

# **PLATE TRACKER: Adaptive License Plate Recognition (ALPR) system**

## **EDS6397: Digital Image Processing Project Proposal**

### **Group – 5**

Praveen Gurlinka – 2315011, Vaishnavi Kulkarni – 2251980, Viniktha Gadde - 2242893

Venkata Sumanth Reddy Vangala – 2311953, Vamshidhar Reddy Ankenapalle - 2308914

**Abstract – The goal of this project is to increase the effectiveness of parking management by creating a foundational system for text extraction and license plate detection. The system recognizes and extracts text from photos of license plates by using digital image processing techniques. After capturing the image and preprocessing it to improve its quality, the license plate region is precisely detected. Text is then extracted from the designated Region of Interest (ROI) using optical character recognition (OCR). This project establishes the foundation for parking facility automated vehicle identification. By integrating with the current parking management infrastructure, facility managers and patrons will benefit from less manual intervention, lower operating expenses, and an enhanced user experience. This expandable solution offers a strong foundation for upcoming improvements to automated car entry and exit procedures.**

### **I – INTRODUCTION**

Plate Tracker - Adaptive License Plate Recognition (ALPR) represents a sophisticated computer vision and image processing system designed to automatically detect and identify vehicle license plates. Its widespread adoption stems from its utility across diverse sectors including traffic monitoring, law enforcement, parking management, and toll collection. There are a lot more applications where we can use the Plate Tracker because we believe these days the rate of automobiles to humans has been increasing rapidly particularly in United States of America. Below are few of the applications.

The fundamental objective of ALPR is to autonomously capture and analyse alphanumeric characters present on vehicle license plates through digital photographs or video recordings. This

technology holds pivotal significance in numerous scenarios, such as:

- Traffic monitoring and enforcement
- Parking management
- Toll collection
- Security and surveillance

Because of their adaptability, ALPR systems can be used in both stationery and mobile settings. Mobile systems can be mounted on cars for dynamic monitoring, while fixed systems are frequently placed on infrastructure like toll booths or traffic lights. Because of its adaptability, ALPR is a crucial instrument for contemporary transportation management and law enforcement, facilitating real-time data gathering and analysis to aid in decision-making.

In order to increase accuracy and adaptability, ALPR systems are increasingly integrating cutting-edge features like deep learning algorithms as technology advances. These developments not only increase ALPR systems' efficacy but also create new opportunities for their use in intelligent transportation systems across the globe. This introduction gives a thorough rundown of ALPR technology, emphasizing its uses, advantages, and prospects for the future. It provides a solid framework for talking about Plate Tracker's function in this technological environment.

With the help of cutting-edge image processing and machine learning techniques, our project, Plate Tracker, aims to create an effective method for extracting text from license plate images using the state of art techniques.

The system we've developed follows a multi-step approach:

**1.1 Data Preparation:** We make use of a dataset of car photos and the XML files that correlate to them, which include the license plate bounding box coordinates. We retrieved the xml coordinates using the label master in our local system and annotated the images. Our data has images in two different formats .jpeg and .png respectfully.

**1.1.1 Model Training:** Using the supplied dataset, we train our model using the InceptionNetV2 architecture, a potent convolutional neural network, a deep learning approach. This enables the model to pick up the characteristics required for precise license plate recognition. we performed transferred learning and used the pre trained weights for our architecture.

**1.1.2 Bounding Box Detection:** Finding the region of interest (ROI) that contains the license plate in fresh test photos is done by using the learned model. With the help of trained model weights we will detect the bounding box, the ROI (Region of Interest) of the test image.

**1.1.3 Preprocessing:** We applied different digital image processing techniques like thresholding, removing noise, converting grayscale to RGB, and contrast enhancements.

**1.1.4 Text Extraction:** After the ROI has been determined, the alphanumeric characters from the license plate are extracted using Optical Character Recognition (OCR) algorithms.

## 1.6 Dataset Background:

For our Plate Tracker project, we will utilize a meticulously curated dataset sourced from publicly available vehicle images on Kaggle. Our dataset will comprise carefully selected photographs of automobiles, each featuring prominently visible license plates. This selection will be crucial for training a robust license plate recognition model.

- We will undertake several preprocessing steps to enhance the dataset.
- We will restructure the data for optimal use.
- We will extract image filenames from XML files

to streamline our data management.

