**License Plate Detector Model creation:**

Dataset: <https://universe.roboflow.com/roboflow-universe-projects/license-plate-recognition-rxg4e/dataset/4>

Procedure:

1. Dataset contained of about 24242 images where there were 21174 images in the training set, 2048 images in the validation set and 1020 images in the testing set in the ratio of 87:8:4.
2. The dataset was trained upon the YOLOV8 model using 20 epochs and image size as 640\*640.
3. The model was validated with the validation set and it produced satisfying results.
4. In addition to the validation results, the model was tested against several test images and the results were checked manually.

**Decoding the entire process (main.py):**

1. **Libraries used:**

* **ultralytics.YOLO**: This library is used for object detection using the YOLO (You Only Look Once) model.
* **cv2:** OpenCV library for computer vision tasks.
* **util:** A custom utility module.
* **sort.sort:** An object tracking algorithm called SORT (Simple Online and Realtime Tracking).
* **matplotlib:** A plotting library

1. **Loading the models:**

* **yolov8n.pt:** Pretrained YOLOV8 model trained on the COCO Dataset (https://cocodataset.org) consisting of several classes. Only classes 2, 3, 5 and 7 were considered. (car, motorbike, bus, truck)
* **license\_plate\_detector.pt:** YOLOV8 model trained on the above-mentioned [dataset](https://universe.roboflow.com/roboflow-universe-projects/license-plate-recognition-rxg4e/dataset/4).

1. **results:** A dictionary to store the results of the object detection, tracking, and license plate recognition
2. **Loop for reading the video:**

* Iterating through each frame of the video using a while loop.
* Incrementing the frame\_number for each iteration.
* Creating an empty dictionary for the current frame in the results dictionary.

1. **Vehicle Detection:**

* Using the YOLO model (yolov8n.pt) to detect objects in the current frame.
* Extracting information (bounding box coordinates, score, class ID) for each detected object.
* Filtering out objects that belong to the specified vehicle classes (vehicles).

1. **Vehicle Tracking:**

* Using the SORT (Simple Online and Realtime Tracking) algorithm to associate and track detected vehicles across frames.
* obj\_tracker.update takes the list of current detections (obj\_detections) and returns updated track IDs.

1. **License Plate Detection:**

* Using the YOLO model (license\_plate\_detector) to detect license plates in the current frame.
* Extracting information (bounding box coordinates, score, class ID) for each detected license plate.

1. **Assign License Plate to Car:**

* Utilizing a custom function get\_car to associate each detected license plate with a tracked car.
* The function returns the bounding box and ID of the corresponding car.

1. **Check Car Association and License Plate Cropping:**

* Ensuring that a valid car ID is obtained from the get\_car function.
* Extracting the region of the frame containing the detected license plate using its coordinates.

1. **License Plate Processing:**

* Converting the cropped license plate to grayscale and applying thresholding to create a binary image.

1. **License Plate Recognition:**

* Using a custom function read\_license\_plate to recognize the text on the license plate from the binary image.
* The function returns the recognized text and its associated confidence score.

1. **Results Storage and writing to local system:**

* Storing the results in the results dictionary for the current frame, including the bounding box and recognized text of the license plate.
* Using a custom function write\_csv to write the results to a CSV file.

**Customized functions in the util.py file:**

1. **write\_csv:**

* The write\_csv function creates a CSV file from a dictionary (results) that contains information about detected vehicles, their associated license plates, and relevant details such as bounding boxes and recognition scores.
* For each frame and car ID, the function checks if valid car and license plate data exist before writing the information to the CSV file.
* The CSV file structure includes columns for frame number (frame\_nmr), car ID (car\_id), car bounding box (car\_bbox), license plate bounding box (license\_plate\_bbox), bounding box score (license\_plate\_bbox\_score), recognized license plate number (license\_number), and its confidence score (license\_number\_score).
* The function ensures that only entries with complete information are written to the CSV file.

1. **get\_car:**

* The get\_car function is designed to associate a detected license plate with a tracked vehicle. It takes the coordinates of the license plate (x1, y1, x2, y2, score, class\_id) and a list of tracked vehicles (vehicle\_track\_ids).
* The function iterates through the tracked vehicles, checking if the license plate coordinates fall within the bounding box of each vehicle.
* If a match is found, it returns the coordinates (x1, y1, x2, y2) and ID of the corresponding vehicle. If no match is found, it returns a tuple of -1 values.
* The function aims to link license plate information to the encompassing vehicle in the tracking data.

