**Coco dataset**

The COCO dataset, which stands for Common Objects in Context, is a widely used dataset in computer vision and machine learning research. It is designed to facilitate object detection, segmentation, and captioning tasks. The dataset is managed by the COCO Consortium, which includes Microsoft, Carnegie Mellon University, and several other institutions.

Key features of the COCO dataset include:

Image Collection: COCO contains a large collection of images, with more than 200,000 images in total. These images cover a wide variety of scenes and scenarios.

Object Annotations: Each image in the COCO dataset is annotated with object instances and their corresponding categories. Object instances are outlined with segmentation masks, and each instance is associated with a specific category label. The dataset includes 80 common object categories, such as person, car, dog, etc.

Captions: In addition to object annotations, the COCO dataset also includes captions for images. This makes it suitable for tasks such as image captioning.

Dataset Splits: The dataset is split into training, validation, and testing sets. This division allows researchers to train and evaluate models on different subsets of data.

The COCO dataset has been widely used as a benchmark for various computer vision tasks, including object detection, instance segmentation, and image captioning. Many research papers and projects in the field of machine learning use the COCO dataset to evaluate the performance of their models.

**YOLO variants**

All of these models belong to the YOLOv8 family, a collection of state-of-the-art real-time object detectors. However, each variant offers different trade-offs between accuracy, speed, and model size:

Number of letters:

* n: smallest model, fastest inference but lowest accuracy
* s: small model, good balance of speed and accuracy
* m: medium model, higher accuracy than small models with moderate inference speed
* l: large model, highest accuracy but slowest inference
* x: extra-large model, best accuracy for resource-intensive applications

**EasyOcr**

**Open CV**

Datasets huge no of data jasma tyo data lai herera image yehi ho bhanera thaha huncha we used coco dataset, collection of data to recognize

**Kasari train gareko vanera pani padhnu paryo** -each data set lai 1 round lagaune epoche

When you say that your dataset took twenty-three hours to be trained for ten epochs, it means that a machine learning model was trained on a dataset for a total of ten complete passes through the entire dataset, and the training process took twenty-three hours to complete.

In the context of machine learning, an "epoch" refers to one complete pass through the entire dataset during the training of a model. The training process involves presenting the model with the entire dataset, making predictions, comparing those predictions to the actual values (labels) in the dataset, and adjusting the model's parameters to improve its performance.

Here's a breakdown:

Dataset: The collection of examples (data points) that the model uses to learn patterns.

Epoch: One complete pass through the entire dataset during training.

Training time: The total duration it takes to complete the training process, which includes multiple epochs.

In our case, the model was trained on the dataset for ten epochs, and each epoch took approximately 2.3 hours on average (23 hours / 10 epochs). The training time can be influenced by various factors, including the complexity of the model, the size of the dataset, the computational resources available, and the optimization techniques used during training.

Long training times may suggest that the model or dataset is computationally demanding, and practitioners often consider ways to optimize the training process, such as using more powerful hardware (e.g., GPUs or TPUs), implementing distributed training, or exploring model architectures that are more computationally efficient.

**Dataset annotation kasari hunxa**

Dataset annotation is the process of labeling or tagging data in a way that helps a machine learning model understand and learn from the data. The type of annotation depends on the specific task, such as image classification, object detection, or natural language processing.

**Hamro dataset ma class euta matra hunxa ie License Plate**

**Trained weight ko bare ma bhujnu parxa**

**Mathematical representation f algorithm ko barema**

**YOLO kasari kaam garxa**

**EasyOCR kasari kaam garxa**

**Grayscalling, Thresholding (formula)**

**Segmentation**

**Image processing ko basics**

tools used

programming ;lang

python model train and detection

yolo easy ocr

database firebase

user vehicle ra licence info rakhna lai

web app next js

styling ko lagi shad cn

component ko lagi react , type script for performance .

**Chapter 5: Implementation and Testing**

**5.1. Implementation**

Python was utilized as the primary programming language for model training and object detection, employing the YOLO v8 model and EasyOCR for character recognition. The frontend of the ANPR system is constructed with a React-based project, which adopts a component-based approach to building extensive User Interfaces. Each component functions as an independent entity, amalgamating to form meaningful User Interfaces. Shad cn was applied for visually styling the majority of the components, while framer motion was employed for animations. The UI design of the website was crafted using Figma.

On the backend, the focus was on generating normalized database tables, implemented through Firebase. Firebase served the dual purpose of handling authentication and storing data in database tables for user information, vehicle information and licence plate information. Nextjs was utilized for server-side logic. Finally, Khalti and esewa were chosen as the payment gateway options.