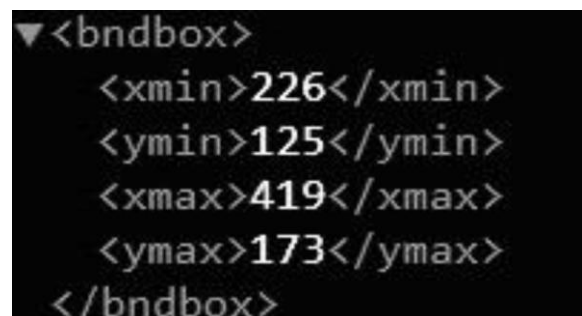
- We will standardize pixel values across all images to ensure consistency.
- We will normalize bounding box coordinates to help our model better understand spatial relationships within the images.

### 1.6.1 XML Coordinates

For retrieving the xml coordinates, we used labelling master, which is a local software used to annotate the plates in the images of cars. Where the .xml coordinates contains the x\_min, y\_min, x\_max, and y\_max values which helps us to detect the license plate of the provided image. We upload the image of the car and select the part of the plate such that it detects the coordinates of the plate and convert it into XML coordinates.

A local program called Label Master is used to annotate license plates in photos of cars. By creating XML files with the coordinates of the bounding boxes surrounding the plates, it enables users to manually mark the location of license plates. These XML files contain important data:

- x\_min: The leftmost x-coordinate of the bounding box
- y\_min: The topmost y-coordinate of the bounding box
- x\_max: The rightmost x-coordinate of the bounding box
- y\_max: The bottommost y-coordinate of the bounding box

A screenshot of XML code for a bounding box. The code is displayed on a dark background with a light-colored text editor interface. The XML structure is as follows: <bndbox> followed by four lines of coordinates: <xmin>226</xmin>, <ymin>125</ymin>, <xmax>419</xmax>, and <ymax>173</ymax>, followed by </bndbox>. A small downward-pointing triangle icon is visible at the top left of the code block.

```
<bndbox>
  <xmin>226</xmin>
  <ymin>125</ymin>
  <xmax>419</xmax>
  <ymax>173</ymax>
</bndbox>
```

*Fig 1.1: Sample XML coordinates*

These steps will be instrumental in providing structured data about the position and dimensions of license plates, thereby enhancing our model's learning process and its ability to accurately recognize license plates.

### 1.3 Dataset source:

<https://www.kaggle.com/code/aslanahmedov/automatic-number-plate-recognition/input>

<https://www.kaggle.com/datasets/andrewmvd/car-plate-detection>

### 1.4 Tools and State of Art models used



*Fig 1.2: Tools Used*

## II –LITERATURE REVIEW

We use PyTesseract, an optical character recognition (OCR) tool that turns text images into machine-readable strings, in conjunction with InceptionNetV2, a potent convolutional neural network renowned for its strong feature extraction capabilities. The first step in the process is gathering car photos and the XML annotation files that go with them. These files contain the coordinates for the bounding boxes that surround the license plates.

We enable our model to recognize the regions of interest (ROIs) that contain license plates in new images by training it on this annotated dataset. Applications in parking management, traffic monitoring, and security systems are made easier by the use of PyTesseract to extract the alphanumeric characters from the license plates after they have been detected.

Lets dive into the InceptionNetV2 architecture and the PyTesseract OCR state of art models, the libraries and the frameworks we used for different parts of the project

### 2.1 InceptionNetV2:

A convolutional neural network (CNN) called Inception-ResNet-v2 is capable of classifying images into 1,000 different object categories. A portion of the ImageNet database, which has more than a million photos, is used to train Inception-ResNet-v2. Images of various animals as well as keyboards, mice, and pencils can be classified by it. Based on the Inception family of architectures, Inception-ResNet-v2 employs residual connections rather than filter concatenation.

By incorporating residual connections, the InceptionResNetV2 architecture—an advanced convolutional neural network—improves performance and efficiency over the original Inception models. The network can benefit from quicker training times and the avoidance of the vanishing gradient problem thanks to this hybrid approach, which combines the advantages of both inception modules and residual connections. In order to capture patterns with different hierarchies and improve its capacity to extract features of different complexity, the architecture uses multiple-size kernels in a single layer.