1. **read\_license\_plate:**

* The `read\_license\_plate` function extracts text from a cropped license plate image using an optical character recognition (OCR) library.
* It utilizes the `reader.readtext` method to obtain text detections with bounding boxes and confidence scores.
* The function iterates through the detections, converting the recognized text to uppercase and removing spaces.
* If the formatted text complies with a predefined license plate format, it is returned along with the confidence score; otherwise, `None` values are returned.

**License Plate Format:**

As we have obtained sufficient data for the UK license plates, we have considered the UK license plate format. The license plate format is described as below:



To adhere to a specific format and handle potential exceptions, a set of rules is defined to ensure consistency in the recognized text. Dictionaries ‘dict\_char\_to\_int’ and ‘dict\_int\_to\_char’ are used to map characters to digits and vice-versa.

Dictionary dict\_char\_to\_int maps [O, I, J, A, G, S] to [0, 1, 3, 4, 6, 5] respectively and dict\_int\_to\_char does the same mapping, vice-versa.

The rules are expressed in two functions:

1. **license\_complies\_format:**

* The license\_complies\_format function verifies whether a given license plate text adheres to a specific format.
* It first checks the length of the text, requiring it to be exactly 7 characters long. Then, it evaluates each character based on predefined rules: the first two characters must be either uppercase letters or mapped to specific characters, the third and fourth characters must be digits or mapped to specific characters, and the last three characters must be uppercase letters or mapped to specific characters.
* The function utilizes dictionaries (dict\_char\_to\_int and dict\_int\_to\_char) to handle character substitutions, ensuring consistency in the format.
* If the text meets these criteria, the function returns True; otherwise, it returns False.

1. **format\_license:**

* The format\_license function serves to transform a provided license plate text into a standardized format by employing character substitutions according to predefined rules.
* It iterates through each character in the input text, determining whether a character at a specific position should undergo conversion based on the mappings stored in dictionaries (dict\_int\_to\_char and dict\_char\_to\_int).
* If a character exists in the mapping dictionary, it is replaced; otherwise, the original character is retained.
* The resulting formatted license plate text is then returned, ensuring consistency and uniformity in representation.

**Adding the missing data:**

Sometimes, couple of frames miss out from the test.csv file where the detections and the co-ordinates of the bounding boxes are stored. In order to restore the data, the average of the frames before and after are considered. The summary of the file add\_missing\_data.py is described as below:

1. **Data Extraction:** The script reads a CSV file containing information such as frame numbers, car IDs, car bounding boxes, license plate bounding boxes, and associated scores.
2. **Interpolation Logic:** It identifies unique car IDs and iterates over each, interpolating missing frames between consecutive data points. Linear interpolation is applied to estimate bounding boxes for frames that lack explicit data.
3. **Imputation for Missing Data:** For each interpolated frame, the script creates a new row with imputed values for license plate bounding boxes, scores, and numbers. If the frame is part of the original data, the script retains the provided values.
4. **Output CSV Generation:** The resulting interpolated data is written to a new CSV file named 'test\_interpolated.csv'. The CSV file includes columns for frame numbers, car IDs, interpolated car bounding boxes, interpolated license plate bounding boxes, associated scores, and license plate numbers.
5. **Libraries Used:** The script leverages libraries such as NumPy for numerical operations, SciPy for interpolation functionality, and the CSV module for reading and writing CSV files.
6. **Data Handling:** The script effectively manages missing data, performing interpolation where necessary, and ensuring consistent formatting in the output CSV file.
7. **File I/O:** The script reads the input CSV file and writes the interpolated data to a new CSV file, facilitating the analysis of vehicle and license plate information over a continuous sequence of frames.
8. **Customization:** The script allows for customization of header names and interpolation methods based on the specific requirements of the dataset.

**Identifying cars and their number plates and visualizing them in the video itself:**

This whole process is done in visualize.py. Its description is as below:

1. **Data Loading:** Reads a CSV file ('test\_interpolated.csv') containing information about vehicle and license plate bounding boxes, frame numbers, and scores.
2. **Video Loading:** Loads a video file ('sample.mp4') using OpenCV for frame-by-frame processing.
3. **Interpolation and Visualization:** Interpolates missing bounding boxes between consecutive frames using the provided data. Draws bounding boxes for detected vehicles and license plates on each frame.
4. **Text and Visual Overlay:** Overlays the license plate text and additional visual elements on the video frames.
5. **Video Writing:** Creates an output video file ('out.mp4') with visualizations and bounding box overlays.
6. **Exception Handling:** Handles cases where frames or bounding boxes are missing, preventing potential errors during execution.
7. **Output Information:** Prints a message indicating whether the video writer was successfully closed.
8. **Customization:** Allows customization of video paths, CSV file paths, and other parameters for flexibility.
9. **Dependencies:** Utilizes OpenCV, NumPy, and Pandas for video processing, numerical operations, and data handling, respectively.