**5.2. Tools**

**5.2.1. Web Application Development Tool**

Nextjs (React)

Shad cn for Styling

**5.2.2. Back-End Tools**

Python (for model training and detection using YOLO and EasyOCR)

Firebase for User information, vehicle information and licence plate information for Authentication, Database and Storage

Nextjs for web application

Khalti for Payment Gateway

**5.2.3. Other Tools**

Vercel for Deployment

Git for Version Control System & GitHub for Collaboration

VS Code for IDE

Epilogue

Task completed

* Employed YOLOv8 on Coco dataset and enhanced license plate recognition with a specialized model trained on RoboFlow dataset, integrated 'best.pt' weights into the application.
* Conducted 23 hours for 10 epochs of training for a dedicated license plate detection model.
* Implemented dual training approach, combining YOLOv8 for general vehicle detection and EasyOCR for efficient license plate recognition.

YOLO Model

Car detect garna ko YOLOV8 Coco dataset bata train gareko model use garera car detect garyo.

Number plate detection ko lagi hamile RoboFlow bata Dataset nikalyo specified for YOLO V8.<https://universe.roboflow.com/roboflow-universe-projects/license-plate-recognition-rxg4e/dataset/4>

Yo data set ma chai pahila dekhi nai data haru annotated with proper bounding box with associated class aauxa. In our case, the License plate matra thiyo detects garnu parne so class ma euta matra object thiyo.

Number plate detection model train garna hamilai 23hr for 10 epochs. It generated a run folder consisting of the train batch validation batch and along with confusion matrix, F1 curve, Precision Confidence curve, Precision Recall Curve, Recall Confidence curve.  
 It generated last.pt and best.pt weight. We used the best.pt weight for our case.

In our program

Step 0: Installing Dependencies

This step installs necessary dependencies, including the YOLO framework (Ultralytics), EasyOCR for OCR, and lapx library.

Step 1: Car Detection

1.1. The script initializes two YOLO models: one for general vehicle detection (coco\_model) and another for license plate detection (np\_model).

1.2. The script reads a video file, processes the first ten frames, and detects vehicles using the COCO-trained YOLO model.

1.3. Detected vehicle bounding boxes are stored in the vehicle\_bounding\_boxes list.

1.4. For each detected vehicle bounding box, a region of interest (ROI) is extracted from the frame.

1.5. The script uses the license plate detector model (np\_model) to identify license plates within the extracted ROIs.

1.6. The script converts the extracted license plate images to grayscale and applies thresholding to create binary images.

Step 2: Read License Plates

2.1. The script initializes an EasyOCR reader (reader) for English.

2.2. The read\_license\_plate function uses the EasyOCR reader to read text from license plate images.

2.3. The script writes the obtained results to a CSV file.

Step 3: Clean-Up License Plate Format

3.1. Character conversion dictionaries (dict\_char\_to\_int and dict\_int\_to\_char) are defined to handle OCR misinterpretations.

3.2. The script defines functions (license\_complies\_format and format\_license) to check and format license plate text.

Step 4: Visualize the Results

4.1. The script processes the entire video, draws bounding boxes around detected vehicles and license plates, and writes the processed video to an output file.

Step 5: Analyzing Results

The script reads the results CSV file, converts the 'license\_text\_score' column to numeric values, calculates the total sum of license text scores for each license plate number, and finds the row with the maximum license text score for each license plate number and track ID. The final result is displayed.

Chatgpt version:

YOLO Model

Car detect garna ko YOLOV8 Coco dataset bata train gareko model use garera car detect garyo.

Number plate detection ko lagi hamile RoboFlow bata Dataset nikalyo specified for YOLO V8. https://universe.roboflow.com/roboflow-universe-projects/license-plate-recognition-rxg4e/dataset/4

Yo data set ma chai pahila dekhi nai data haru annoted with proper bounding box with associated class aauxa. In our case, the License plate matra thiyo detects garnu parne so class ma euta matra object thiyo.

Number plate detection model train garna hamilai 23hr for 10 epochs. It generated a run folder consisting of the train batch validation batch and along with confusion matrix, F1 curve, Precision Confidence curve, Precision Recall Curve, Recall Confidence curve.

It generated last.pt and best.pt weight. We used the best.pt weight for our case.

In our program

Step 0: Installing Dependencies

This step installs necessary dependencies, including the YOLO framework (Ultralytics), EasyOCR for OCR, and lapx library.

Step 1: Car Detection

1.1. The script initializes two YOLO models: one for general vehicle detection (coco\_model) and another for license plate detection (np\_model).

1.2. The script reads a video file, processes the first ten frames, and detects vehicles using the COCO-trained YOLO model.

