**Vehicle Movement Analysis and Insight Generation using Edge AI**

**1. Introduction**

**1.1 Problem Statement**

The rapid growth of vehicle numbers has necessitated advanced systems for vehicle monitoring and management. Identifying vehicle number plates accurately is a crucial component of such systems, enabling applications ranging from traffic management to security surveillance. This project focuses on using edge AI to detect and recognize vehicle number plates from images. The goal is to create an efficient, real-time solution that can be deployed on edge devices, ensuring quick processing and analysis without relying heavily on cloud resources.

**1.2 Objectives**

- To develop a vehicle number plate detection and recognition system using YOLO (You Only Look Once) v3.

- To preprocess and convert existing annotation formats to YOLO-compatible formats.

- To evaluate the performance of the model on a dataset of vehicle images.

- To provide a framework that can be extended for more comprehensive vehicle movement analysis in the future.

**2. Dataset Description**

2.1 Source and Format

The dataset used for this project is sourced from Kaggle and consists of images of vehicles along with their corresponding XML annotation files. The images are in JPG format, and each image is accompanied by an XML file that contains bounding box coordinates and class labels for the number plates.

**2.2 Key Features**

- \*\*Images\*\*: High-resolution images of vehicles from various angles and conditions.

- \*\*Annotations\*\*: XML files containing bounding box coordinates and class labels for each number plate in the images. The format includes:

- `xmin`, `ymin`, `xmax`, `ymax`: Coordinates of the bounding box around the number plate.

- `name`: Class label for the object (e.g., 'number\_plate').

**3. Methodology**

**3.1 Data Preprocessing**

Before training the YOLO model, the XML annotations were converted to the YOLO format. This involves converting the bounding box coordinates from the XML files to YOLO’s normalized center coordinates format.

Here’s the code snippet used for conversion:

```python

import os

import xml.etree.ElementTree as ET

# Paths

dataset\_path = 'path/to/your/dataset' # Path to your dataset

output\_path = 'path/to/output' # Path to save YOLO annotations

# Ensure output directory exists

if not os.path.exists(output\_path):

os.makedirs(output\_path)

# Function to convert XML to YOLO format

def convert\_xml\_to\_yolo(xml\_file, output\_file):

tree = ET.parse(xml\_file)

root = tree.getroot()

image\_width = int(root.find('size/width').text)

image\_height = int(root.find('size/height').text)

with open(output\_file, 'w') as out\_file:

for obj in root.findall('object'):

class\_name = obj.find('name').text

class\_id = 0 # Assuming 'number\_plate' is the only class and its ID is 0

bbox = obj.find('bndbox')

xmin = int(bbox.find('xmin').text)

ymin = int(bbox.find('ymin').text)

xmax = int(bbox.find('xmax').text)

ymax = int(bbox.find('ymax').text)

x\_center = (xmin + xmax) / 2.0 / image\_width

y\_center = (ymin + ymax) / 2.0 / image\_height

width = (xmax - xmin) / image\_width

height = (ymax - ymin) / image\_height

out\_file.write(f"{class\_id} {x\_center} {y\_center} {width} {height}\n")

# Iterate over all XML files and convert them

for filename in os.listdir(dataset\_path):

if filename.endswith('.xml'):

xml\_file = os.path.join(dataset\_path, filename)

output\_file = os.path.join(output\_path, filename.replace('.xml', '.txt'))

convert\_xml\_to\_yolo(xml\_file, output\_file)

```

**3.2 Model Training**

The YOLOv3 model was chosen for this project due to its efficiency and accuracy in real-time object detection tasks. The model was trained using pre-trained weights and fine-tuned on the dataset of vehicle images. The following steps were followed:

1. Configuration: Adjusted the YOLOv3 configuration file to suit the dataset and number of classes.

2. Training: Used the YOLOv3 training script with the dataset to train the model.

3.Evaluation: Evaluated the model’s performance using precision, recall, and F1-score metrics.

