# Vehicle License Plate Detection using YOLOv8

**Wenhao Fang: N01555914**

**Data: 25/11/2023**

## Abstract

This project delves into implementing YOLOv8, a state-of-the-art object detection algorithm in computer vision, focusing on developing a custom model for car plate prediction. The comprehensive workflow includes data collection using a Google image downloader, data processing involving cleansing and annotation, establishment of the YOLOv8 environment, model configuration, and training of the custom detector. Performance evaluation presents high precision, recall, and mAP50-95 values. Predictions on new images affirm the model's efficacy in real-world scenarios, emphasizing its utility for car plate detection tasks. The project concludes with a pipeline to train custom YOLO models.

## 1. Introduction

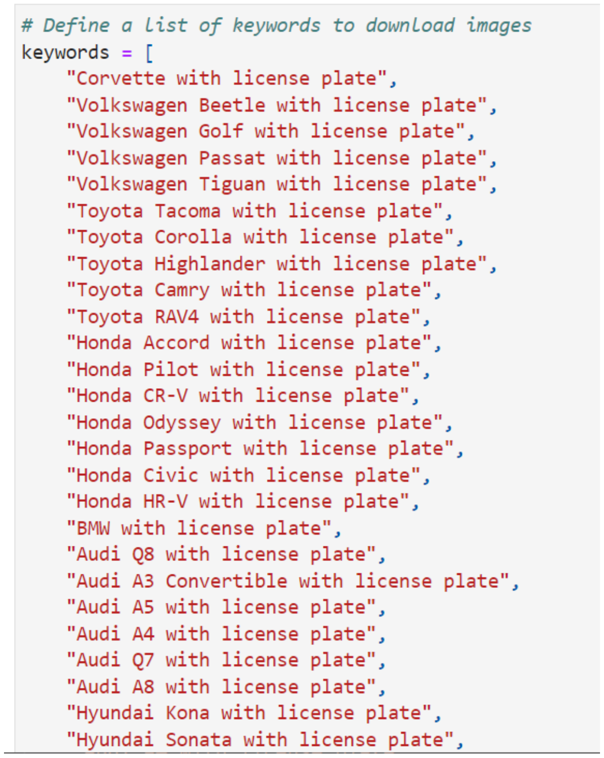
YOLOv8, or You Only Look Once version 8, is a cutting-edge object detection deep learning algorithm model in computer vision. Renowned for its speed and accuracy, YOLOv8 processes entire images in a single pass through the neural network, predicting bounding boxes and class probabilities for multiple objects simultaneously.

In my last data analytics coursework, I trained a custom model to predict Humber College's logo. While this highlighted practical applications in data analytics, it also revealed gaps in my understanding in terms of deep learning. Therefore, I started this project focusing on the hands-on skills and knowledge regarding deep learning. In subsequent sections, I will navigate through the practical steps to create a custom YOLO model to predict car plates.

## 2. Data Collection

Unlike the alphanumeric data commonly used in the machine learning course, the data used in this project are image-based, leading to a challenge to collect data. While manually collecting images from daily surroundings, such as capturing vehicle plates on busy city roads, is an option, it proves time-consuming and burdensome, especially given the nature of media data.

To address this challenge, I used a Google image downloader named '**simple-image-download**', a Python script to search for and download images from Google Images. I listed various keywords of automotive brands to simplify the collection process and ensure a diverse dataset for YOLO model training.



```python
# Define a list of keywords to download images
keywords = [
    "Corvette with license plate",
    "Volkswagen Beetle with license plate",
    "Volkswagen Golf with license plate",
    "Volkswagen Passat with license plate",
    "Volkswagen Tiguan with license plate",
    "Toyota Tacoma with license plate",
    "Toyota Corolla with license plate",
    "Toyota Highlander with license plate",
    "Toyota Camry with license plate",
    "Toyota RAV4 with license plate",
    "Honda Accord with license plate",
    "Honda Pilot with license plate",
    "Honda CR-V with license plate",
    "Honda Odyssey with license plate",
    "Honda Passport with license plate",
    "Honda Civic with license plate",
    "Honda HR-V with license plate",
    "BMW with license plate",
    "Audi Q8 with license plate",
    "Audi A3 Convertible with license plate",
    "Audi A5 with license plate",
    "Audi A4 with license plate",
    "Audi Q7 with license plate",
    "Audi A8 with license plate",
    "Hyundai Kona with license plate",
    "Hyundai Sonata with license plate",
```

***Figure 1**.Keywords list*

## 3. Data Processing

### 3.1. Data Cleansing

Though the "**simple-image-download**" script provides a convenient way to collect images that include car plates, it also downloads irrelevant images. Thus, a data cleansing process is required. It involved the removal of all irrelevant images, ensuring that only images in the jpeg format were retained. Eventually, 543 jpeg files are relocated to a designated folder named "**data_label_image**" as a unified data source for the subsequent data processing step.