1.3. Detected vehicle bounding boxes are stored in the vehicle\_bounding\_boxes list.

1.4. For each detected vehicle bounding box, a region of interest (ROI) is extracted from the frame.

1.5. The script uses the license plate detector model (np\_model) to identify license plates within the extracted ROIs.

1.6. The script converts the extracted license plate images to grayscale and applies thresholding to create binary images.

Step 2: Read License Plates

2.1. The script initializes an EasyOCR reader (reader) for English.

2.2. The read\_license\_plate function uses the EasyOCR reader to read text from license plate images.

2.3. The script writes the obtained results to a CSV file.

Step 3: Clean-Up License Plate Format

3.1. Character conversion dictionaries (dict\_char\_to\_int and dict\_int\_to\_char) are defined to handle OCR misinterpretations.

3.2. The script defines functions (license\_complies\_format and format\_license) to check and format license plate text.

Step 4: Visualize the Results

4.1. The script processes the entire video, draws bounding boxes around detected vehicles and license plates, and writes the processed video to an output file.

Step 5: Analyzing Results

The script reads the results CSV file, converts the 'license\_text\_score' column to numeric values, calculates the total sum of license text scores for each license plate number, and finds the row with the maximum license text score for each license plate number and track ID. The final result is displayed.

ChatGPT version:

YOLO Model

Video input garne-> frame by frame read garxa-> YOLOv8 fordetecting vehicles wala model le car motorcycle ani truck detect garxa-> detect gareko vehicle ko bounding box haru store garxa-> tyo vehicle ko bounding box use garera tyo vehicle ko cropped image nikalxa ie region of interest (roi)-> vehicle ko roi hamile license plate model lai feed garxam to detect license plate( yo model chai hamile manually train gareko ho)-> yo np\_model le license plate ko bounding box along with prediction score nikalxa-> Model le nikaleko license pate ko bounding box use garera hamile tyo license plate lai matra crop garxam-> Cropped image lai hamile image processing garna greyscale ma convert garxam using OpenCv function-> Tyo greyscale image lai pheri threshold ma convert garna hamile greyscale image lai OpenCv function ma feed garera threshold image nikalxam-> threshold image lai EasyOCR ma as input dinxam-> Easy OCR le license plate read garera text return garxa-> tyo text chai hamro lience plate ko format sanga milxa ki nai vanera check garxa ani yedi hamro license plate format satisfy garxa vani matra return garxa (In our case license plate format chai B AB 1234 hunxa for nepali embossed number plate-> All the values including vehicle bounding box, license plate bounding box, vehicle confidence score, license plate confidence score , track id ani license plate number etc values chai csv file ma write hunxa-> tyo csv file lai read garera hamile input deko video ma bounding box create garxam which is used for easily visualization of the output-> Finally kun chai license plate ko confidence score dherai aauxa tesla sort garera tyo license plate number along with its track id display hunxa.

**How does the project work?**

The video processing workflow begins by capturing input and systematically reading frames. Utilizing the YOLOv8 model for vehicle detection, the system identifies cars, motorcycles, and trucks, storing bounding boxes for later use. These boxes are employed to extract Region of Interest (ROI) images of vehicles. The cropped vehicle images then undergo license plate detection using a manually trained np\_model, identifying the license plate's bounding box and prediction score. The system further processes the isolated license plate image, converting it to grayscale and then binary using OpenCV functions. The binary image is input to EasyOCR for text recognition, checked against a predefined format, and retained or discarded accordingly. Values, including bounding box coordinates and confidence scores, are logged into a CSV file. The file is read to create bounding boxes in the input video for visualization. The system analyzes license plate confidence scores, sorting and displaying them, emphasizing those with higher confidence. In essence, this process ensures accurate license plate recognition and detailed visualization of detected vehicles in the input video.

**What is completed in our project?**

In our project, we employed a YOLOV8 model trained on the Coco dataset for general vehicle detection, achieving successful results with the detection of cars using the trained weights. To enhance number plate recognition, we utilized the RoboFlow dataset tailored for YOLO V8, featuring annotated data with bounding boxes limited to the License plate class. The dedicated number plate detection model underwent 23 hours of training across 10 epochs, generating a run folder with batches and crucial metrics such as confusion matrix and F1 curve. Notably, the model produced last.pt and best.pt weights, the latter being integrated into our application. The application's script processes a results CSV file, converting the 'license\_text\_score' column, calculating total scores for each license plate, and determining the maximum score for each plate and track ID. In conclusion, our approach involved dual training – a YOLOV8 model for general vehicle detection and a specialized model for license plate detection, yielding a comprehensive solution for efficient car detection and license plate recognition within our application.

**EasyOCR**

EasyOCR is an open-source optical character recognition (OCR) library that is designed to make it easy for developers to integrate OCR capabilities into their applications. OCR is a technology that converts different types of documents, such as scanned paper documents, PDFs, or images captured by a digital camera, into editable and searchable data.

