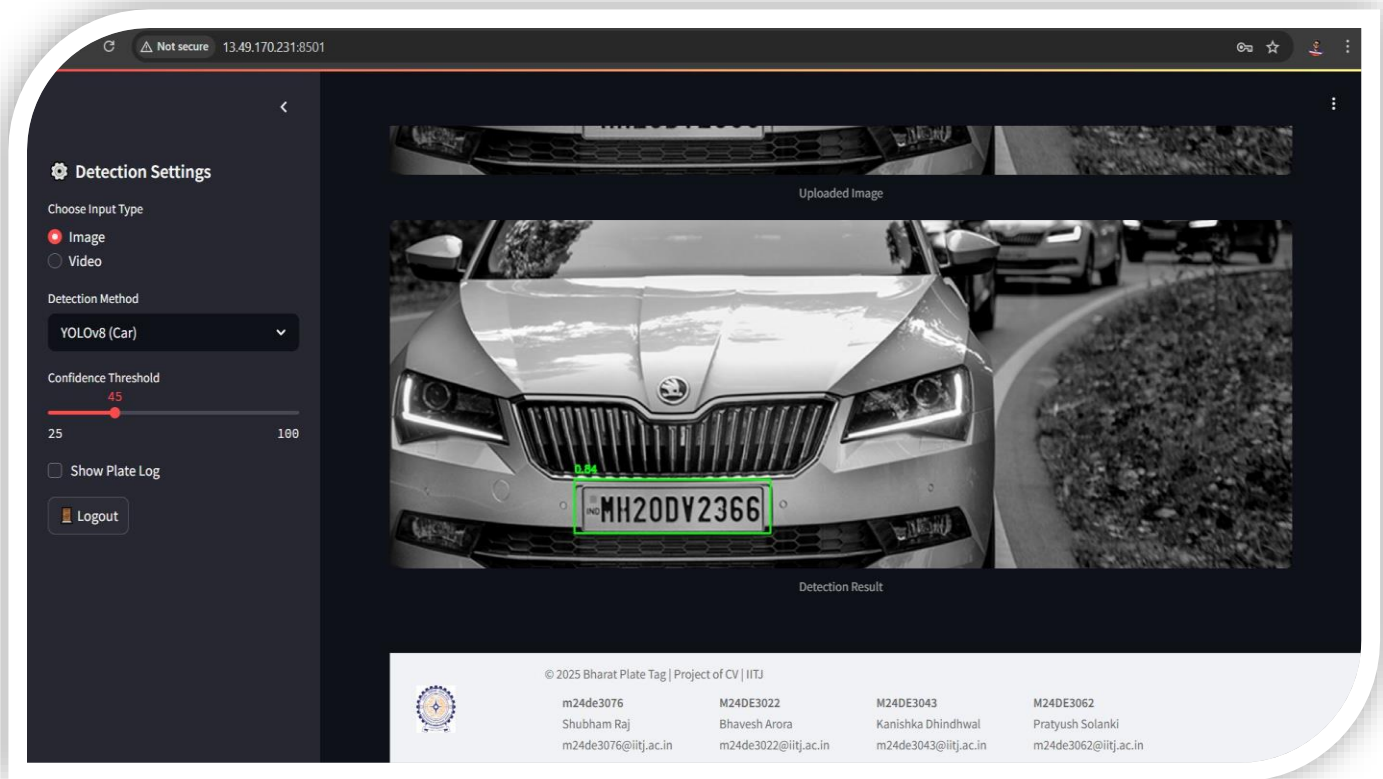


Traditional CV (Canny + Contours)



OCR Plate Recognition (Bike/Car)





9 OCR Plate Recognition: Benchmarking and Evaluation

9.1 Purpose of OCR Implementation

The OCR plate recognition module was implemented primarily for benchmarking and comparison purposes. Here's why:

Performance Benchmarking:

- Provides a baseline for comparison
- Helps identify strengths and weaknesses
- Validates the effectiveness of other methods

Evaluation Criteria:

- **Detection Speed:**
 - OCR is significantly slower (500ms)
 - Other methods are real-time capable
- **Resource Efficiency:**
 - OCR requires more computational resources
 - Other methods are more optimized

10 Future Scope of Work

10.1 CNN Classifier Enhancement

Architecture Improvements:

- Implement residual connections for deeper networks
- Add attention mechanisms for better feature extraction
- Explore transfer learning with pre-trained models

Data Augmentation:

- Implement more sophisticated augmentations
- Use mixup and cutmix techniques
- Generate synthetic plates for diverse scenarios

Training Optimization:

- Implement curriculum learning
- Use progressive resizing
- Apply learning rate warmup and cosine annealing

10.2 EDGE Morph Filter Enhancement

Advanced Edge Detection:

- Implement multi-scale edge detection
- Use anisotropic diffusion filtering
- Apply adaptive thresholding techniques

Morphological Operations:

- Implement adaptive structuring elements
- Use multi-scale morphological operations
- Combine with region growing techniques

Post-processing:

- Implement plate region verification
- Add geometric consistency checks
- Use confidence scoring for detections

The focus will be on making the system more robust and accurate, particularly in challenging scenarios like low light, occlusions, and varying plate orientations.