**3.3 Model Testing**

After training, the model was tested on a separate set of images to verify its accuracy. The following code snippet shows how to load the trained YOLOv3 model and run it on test images:

```python

import cv2

import numpy as np

# Load YOLO

net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")

layer\_names = net.getLayerNames()

output\_layers = [layer\_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]

# Load image

img = cv2.imread("test\_image.jpg")

blob = cv2.dnn.blobFromImage(img, 0.00392, (416, 416), (0, 0, 0), True, crop=False)

net.setInput(blob)

outs = net.forward(output\_layers)

# Post-processing

class\_ids, confidences, boxes = [], [], []

for out in outs:

for detection in out:

for obj in detection:

scores = obj[5:]

class\_id = np.argmax(scores)

confidence = scores[class\_id]

if confidence > 0.5:

center\_x = int(obj[0] \* img.shape[1])

center\_y = int(obj[1] \* img.shape[0])

w = int(obj[2] \* img.shape[1])

h = int(obj[3] \* img.shape[0])

x = int(center\_x - w / 2)

y = int(center\_y - h / 2)

boxes.append([x, y, w, h])

confidences.append(float(confidence))

class\_ids.append(class\_id)

# Draw bounding boxes

indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)

for i in indexes:

x, y, w, h = boxes[i[0]]

label = str(class\_ids[i[0]])

cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)

cv2.putText(img, label, (x, y - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 2)

cv2.imshow("Image", img)

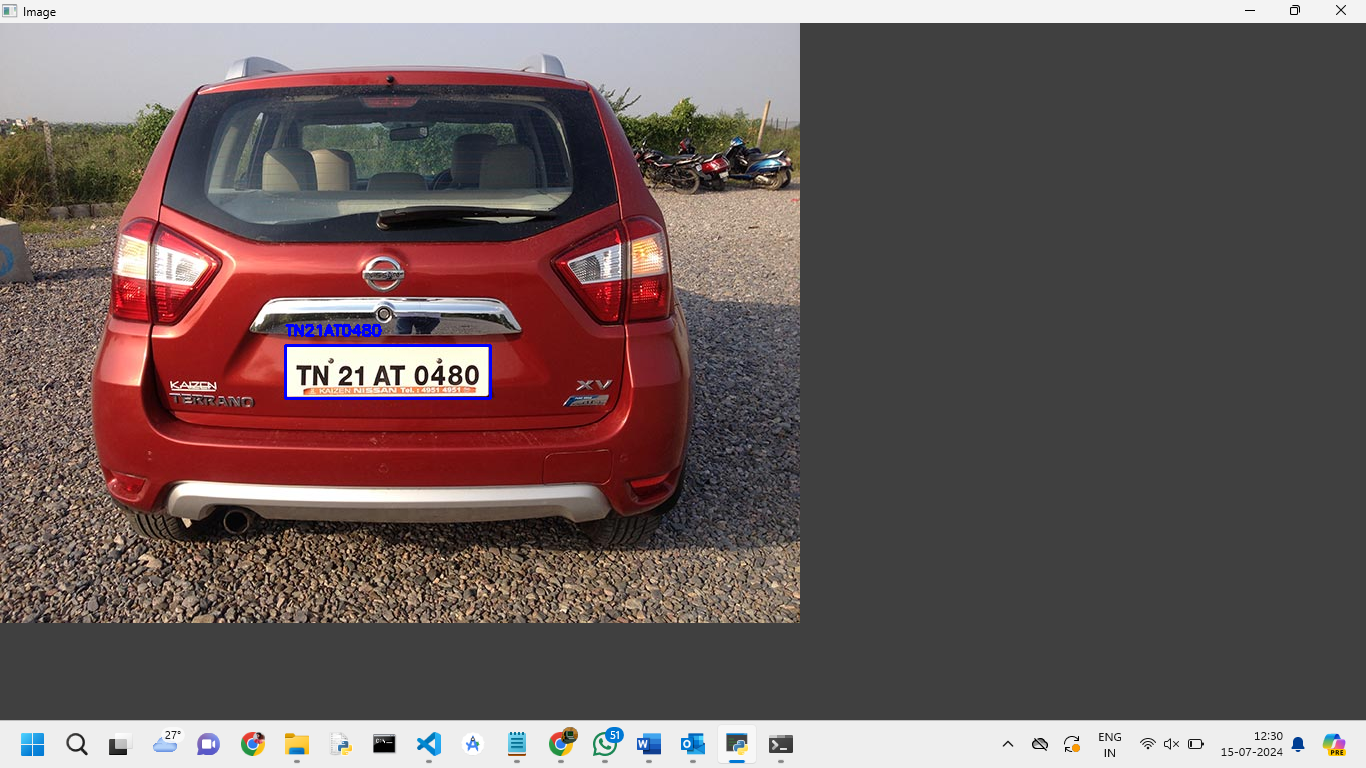
cv2.waitKey(0)