Here's a general overview of how EasyOCR works:

Input:

Image Source: EasyOCR takes an image as input. This image can be sourced from various places, including scanned documents, images, or photos.

Language: It supports multiple languages for character recognition. You can specify the language(s) of the text in the image to improve accuracy.

Preprocessing:

The input image undergoes preprocessing steps to enhance the quality of the image for OCR. This may involve operations like resizing, normalization, or noise reduction to improve the accuracy of character recognition.

Text Detection:

EasyOCR employs methods to detect the regions of the image where text is present. This involves identifying bounding boxes or regions of interest that contain text.

Text Recognition:

The text within the identified regions is then recognized using OCR algorithms. This step involves the conversion of the image-based text into machine-readable text. EasyOCR supports various OCR engines and models, and it may use a deep learning-based approach for character recognition.

Output:

The recognized text is extracted and provided as output. This text can be further processed, stored, or used for various applications, such as data extraction, search, or translation.

It's important to note that EasyOCR, like other OCR libraries, might have different models or engines for different languages and scripts. Some models may be more accurate for specific types of text or languages. Additionally, the accuracy of OCR can be influenced by factors such as image quality, text font, and language complexity.

Developers typically use EasyOCR by integrating it into their applications or workflows through APIs or libraries. The library abstracts away much of the complexity of OCR, making it easier for developers to implement text recognition without delving deeply into the intricacies of the underlying algorithms.

**Objectives**

* Utilizing YOLO-algorithm for automatic license plate recognition of the uploaded video.
* Eliminate manual ticketing for a more efficient entry and exit process in parking facilities.
* Introduce time-based pricing for registered vehicles during check-in and check-out.
* Reduce wait times in the parking facility for an improved user experience.
* Integrate mobile/web app access and automated payment options to ensure seamless transactions.

All of these models belong to the YOLOv8 family, a collection of state-of-the-art real-time object detectors. However, each variant offers different trade-offs between accuracy, speed, and model size:

Number of letters:

* n: smallest model, fastest inference but lowest accuracy
* s: small model, good balance of speed and accuracy
* m: medium model, higher accuracy than small models with moderate inference speed
* l: large model, highest accuracy but slowest inference
* x: extra-large model, best accuracy for resource-intensive applications

**Code Description:**

coco\_model = YOLO('yolov8n.pt')

np\_model = YOLO('../model/runs/detect/train/weights/best.pt')

**coco\_model**: This YOLO model (coco\_model) is initialized using the weights file named '*yolov8n.pt'*. The weights file likely contains pre-trained weights on the COCO dataset.

**np\_model**: This YOLO model (np\_model) is initialized using a different weights file located at '../model/runs/detect/train/weights/best.pt'. The path suggests that these weights may be specific to your use case or have been fine-tuned for a particular task.

videos = glob('./inputs/embossed.mp4')

print(videos)

**glob('./inputs/embossed.mp4')**: This line uses the glob function to find files matching the pattern './inputs/embossed.mp4'. The glob function returns a list of path names that match the specified pattern.

**videos**: This variable stores the list of video files obtained from the glob function. In this case, it appears you are expecting only one video file to match the pattern.

**print(videos)**: This line prints the list of video files to the console.

It's important to note that the glob function supports wildcard patterns, so if you have multiple video files matching the pattern, they will all be included in the list. If you want to process all video files in a directory, you might use a wildcard like '\*.mp4' instead of specifying a specific file name.

video = cv.VideoCapture(videos[0])

**cv.VideoCapture(videoPath)** creates a **VideoCapture** object for the video file specified by the path. Once the **VideoCapture** object is created, we can use it to read each frame from the video, perform various operations on each frame, and analyze or process the video.

ret = True

In OpenCV, the variable **ret** is a boolean that represents whether a frame was successfully read from the video source (file or camera). When **ret** is *True*, it indicates that the frame was successfully read. Conversely, when **ret** is *False*, it means that there are no more frames to be read, or there was an error in reading the frame.

frame\_number = -1

Setting **frame\_number** to -1 in the context of video processing typically means that you want to read the next frame from the video source. The frame number is often used to navigate to a specific frame in a video. If **frame\_number** is -1, it's a signal to read the next frame in sequence.

vehicles = [2,3,5]

We can directly detect multiple vehicles in a single frame like car, motorbike, truck using the COCO dataset. The COCO dataset has a list of vehicle class IDs. Each vehicle class has a unique ID. For example, car is 2, motorbike is 3, truck is 5. We can search this information in<https://docs.ultralytics.com/datasets/detect/coco/#dataset-yaml>

while ret:

ret, frame = video.read()

frame\_number += 1

if ret:

results[frame\_number] = {}

This part of the code is a loop that reads frames from the video using video.read() until there are no more frames (ret becomes False). Let's break down this loop:

ret, frame = video.read(): This line reads a frame from the video using the read() method. It returns two values: ret (a boolean indicating whether the frame was successfully read) and frame (the actual frame).