### 3.2. Image Annotation

To train the object detector, bounding box annotations within images are required to supervise its learning. To complete this critical task, I facilitated a Python program named "**labelImg**" to crop and label car plates with the class name "**car_plate**" for each image. This program allows for accurate annotation and generates a Txt file for each image, containing detailed information about the area occupied by the car plate.

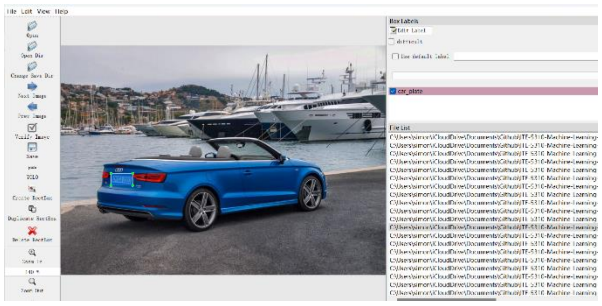


***Figure 2**. Object annotation*

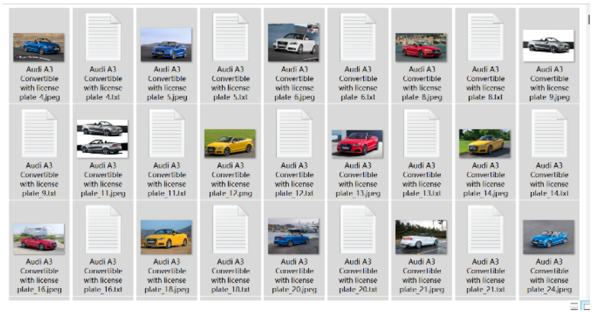

Audi A3 Convertible with license plate 4.jpeg
Audi A3 Convertible with license plate 4.txt
Audi A3 Convertible with license plate 5.jpeg
Audi A3 Convertible with license plate 5.txt
Audi A3 Convertible with license plate 6.jpeg
Audi A3 Convertible with license plate 6.txt
Audi A3 Convertible with license plate 8.jpeg
Audi A3 Convertible with license plate 8.txt
Audi A3 Convertible with license plate 9.jpeg
Audi A3 Convertible with license plate_9.txt
Audi A3 Convertible with license plate_11.jpeg
Audi A3 Convertible with license plate_11.txt
Audi A3 Convertible with license plate_12.png
Audi A3 Convertible with license plate_12.txt
Audi A3 Convertible with license plate_13.jpeg
Audi A3 Convertible with license plate_13.txt
Audi A3 Convertible with license plate_14.jpeg
Audi A3 Convertible with license plate_14.txt
Audi A3 Convertible with license plate_16.jpeg
Audi A3 Convertible with license plate_16.txt
Audi A3 Convertible with license plate_18.jpeg
Audi A3 Convertible with license plate_18.txt
Audi A3 Convertible with license plate_20.jpeg
Audi A3 Convertible with license plate_20.txt
Audi A3 Convertible with license plate_21.jpeg
Audi A3 Convertible with license plate_21.txt
Audi A3 Convertible with license plate_24.jpeg

***Figure 3**.Images and Txt files*

## 4. Configure YOLOv8 Environment and Model

### 4.1. Dataset Architecture

The YOLOv8 model demands a specific directory architecture for optimal functioning. Thus, I defined a function to establish the necessary directory structure. This function efficiently splits and copies image files and their corresponding text files into distinct directories. Files were split at a 8:1:1 (training/validation/prediction) ratio, resulting in 431 images for training, 56 images for validation, and 56 for prediction.