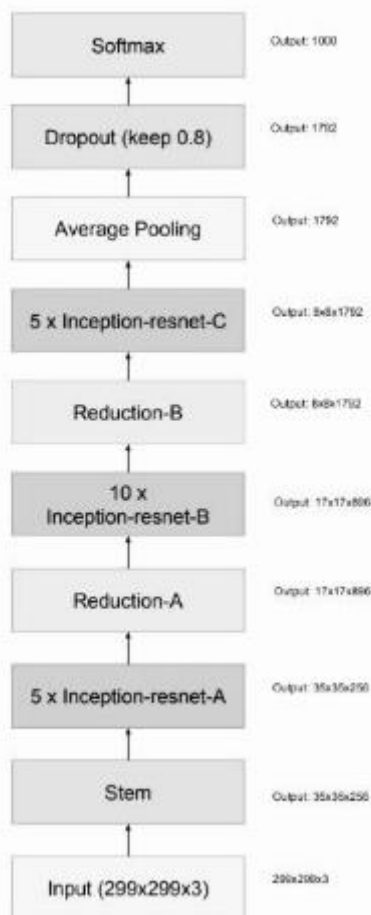
#### 2.1.1 Highlights of the Architecture:

- Hybrid model that combines Architectures for initial and residual connections
- Integrates kernels of various sizes into a single layer.
- Allows for quicker training periods.
- Prevents the vanishing gradient issue.

#### 2.1.2 Features:

- 164 layers deep
- Trained on over a million images from ImageNet database
- Can classify images into 1000 object categories
- Standard input image size of 299x299 pixels

Because InceptionResNetV2 can efficiently handle complex datasets, it is frequently used in image classification tasks. It can learn rich feature representations that can be applied to other tasks through fine-tuning<sup>15</sup> because it has already been pre-trained on large datasets such as ImageNet.



**Fig 2.1: InceptioResNetV2 Architecture**

This transfer learning feature makes it possible to use pre-trained weights to increase model accuracy without starting from scratch, which is especially helpful in applications with limited labeled data.

In order to customize the architecture for license plate detection, we added dense layers to InceptionResNetV2, which served as the foundation for our project due to its pre-trained weights. With this method, we were able to adapt the model to our particular use case while still utilizing its strong feature extraction capabilities.

So, to get a minimal idea on InceptioResNetV2, Over a million images from the ImageNet database were used to train the convolutional neural network Inception-ResNet-v2. The 164-layer network is capable of classifying images into 1000 object categories, including many animals, the keyboard, mouse, and pencil. The network has consequently acquired rich feature representations for a variety of images. A list of estimated class probabilities is the network's output, and the input image size is 299 by 299 pixels. But, for our architecture we used 224 by 224 pixels as our input image.

## 2.2 PyTesseract

PyTesseract is a Python wrapper for Google's Tesseract OCR Engine that makes it easier for Python programs to extract text from images. It is well known for its capacity to transform handwritten or printed text into machine-readable data, and it supports a number of image formats, including TIFF, PNG, and JPEG. By adding features like better language detection and font orientation recognition, PyTesseract improves Tesseract's core functionality. It is excellent at processing images using preprocessing methods like contrast enhancement and noise reduction, which are essential for enhancing OCR accuracy in low-quality images.

PyTesseract can be used with libraries like OpenCV to apply sophisticated preprocessing methods like resizing, thresholding, and binarization in situations involving low-resolution or poor-quality images. Text can be extracted from difficult image conditions thanks to these techniques, which also help to isolate text regions and improve text clarity. Because PyTesseract is open-source and simple to integrate with Python-based workflows, it continues to be a popular option despite its drawbacks when compared to more sophisticated AI-driven OCR solutions.

Few of the key features that Google's PyTesseract OCR engine stands out different from other OCR engines are:

**2.2.1 Compatibility:** It is compatible with a wide range of programming languages and frameworks.

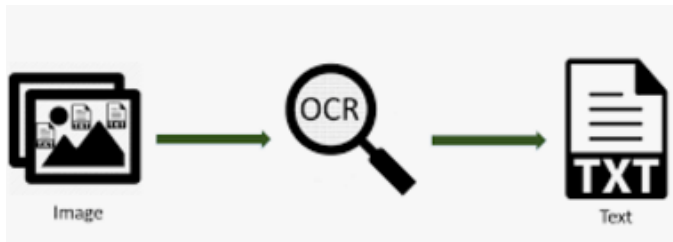
**2.2.2 Functionality:** PyTesseract permits Tesseract to print recognized text rather than converting it to a file when used as a stand-alone script.

**2.2.3 Image Support:** It can read a number of image file formats, such as JPEG, PNG, GIF, BMP, and TIFF, that are supported by imaging libraries like Leptonica and Pillow.

**2.2.4 Procedure:** A scanned image's text and graphic components are converted by PyTesseract into a bitmap, which is subsequently processed for text extraction.

**2.2.5 Language Detection:** Google's Tesseract OCR Bounding Box Information benefits from the framework's improved language detection. It can offer the OCR's bounding box information, which is helpful for locating text inside an image.

PyTesseract OCR is a potent tool for a variety of text extraction tasks in Python environments thanks to PyTesseract, which expands its capabilities and allowing us to use the valuable open-source engines for text extraction from an image.