frame\_number += 1: This line increments the frame\_number variable, indicating the current frame number.

if ret:: This condition checks if the frame was successfully read. If ret is True, it means a frame was read, and the code inside the if block will be executed.

results[frame\_number] = {}: This line initializes an empty dictionary at the current frame\_number in the results dictionary. This dictionary will be used to store information about detections in the current frame.The loop continues until there are no more frames (ret becomes False). Inside the loop, you perform vehicle and license plate detection for each frame and store the results in the results dictionary.

detections = coco\_model.track(frame, *persist*=True)[0]

coco\_model.track(frame, persist=True): This is calling a method or function named track on the coco\_model object. It performs object tracking on the input frame. The frame is passed as an argument to this method.

[0]: The result of this method call is likely a list or iterable containing tracking information. [0] is used to get the first element of this list. The method returns a list, and the tracked objects are in the first element of that list.

So, after this line, detections hold information about the tracked objects in the current frame. It includes information like the bounding boxes, track IDs, scores, class IDs, etc., for the detected objects in the frame.

for detection in detections.boxes.data.tolist():

x1, y1, x2, y2, track\_id, score, class\_id = detection

if int(class\_id) in vehicles and score > 0.5:

vehicle\_bounding\_boxes = []

vehicle\_bounding\_boxes.append([x1, y1, x2, y2, track\_id, score])

In this part of the code, you are iterating over the detections obtained from coco\_model.track() and filtering them based on certain criteria.  
 for detection in detections.boxes.data.tolist()::  
 This loop iterates over each detection in the list of boxes obtained from detections.

x1, y1, x2, y2, track\_id, score, class\_id = detection:  
 This line unpacks the values in each detection box into individual variables for easier access. The values typically include coordinates (x1, y1, x2, y2), a track ID, a confidence score, and a class ID.

if int(class\_id) in vehicles and score > 0.5::  
 This condition checks whether the class ID is in the list of vehicles and if the confidence score (score) is greater than 0.5. If both conditions are true, it means the detected object is a vehicle with a confidence score above the specified threshold.

vehicle\_bounding\_boxes = []:  
 This line initializes an empty list to store the bounding box information of the detected vehicles.

vehicle\_bounding\_boxes.append([x1, y1, x2, y2, track\_id, score]):  
 If the conditions are met, the bounding box information for the current detection is appended to the vehicle\_bounding\_boxes list.

for bbox in vehicle\_bounding\_boxes:

print(bbox)

roi = frame[int(y1):int(y2), int(x1):int(x2)]

for bbox in vehicle\_bounding\_boxes:: This loop iterates over each bounding box in vehicle\_bounding\_boxes.

print(bbox):  
 This line prints the bounding box information. The information includes x1, y1, x2, y2, track\_id, and score.

x1, y1, x2, y2, track\_id, score = bbox:  
 This line unpacks the values in the bounding box into individual variables for easier access. The variables now hold the coordinates (x1, y1, x2, y2), the track ID, and the confidence score.

roi = frame[int(y1):int(y2), int(x1):int(x2)]:  
 This line extracts the region of interest (ROI) from the current frame using the bounding box coordinates. It's creating a sub-image from the frame that corresponds to the region where the detected vehicle is.

license\_plates = np\_model(roi)[0]

np\_model(roi):  
 This code is applying the np\_model to the region of interest (roi). The result is likely a list or iterable containing information about detected license plates.

[0]:  
 This is used to get the first element of the result. If the np\_model returns a list of predictions, [0] is used to access the first prediction.

So, license\_plates now hold information about the detected license plates in the region of interest.

for license\_plate in license\_plates.boxes.data.tolist():

plate\_x1, plate\_y1, plate\_x2, plate\_y2, plate\_score, \_ = license\_plate

for license\_plate in license\_plates.boxes.data.tolist()::

This loop iterates over each detected license plate in license\_plates.

plate\_x1, plate\_y1, plate\_x2, plate\_y2, plate\_score, \_ = license\_plate:

This line unpacks the values in each detected license plate box into individual variables for easier access. The variables now hold the coordinates of the license plate bounding box (plate\_x1, plate\_y1, plate\_x2, plate\_y2), the confidence score (plate\_score), and possibly other information (ignored using \_).

plate = roi[int(plate\_y1):int(plate\_y2), int(plate\_x1):int(plate\_x2)]

plate = roi[int(plate\_y1):int(plate\_y2), int(plate\_x1):int(plate\_x2)]:

This line extracts the license plate region from the ROI using the previously obtained bounding box coordinates (plate\_x1, plate\_y1, plate\_x2, plate\_y2). It uses integer casting to ensure that the coordinates are treated as integers.

cv.imwrite('outputs/plates/roi/'+str(track\_id)+ '.jpg', plate):

This line uses OpenCV's imwrite function to save the extracted license plate as an image file. The file is saved in a directory named 'outputs/plates/roi/' with the filename as str(track\_id) + '.jpg'. This assumes that track\_id is a unique identifier for the current detected vehicle.

plate\_gray = cv.cvtColor(plate, cv.COLOR\_BGR2GRAY)

\_, plate\_treshold = cv.threshold(plate\_gray, 64, 255, cv.THRESH\_BINARY\_INV)

plate\_gray = cv.cvtColor(plate, cv.COLOR\_BGR2GRAY):

This line converts the license plate image (plate) from BGR (color) to grayscale using OpenCV's cvtColor function. Grayscale simplifies the image to a single channel, which is often useful for various image processing tasks.

\_, plate\_threshold = cv.threshold(plate\_gray, 64, 255, cv.THRESH\_BINARY\_INV):

This line applies a binary thresholding operation to the grayscale license plate image (plate\_gray). Pixels with intensity values less than 64 are set to 0, and pixels with values greater than or equal to 64 are set to 255. The cv.THRESH\_BINARY\_INV flag inverts the binary threshold, making foreground pixels (the license plate characters) white and background pixels black.

The underscore (\_) is often used as a placeholder for a variable that you don't intend to use. In this case, the threshold value returned by cv.threshold is not used, so it's assigned to \_ to indicate that it's intentionally ignored.

np\_text, np\_score = read\_license\_plate(plate\_treshold)

read\_license\_plate(plate\_threshold): This line calls the read\_license\_plate function, passing the thresholded license plate image (plate\_threshold) as an argument. The function is expected to return information about the recognized text and possibly a confidence score.

np\_text, np\_score = ...: This line unpacks the return values from the read\_license\_plate function into two variables, np\_text and np\_score. This assumes that the function returns a tuple or a pair of values representing the recognized text and the associated confidence score.

def read\_license\_plate(*license\_plate\_crop*):

detections = reader.readtext(*license\_plate\_crop*)

for detection in detections:

bbox, text, score = detection

text = text.upper().replace(' ', '')

*# verify that text is confirmed to a standard license plate*

if license\_complies\_format(text):

*# bring text into the default license plate format*

return format\_license(text), score

return None, None

reader.readtext(license\_plate\_crop): This line uses the reader.readtext function to perform OCR on the license plate image (license\_plate\_crop). It returns a list of detections, where each detection includes the bounding box (bbox), recognized text (text), and a confidence score (score).

for detection in detections:: This loop iterates over each detection obtained from the OCR.

bbox, text, score = detection: This line unpacks the values from each detection into individual variables for easier access.

text = text.upper().replace(' ', ''): This line converts the recognized text to uppercase and removes any spaces. This is likely done to standardize the format of the license plate text.

if license\_complies\_format(text):: This condition checks whether the processed text complies with a certain format. If it does, the text is further processed.

return format\_license(text), score: If the text complies with the format, the function returns the formatted license plate text and the confidence score.

return None, None: If no valid license plate is found, the function returns None for both the license plate text and the confidence score.

reader = easyocr.Reader(['en'], *gpu*=True)

Initialize the OCR reader. EasyOCR is an open-source library for optical character recognition (OCR) that is used for text recognition. 'en' is the language of the OCR reader.

dict\_char\_to\_int = {'O': '0',

'I': '1',

'J': '3',

'A': '4',

'G': '6',

'S': '5'}

dict\_int\_to\_char = {'0': 'O',

'1': 'I',

'3': 'J',

'4': 'A',

'6': 'G',

'5': 'S'}

Here we are mapping dictionaries for character conversion.If we know that the first character in the number plate always is an string eg. `O` then if our OCR reader reads that O as `0` then it would be a mistake. To prevent this we are mapping dictionaries with similar keys and values.

def license\_complies\_format(*text*):

if len(*text*) != 7:

return False

if (*text*[0] in string.ascii\_uppercase or *text*[0] in dict\_int\_to\_char.keys()) and \

(*text*[1] in string.ascii\_uppercase or *text*[1] in dict\_int\_to\_char.keys()) and \

(*text*[2] in string.ascii\_uppercase or *text*[2] in dict\_int\_to\_char.keys()) and \

(*text*[3] in ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'] or *text*[3] in dict\_char\_to\_int.keys()) and \

(*text*[4] in ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'] or *text*[4] in dict\_char\_to\_int.keys()) and \

(*text*[5] in ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'] or *text*[5] in dict\_char\_to\_int.keys()) and \

(*text*[6] in ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'] or *text*[6] in dict\_char\_to\_int.keys()):

return True

else:

return False

license\_complies\_format is a function that checks if the license plate complies with the specified format. In this case the format is `[A-Z][A-Z][A-Z][0-9][0-9][0-9]`. We can change this format for specific use cases. For example, now it is configured for Nepali Embossed number plates.