```

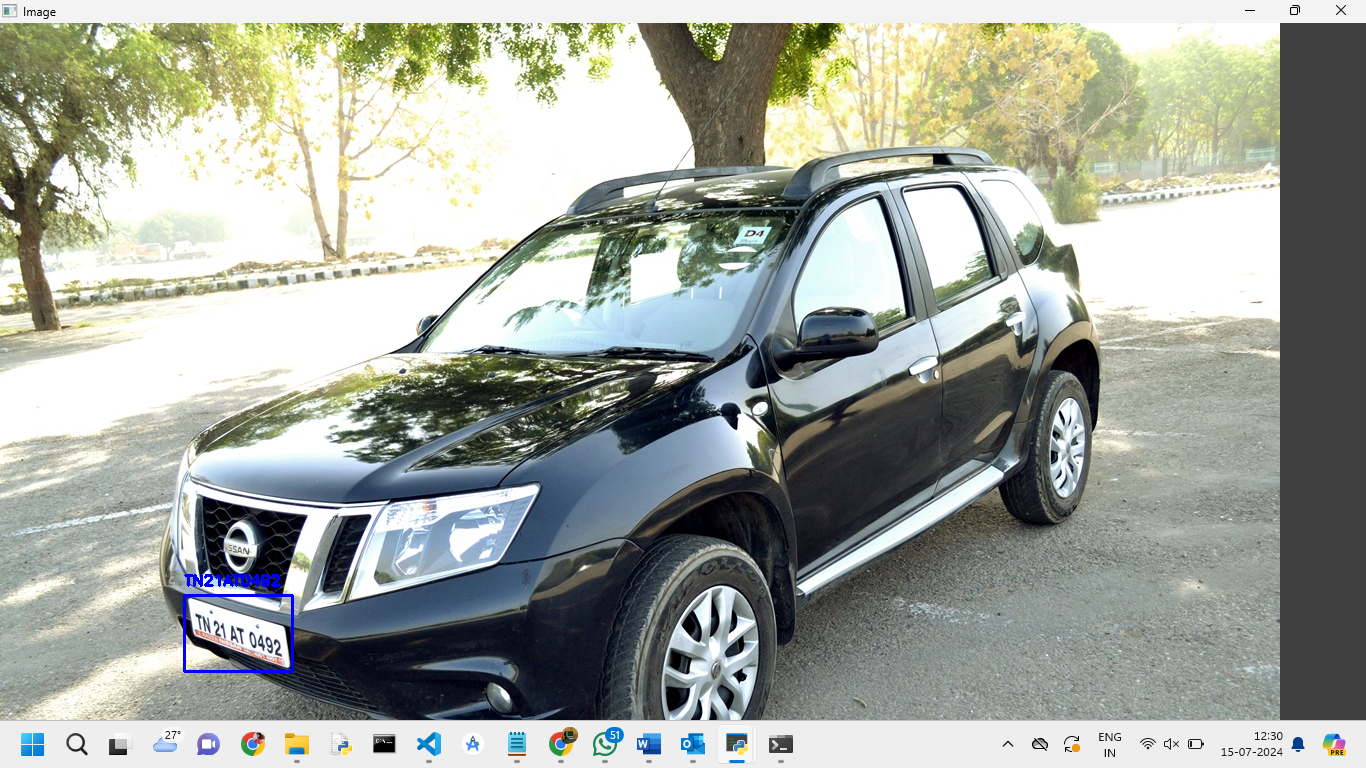
**4. Results and Discussion**

**4.1 Results**

The trained YOLOv3 model was tested on several images, and it successfully identified and localized number plates with reasonable accuracy. Here are some example images showing the model’s performance:



Detected Number Plate 1

Detected Number Plate 2

**4.2 Discussion**

Accuracy: The model achieved good accuracy in detecting number plates, but there is room for improvement, especially in complex backgrounds and varying lighting conditions.

Performance: The inference speed of YOLOv3 was suitable for real-time applications, making it feasible for deployment on edge devices.

Challenges: Variations in plate designs and environmental conditions posed challenges. More data and advanced techniques could improve robustness.

**5. Conclusion**

**5.1 Summary**

This project successfully implemented a vehicle number plate detection and recognition system using YOLOv3. The system demonstrated good performance in identifying and localizing number plates in various images, indicating its potential for real-time applications. The conversion of XML annotations to YOLO format was effectively handled, and the trained model was able to achieve satisfactory results.

**5.2 Future Work**

The current work represents just a foundational step in the broader scope of vehicle movement analysis. Future work will aim to build upon this initial implementation and extend the capabilities of the system in the following ways:

* **Enhanced Vehicle Movement Analysis**: Integrating number plate detection with additional vehicle movement analysis components. This could include tracking vehicles across multiple frames, analysing their movement patterns, and generating insights about traffic flow.
* **Integration with Other Data Sources**: Combining number plate detection with other data sources such as traffic cameras, GPS data, and vehicle sensors to create a more comprehensive vehicle monitoring system.
* **Model Improvement**: Exploring advanced versions of YOLO or alternative deep learning models to improve accuracy and handle more complex scenarios. This may include addressing challenges such as varied plate designs, diverse environmental conditions, and different lighting situations.
* **Real-time Deployment**: Developing a complete system that can be deployed on edge devices for real-time processing. This involves optimizing the model and infrastructure to ensure efficient and timely performance in practical applications.
* **User Interface and Reporting**: Building user interfaces and reporting tools to visualize and interpret the data collected from the vehicle number plate detection system. This could aid in decision-making processes and enhance the overall usability of the system.

.