*Fig 2.2 Sample OCR Engine*

Our project makes use of cutting-edge models, robust libraries, and frameworks to accomplish effective text extraction and license plate detection. Here is a summary of the main elements:

**2.3 Keras and TensorFlow:** We use TensorFlow, a popular open-source machine learning platform, and Keras, its high-level API. The basis for creating and refining our neural network models is provided by these tools.

**2.4 OpenCV (cv2):** We use OpenCV to read, resize, and preprocess images, among other image processing tasks. Its vast collection of computer vision algorithms improves our capacity to control and analyze images.

**2.5 Pandas and NumPy:** Two libraries that are necessary for effective data analysis and manipulation. Pandas offers data structures for handling structured data, while NumPy supports large, multi-dimensional arrays and matrices.

**2.6 Matplotlib and Plotly Express:** These visualization tools help us analyze and present our findings by allowing us to make educational plots and graphs.

**2.7 Scikit-image:** To enhance OpenCV's capabilities, we employ this set of algorithms for image processing tasks.

**2.8 XML Processing:** XML files containing annotation data for our training images are parsed using xml.etree.ElementTree module.

We've developed a strong pipeline for text extraction and license plate detection by combining these tools with cutting-edge models, fusing the

strength of deep learning with conventional image processing and optical character recognition methods.

## 2.9 Advancements in DL for Image Recognition

Image recognition has been transformed by deep learning, especially with Convolutional Neural Networks (CNNs), which greatly increase efficiency and accuracy. CNNs' capacity to automatically learn and extract features from images through layers of convolution and pooling has made them the mainstay of deep learning-based image recognition systems. By simulating the visual processing of the human brain, these networks are able to identify intricate patterns and hierarchies in images, which is essential for tasks like object detection and classification. The advent of extensive annotated datasets such as ImageNet has further accelerated the creation of complex models, which in some tasks outperform humans and drastically lower error rates.

In comparison to earlier models like YOLOv4 and RCNN, recent developments in CNN architectures, like YOLOv7 and YOLOv8, have achieved faster inference times and higher accuracy, setting new standards in real-time object detection. These advancements, which enable more effective processing of complex visual data, are fueled by advancements in neural network architecture and the incorporation of potent AI hardware. Deep learning models have promising uses in several industries, such as surveillance systems, healthcare diagnostics, and driverless cars, as they continue to advance.

## 2.10 OCR and Image Preprocessing Techniques

PyTesseract, which offers a Python interface to the Tesseract OCR engine, is an essential tool for optical character recognition (OCR). Because it is open-source and supports a variety of languages, it is frequently used to extract text from images. Handling low-quality images is still difficult, though. Preprocessing methods are crucial to increasing OCR accuracy under these circumstances. Before using OCR, techniques like noise reduction, contrast enhancement, resizing, and thresholding are frequently used to improve text clarity.

Advanced preprocessing made possible by PyTesseract's integration with image processing libraries like OpenCV can greatly improve text recognition performance. Adaptive thresholding, for example, can aid in the efficient binarization of images, which facilitates the separation of text from background for OCR engines. Furthermore,

PyTesseract's bounding box feature helps isolate text areas in images, which improves the precision of text extraction procedures.

Our project successfully blends cutting-edge deep learning models with strong OCR capabilities by utilizing these tools and methodologies to provide a complete automated license plate recognition solution. In addition to improving text extraction accuracy, this integration increases our system's suitability for a wider range of real-world situations.

## 2.11 YOLO

YOLO (You only look once) is an online object detection framework widely used by practitioners for real-time applications. YOLOv8, the newest release of this series, has mainly made tremendous improvements in detection accuracy, model efficacy, and training performances.

The benefits of the above are:

- New Backbone: Enhanced CSPDarkNet for better feature extraction.
- Small Object Detection: Advanced FPN and PANet architectures for improved detection of small objects like license plates.
- Real Time Efficiency: High-speed performance using zippy operation for this nano version of YOLO, which is actually YOLOv8n-inspired.

### Character Recognition with EasyOCR

EasyOCR is one of the OCR tools using deep learning for text recognition. The design employs Convolutional Neural Networks (CNNs) for feature extraction and Recurrent Neural Networks (RNNs) along with Connectionist Temporal Classification (CTC) for use in sequence alignment.