The above character conversion comes handy in this situation where if we are sure that in the second letter of our text we should get a string then if our OCR Reader reads a integer that looks similar to an alphabet maybe 4 then we can neglect the '4 and read 'A' instead.

def format\_license(*text*):

license\_plate\_ = ''

mapping = {0: dict\_int\_to\_char, 1: dict\_int\_to\_char, 2: dict\_int\_to\_char, 3: dict\_char\_to\_int, 4: dict\_char\_to\_int,

5: dict\_char\_to\_int, 6: dict\_char\_to\_int}

for j in [0, 1, 2, 3, 4, 5, 6]:

if *text*[j] in mapping[j].keys():

license\_plate\_ += mapping[j][*text*[j]]

else:

license\_plate\_ += *text*[j]

return license\_plate\_

license\_plate\_ = '': Initializes an empty string to store the formatted license plate.

mapping: A dictionary that maps the position in the license plate text to the corresponding conversion dictionary (dict\_int\_to\_char or dict\_char\_to\_int).

for j in [0, 1, 2, 3, 4, 5, 6]:: Iterates over each position in the license plate text.

if text[j] in mapping[j].keys():: Checks if the character at the current position (j) is present in the corresponding conversion dictionary. If it is, it appends the converted character to license\_plate\_.

else:: If the character is not in the dictionary, it means there's no conversion needed, so the original character is appended to license\_plate\_.

return license\_plate\_: Returns the formatted license plate.

This function essentially applies character conversion based on the specified mappings for each position in the license plate text. If a character at a certain position needs conversion, it uses the appropriate dictionary for that position.

if np\_text is not None:

results[frame\_number][track\_id] = {

'car': {

'bbox': [x1, y1, x2, y2],

'bbox\_score': score},

'license\_plate': {

'bbox': [plate\_x1, plate\_y1, plate\_x2, plate\_y2],

'bbox\_score': plate\_score,

'number': np\_text,

'text\_score': np\_score

}

}

if np\_text is not None:: This condition checks whether np\_text contains a non-None value, meaning that the OCR process successfully recognized a license plate text.

results[frame\_number][track\_id]: If the condition is True, you are updating the results dictionary. The frame\_number is used as a key to access a dictionary representing a specific frame, and within that frame dictionary, the track\_id is used as a key to store information about a specific tracked object.

'car': This key represents information about the detected car, including the bounding box (bbox) and the confidence score (bbox\_score).

'license\_plate': This key represents information about the license plate, including the bounding box (bbox), confidence score (bbox\_score), recognized number (number), and the text score (text\_score).

So, if a license plate was successfully recognized (np\_text is not None), the relevant information is added to the results dictionary for the specific frame and tracked object.

write\_csv(results, './outputs/results.csv')

video.release()

write\_csv(results, './outputs/results.csv'): This line is calling a function named write\_csv with the results dictionary and a file path as arguments. It is using this function to write the results to a CSV file.

video.release(): This line releases the video capture object (video). It's essential to release the video capture object once you have finished using it to free up system resources.

User

The system should facilitate user registration by allowing individuals to create an account. Users must be able to register their vehicles within the system. The registration process should detect and extract license plate information from the uploaded videos. Upon the successful detection of license plates, the system is responsible for generating bills to the user corresponding to the identified vehicles. Users should have the ability to access their generated bills within their accounts. The system must incorporate a secure payment gateway, allowing users to make payments online.

Admin

Admins are granted the privilege to upload videos containing moving vehicles for processing. These uploaded videos should be securely stored within the system and made accessible for processing. The system is responsible for generating bills to the admin corresponding to the identified vehicles.

Yolo V8

Working Principle: YOLOv8 is a state-of-the-art object detection algorithm that was first released in May 2023. It is the latest version of the popular YOLO (You Only Look Once) family of algorithms, which are known for their speed and accuracy.

YOLOv8 is based on a deep convolutional neural network (CNN) architecture that is similar to its predecessors. However, it introduces a number of new features and improvements, including:

· A new backbone architecture called CSPNet, which is more efficient and accurate than previous backbones.

· A new neck architecture called FPN+PAN, which better aggregates features from different levels of the backbone.

· A new head architecture called PANet, which is more robust to occlusion and scale variations.

· A new training procedure that uses a combination of supervised and unsupervised learning.