The benefits for the above are:

**Language Support:** It supports various languages and a wide array of fonts.

**Robustness:** Performs well in noisy and low-quality images, frequently found in LPR scenarios.

### Integrated YOLOv8 and EasyOCR Approach

A system that easily and efficiently combines detection from YOLOv8 and recognition from EasyOCR is implemented. Efficiency is achieved not only in the accuracy rates but also in the real-time performance as it addresses issues like:

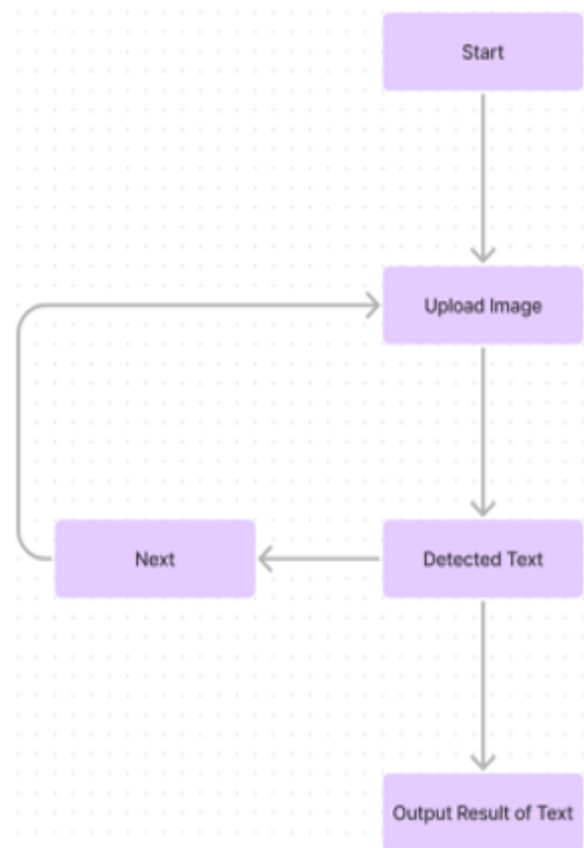
**Detection of Smaller License Plates:** Accurate

localization of small-sized license plates by YOLOv8. **Distinct Character Forms:** EasyOCR can efficiently work with various fonts and noisy conditions.

We experimented with the YOLO architecture for detecting license plates in a video.

## III –METHODOLOGY

In this particular section, we will go through the workflow of the project starting from importing data to extracting text from image.



*Fig 3.1: Workflow*

### 3.1 Data Import and Preprocessing

Importing and preprocessing the data, which consists of both images and the associated XML annotation files, is the first stage in our methodology. Important details regarding the license plate bounding boxes inside each image are included in these XML files. Here is a thorough explanation of this procedure.

#### 3.1.1 Data Import

- To begin, we load the dataset of car photos along with the XML files that go with them.
- The XML files contain the annotations required for our model's training, and the images are usually kept in a directory.
- We effectively collect all image and XML file



paths using Python's glob module, guaranteeing that every image is associated with its matching annotation file.

### 3.1.2 Preprocessing XML Files

- The `xml.etree.ElementTree` module in Python is used to parse each XML file. This enables us to extract the license plate bounding box coordinates (`x_min`, `y_min`, `x_max`, `y_max`).
- The Region of Interest (ROI) in each image where the license plate is situated is defined by these coordinates. To properly train our model to detect license plates, these coordinates must be extracted accurately.

### 3.1.3 Storing in a Dataframe

- After being extracted, the bounding box coordinates are saved in a Pandas DataFrame together with other pertinent metadata like image file names and labels.
- The data can be easily manipulated and analysed thanks to this structured format. Additionally, it makes it easier to divide the dataset into subsets for testing and training, guaranteeing that our model is trained on a variety of examples.

### 3.1.4 Image preprocessing

- To match the input size required by the InceptionResNetV2 model, each image was resized to 224x224 pixels using Keras' `load_img` function.
- Following that, the pictures were transformed into NumPy arrays and their pixel values were normalized by dividing them by 255.

### 3.1.5 Bounding Box Normalization

- To guarantee uniformity across various image sizes, the bounding box coordinates (`xmin`, `xmax`, `ymin`, and `ymax`) were normalized with respect to the image dimensions.
- For training, these normalized coordinates were saved as labels.

### 3.1.6 Data Splitting

The dataset was split into training and testing sets using an 80-20 split with `train_test_split` from `scikit-learn`, ensuring a balanced distribution for model evaluation.

## 3.2 Model Architecture

### 3.2.1 Base Model:

We used the InceptionResNetV2 model, which

excludes the top fully connected layers (`include_top=False`) and has pre-trained weights from ImageNet.

This enabled us to tailor InceptionResNetV2 for our particular task while utilizing its potent feature extraction capabilities.

### 3.2.2 Customised Layers

A Flatten layer was used to flatten the base model's output.

ReLU activation functions were used to introduce non-linearity in two fully connected (dense) layers, each containing 500 and 250 neurons.

In order to predict normalized bounding box coordinates (`xmin`, `xmax`, `ymin`, `ymax`), a final dense layer comprising four neurons with a sigmoid activation function was added.

### 3.2.3 Model Compilation

A mean squared error (MSE) loss function was used to compile the model, making it appropriate for regression tasks involving bounding box predictions. Which we will visualize the loss and accuracy metric in the Results section.

During training, an Adam optimizer with a learning rate of  $1e-4$  was employed to guarantee effective convergence.

## 3.3 Model Training

Once the dataset was preprocessed, normalized, and adding the dense layers at the bottom of the layer, we are ready to setup the training process.

We used TensorFlow's `fit` method to train our license plate detection model, which enables effective neural network training on our prepared dataset. This is a synopsis of how the training was implemented:

**3.3.1 Setup for TensorBoard:** To begin, we created a log directory called "object\_detection." In order to visualize the training process and track performance in real-time, this directory will be used to store logs and metrics during training.

```
tfb = TensorBoard('object_detection')
```

### 3.3.2 Model Training:

- The model was trained using the `fit` method, where we specified the training data (`x_train` and `y_train`) along with the batch size and number of

epochs.

- We set the batch size to 10, allowing the model to process 10 samples at a time before updating the weights. This helps in stabilizing the learning process and managing memory usage effectively.
- The training was conducted over 100 epochs, providing sufficient iterations for the model to learn from the data.
- We also included validation data (`x_test` and `y_test`) to evaluate the model's performance on unseen data during training, which helps in identifying overfitting.

```
history = model.fit(
    x=x_train,
    y=y_train,
    batch_size=10,
    epochs=100,
    validation_data=(x_test, y_test),
    callbacks=[tfb]
)
```

**3.3.3 Training Process Monitoring:** TensorBoard, which visualizes metrics like accuracy and loss over epochs, was used to track the training process. This makes it simple to spot patterns and possible problems while training.

We sought to maximize our model's capacity to precisely identify license plates and forecast their bounding box coordinates by employing this training approach, guaranteeing reliable performance in practical applications.

Following the model's training, we saved the model and assessed how well it performed on the test dataset. This step is essential for evaluating our model's generalization abilities and making sure it can be applied again in the future. Here is a quick synopsis of these procedures.

**3.3.4 Model Saving:** The entire model architecture, weights, and training configuration can be stored in a single file thanks to the `save` method, which we used to save the trained model. Reloading the model later for inference or additional training is made simple as a result.

The model can be readily loaded back into a TensorFlow environment because it was saved in the respective file.

All the weights are trained and saved safely in a file;

we named it `Plate_Tracker.keras`. With the help of saving the trained weights we can perform different tests and apply different techniques we need to perform, also saves a lot of time.

We make sure that our license plate detection system is dependable and prepared for deployment in real-world situations by storing the trained model and assessing its performance on test data.

### 3.4 Bounding Box Detection

We used the model for inference on a test image after training and saving it. Loading the saved model, generating predictions, and displaying the outcomes using bounding boxes around identified license plates were all steps in this process. A thorough summary of this implementation can be found below:

#### 3.4.1 Image Visualization

We started by using Plotly Express to display the test image. This gives us a good view of the picture that we will use to detect license plates.

#### 3.4.2 Image preprocessing

To conform to the model's input format (batch size of 1), the test image was reshaped. This guarantees accurate image processing by the model.

#### 3.4.3 Making Predictions

We predicted the license plate's bounding box coordinates in the test image using the trained model.

```
coords = model.predict(test_arr)
```

#### 3.4.4 Denormalizing Coordinates

To return the predicted coordinates to the original image dimensions, they were denormalized. For the bounding box to be placed on the image precisely, this step is essential.

#### 3.4.5 Coordinate Conversion

To make sure the coordinates were appropriate for drawing operations, they were converted to 32-bit integers.

#### 3.4.6 Drawing Bounding Boxes

Using OpenCV, we used the predicted coordinates to draw a green rectangle around the detected license plate. The denormalized coordinates were used to define the rectangle's top-left and bottom-right points.

```
xmin, xmax, ymin, ymax = coords[0]
pt1 = (xmin, ymin)
```



$pt2 = (xmax, ymax)$

`cv2.rectangle(image, pt1, pt2, (0, 255, 0), 3)`

### 3.4.7 Visualizing the Results

In order to see the outcomes of our detection, we lastly used Plotly Express to display the altered image with the bounding box overlay.