Working principle of YOLOv8

YOLOv8 works by first dividing the input image into a grid of cells. For each cell, YOLOv8 predicts a set of bounding boxes, along with the class probabilities for each bounding box.

YOLOv8 then uses a non-maxima suppression (NMS) algorithm to filter out overlapping bounding boxes and select the most likely bounding boxes for each object in the image.

COCO (Common Objects in Context) is the industry standard benchmark for evaluating object detection models. When comparing models on COCO, we look at the mAP value and FPS measurement for inference speed. Models should be compared at similar inference speeds.:YOLOv8 COCO accuracy is state of the art for models at comparable inference latencies as of writing this post.

Difference between variants of Yolo V8: YOLOv8 is available in three variants: YOLOv8, YOLOv8-L, and YOLOv8-X. The main difference between the variants is the size of the backbone network. YOLOv8 has the smallest backbone network, while YOLOv8-X has the largest backbone network.

The larger backbone network in YOLOv8-X gives it better accuracy, but it also makes it slower than YOLOv8 and YOLOv8-L.

Advantages of YOLOv8 over previous versions of YOLO

YOLOv8 has a number of advantages over previous versions of YOLO, including:

· Accuracy: YOLOv8 is more accurate than previous versions of YOLO on a variety of object detection benchmarks.

· Speed: YOLOv8 is faster than previous versions of YOLO, especially on smaller devices.

· Robustness: YOLOv8 is more robust to occlusion and scale variations than previous versions of YOLO.

Overall, YOLOv8 is a state-of-the-art object detection algorithm that is more accurate, faster, and more robust than previous versions of YOLO.

Here is a table that summarizes the key differences between the different variants of YOLOv8:

Variant

Backbone network

Accuracy

Speed

Robustness

YOLOv8

Small

Good

Fast

Good

YOLOv8-L

Medium

Great

Medium

Great

YOLOv8-X

Large

Excellent

Slow

Excellent

drive\_spreadsheetExport to Sheets

Which variant of YOLOv8 you choose will depend on your specific needs. If you need the fastest possible object detection algorithm, then YOLOv8 is a good choice. If you need the most accurate possible object detection algorithm, then YOLOv8-X is a good choice. If you need a balance between accuracy and speed, then YOLOv8-L is a good choice.

Here are a few main reasons why you should consider using YOLOv8 for your next computer vision project:

YOLOv8 has a high rate of accuracy measured by COCO and Roboflow 100.

YOLOv8 comes with a lot of developer-convenience features, from an easy-to-use CLI to a well-structured Python package.

There is a large community around YOLO and a growing community around the YOLOv8 model, meaning there are many people in computer vision circles who may be able to assist you when you need guidance.

YOLOv8 achieves strong accuracy on COCO. For example, the YOLOv8m model — the medium model — achieves a 50.2% mAP when measured on COCO. When evaluated against Roboflow 100, a dataset that specifically evaluates model performance on various task-specific domains, YOLOv8 scored substantially better than YOLOv5. More information on this is provided in our performance analysis later in the article.

Furthermore, the developer-convenience features in YOLOv8 are significant. As opposed to other models where tasks are split across many different Python files that you can execute, YOLOv8 comes with a CLI that makes training a model more intuitive. This is in addition to a Python package that provides a more seamless coding experience than prior models.

The community around YOLO is notable when you are considering a model to use. Many computer vision experts know about YOLO and how it works, and there is plenty of guidance online about using YOLO in practice. Although YOLOv8 is new as of writing this piece, there are many guides online that can help.

Here are a few of our own learning resources that you can use to advance your knowledge of YOLO:

YOLOv8 Model Card on Roboflow Models

How to Train a YOLOv8 Model on a Custom Dataset

How to Deploy a YOLOv8 Model to a Raspberry Pi

Google Colab Notebook for Training YOLOv8 Object Detection Models

Google Colab Notebook for Training YOLOv8 Classification Models

Google Colab Notebook for Training YOLOv8 Segmentation Models

Track and Count Vehicles with YOLOv8 and ByteTRACK

Let’s do a deep dive into the architecture and what makes YOLOv8 different from prior YOLO models.

YOLOv8 Architecture: A Deep Dive

YOLOv8 does not yet have a published paper, so we lack direct insight into the direct research methodology and ablation studies done during its creation. With that said, we analyzed the repository and information available about the model to start documenting what’s new in YOLOv8.

If you want to peer into the code yourself, check out the YOLOv8 repository and you view this code differential to see how some of the research was done.

Here we provide a quick summary of impactful modeling updates and then we will look at the model’s evaluation, which speaks for itself.

The following image made by GitHub user RangeKing shows a detailed visualisation of the network’s architecture.

Yolo V8