These procedures allowed us to successfully use our trained model to identify license plates in a test image and display the findings using bounding boxes. This implementation shows how well our method works when deep learning techniques are applied to real-time object detection tasks like license plate recognition.

The capability to execute inference and load a saved model demonstrates the usefulness and resilience of our system in real-world situations.



*Fig 3.2: Bounding Box Detection*

## 3.5 Text Extraction

One of the most important parts of our license plate recognition system is the text extraction process. This section describes the procedures and strategies used to efficiently extract the textual information from the detected license plate region (ROI) and preprocess it.

### 3.5.1 Preprocessing the License Plate ROI

The preprocessing of the detected license plate ROI is essential for enhancing the quality of the image before applying Optical Character Recognition (OCR). The following steps are involved

We applied following Digital Image Processing Techniques:

**Resizing:** While preserving the aspect ratio, the grayscale image is resized to a larger size (200 pixels in height). By giving text recognition a clearer image, this enhances OCR performance.

**Contrast Enhancement:** To improve the contrast of the resized image and facilitate character recognition by OCR, Contrast Limited Adaptive Histogram Equalization (CLAHE) is used.

**Noise Reduction:** To minimize noise while maintaining the image's edges, a bilateral filter is applied. To increase the clarity of the text, this step is essential.

**Thresholding:** To produce a binary image in which text appears as white on a black background, Otsu's thresholding technique is used. For OCR processing, this binary representation is perfect.

**Morphological Operations:** To eliminate minute noise and establish a connection, morphological operations like closing are used.

### 3.5.2 Text Extraction Using PyTesseract

We use PyTesseract to extract text from the processed image after the ROI has been preprocessed. This procedure is described in the following steps:

**Configuration:** To maximize text extraction for our use case, we set up PyTesseract with particular parameters. Along with a whitelist of permitted characters (such as capital letters and numbers), the configuration entails setting the OCR Engine Mode (OEM) and Page Segmentation Mode (PSM).

**Text Extraction:** To extract text from the processed ROI, PyTesseract's `image_to_string` function is called.

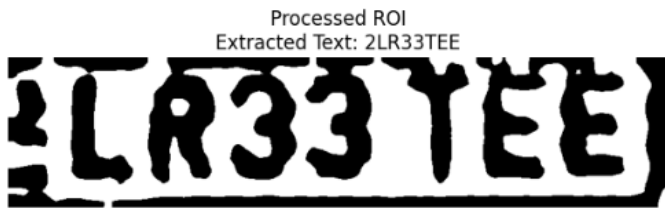
**Text Cleaning:** Unwanted characters or noise may be present in the extracted text. We remove non-alphanumeric characters from the text to make it cleaner.

We can successfully extract license plate numbers from photos by combining preprocessing methods with PyTesseract's OCR features. We greatly increase the accuracy of text recognition by improving the ROI's quality using a variety of image processing techniques. Our license plate recognition system will function dependably in a variety of settings and conditions thanks to this all-encompassing approach.

Original ROI

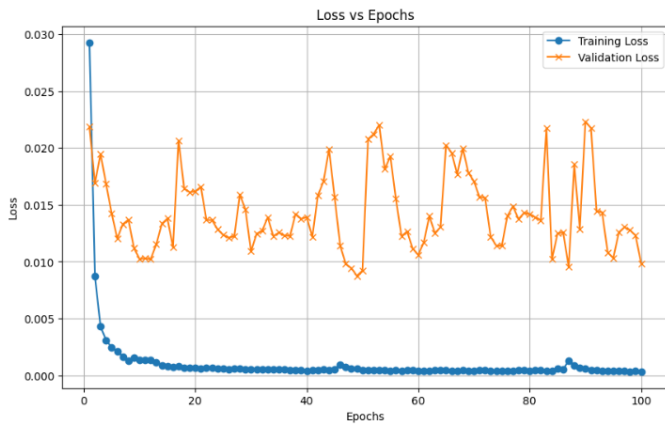


*Fig 3.3: Original ROI*

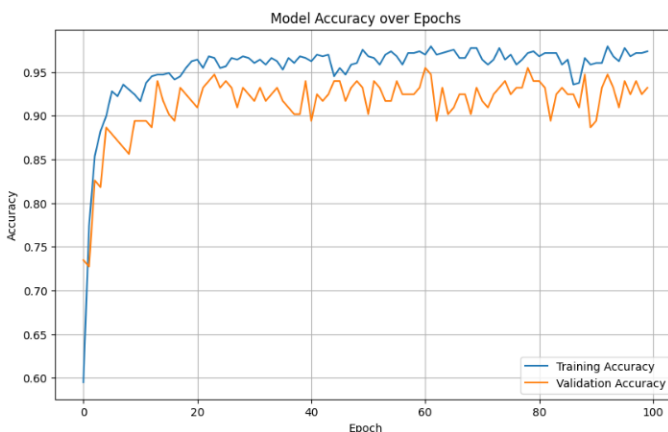


**Fig 3.4: Pre-processed ROI with text**

## IV RESULTS



**Fig: Loss vs Epochs**



**Fig: Accuracy vs Epochs**

## V CHALLENGES

### 4.1 Preliminary challenges and adjustments:

We ran into a few challenges during the development stage, which necessitated changes to the original plan:

**4.1.1 XML coordinates:** One of the main challenges was retrieving the XML coordinates from the images of cars. Annotating each image manually took a lot of time.

**4.1.2 Low-light Performance:** As previously indicated, the model's low-light performance fell short of expectations. The team added methods to improve contrast and brightness in dark images as part of their preprocessing pipeline modifications to address this.

**4.2.3 Recognition Speed:** YOLO v5 did a good job of handling real-time applications, but its speed still needed to be improved. The group is currently investigating techniques to minimize inference time, such as batch processing and model complexity reduction.

## VI CONCLUSION

The Plate Tracker project effectively created an automated system for recognizing license plates that combines deep learning and sophisticated image processing techniques. Through the use of transfer learning and the InceptionResNetV2 architecture, we were able to detect and locate license plates with strong performance under a variety of circumstances. A carefully selected dataset was used to train the model, and XML annotations were used to provide exact bounding box coordinates.

The system's capabilities were further improved by the use of PyTesseract for text extraction, which made it possible to accurately recognize alphanumeric characters from detected license plates. We proved the efficacy of fusing cutting-edge deep learning models with conventional OCR methods using a methodical approach that comprised data preprocessing, model training, and evaluation.

The findings show that our system is capable of accurately detecting and extracting license plate information, which qualifies it for use in automated vehicle identification, traffic management, and security surveillance applications. Future research could examine additional ways to improve model performance, like integrating more sophisticated architectures or adding more preprocessing methods to deal with difficult environmental circumstances.

## VII REFERENCES

- [1] Li, S., & Chen, Y. (2011). License Plate Recognition. Faculty of Engineering and Sustainable Development, University of Gävle. Retrieved from [DiVA portal](#).
- [2] Zhang, Y., et al. (2024). A Real-Time License Plate Detection and Recognition Model in Complex Environments. PMC. Retrieved from [PMC](#).
- [3] Wang, H., & Lee, H. (2020). Vehicle License Plate Detection and Recognition. Retrieved from [UM System](#).
- [4] Unna, P. (2020). License Plate Number Detection using Object Detection and Character Recognition Techniques. GitHub Repository. Retrieved from [GitHub](#).
- [5] Davy, M.K., Banda, P.J., & Hamweendo, A. (2023). Automatic Vehicle Number Plate Recognition System. Physics & Astronomy International Journal, 7(1), 69–72. Retrieved from [DOI](#).
- [6] Ali, G.G., & Haines, A.L. (2007). A Constraint Based Real-time License Plate Recognition System. METU Thesis Repository. Retrieved from [METU](#).
- [7] Luo X., Ma D., Jin S., Gong Y., Wang D. Queue

length estimation for signalized intersections using license plate recognition data. IEEE Intell. Transp. Syst. Mag. 2019;11:209–220. doi: 10.1109/MITS.2019.2919541

[8] Tsakanikas, V.; Dagiuklas, T. Video surveillance systems-current status and future trends. Comput. Electr. Eng. 2018, 70, 736–753.

[9] Shan Du; Ibrahim, M.; Shehata, M.; Badawy, W., "Automatic License Plate Recognition (ALPR): A State-of-the-Art Review," Circuits and Systems for Video Technology, IEEE Transactions on , vol.23, no.2, pp.311,325, Feb. 2013

[10] B. Well and N. Ronald, Two-Dimensional Imaging, Englewood Cliffs, NJ, Prentice Hall, 1995, pp. 505-537.

[11] M. Fang, C. Liang, and X. Zhao, "A method based on rough set and SOFM neural network for the car's plate character recognition, " in Intelligent Control and Automation, WCICA 2004, Fifth World Congress on, 5, 2004, 4037-4040. IEEE.

[12] E. R. Lee, P. K. Kim, and H. J. Kim, "Automatic recognition of a car license plate using color image Processing, " in Image Processing, ICIP-94., IEEE International Conference, 2, 1994, 301-305

#### **Github url:**

<https://github.com/Vamc-44/Plate-Tracker/tree/main>