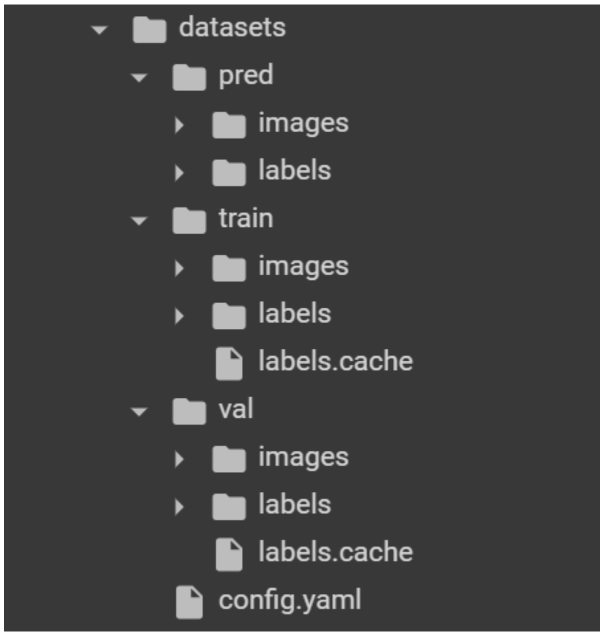


***Figure 4:** Dataset Architecture*

### 4.2. Dedicated Environment

It is recommended to establish a dedicated environment to train a custom YOLO model. I leveraged **Google Colab**, a hosted Jupyter Notebook service, and installed the "**ultralytics**" library, the core library to train a custom model.

### 4.3. YOLO Model Configuration

To connect labels with images and Txt files, “config.yaml", a configuration file, was created in the "datasets" directory. This pivotal file contains the paths of train and val images, along with the label name.

This project utilizes the base **yolov8n.pt** model as a starting point for car plate detection using the default arguments. The default values of the critical arguments are:

- epochs:100, number of epochs to train for,

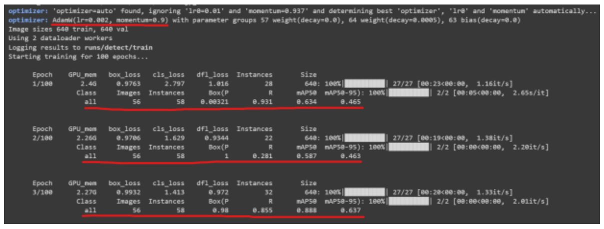
- batch: 16, number of images per batch,

- optimize: auto, optimizer to use,

- patience: 50, epochs to wait for no observable improvement for early stopping of training.

## 5. Train a Custom YOLOv8 Detector

At the beginning of the training, more detailed arguments were shown. For example, the optimizer used in this project is AdamW, a gradient descent method. The learning rate was 0.002 and momentum was 0.9.



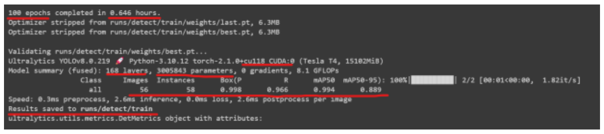
```
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' and 'momentum' automatically...
optimizer: AdamW(lr=0.002, momentum=0.9) with parameter groups 57 weight(decay=0.0), 64 weight(decay=0.0005), 63 bias(decay=0.0)
Image sizes 640 train, 640 val
Using 2 dataloader workers
Logging results to runs/detect/train
Starting training for 100 epochs...

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      1/100      2.4G     0.9763      2.797      1.016         28        640: 100%|██████████| 27/27 [00:23<00:00,  1.16it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 2/2 [00:05<00:00,  2.65s/it]
                   all         56         58    0.00321      0.931      0.634      0.465

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      2/100     2.26G     0.9706      1.629     0.9344         22        640: 100%|██████████| 27/27 [00:19<00:00,  1.38it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 2/2 [00:00<00:00,  2.20it/s]
                   all         56         58          1      0.281      0.587      0.463

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      3/100     2.27G     0.9932      1.413      0.972         32        640: 100%|██████████| 27/27 [00:20<00:00,  1.33it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 2/2 [00:00<00:00,  2.01it/s]
                   all         56         58       0.98      0.855      0.888      0.637
```

***Figure 5:*** *Metrics*

During the training process, a suite of crucial metrics for each epoch, including box_loss and cls_loss, were continuously displayed. These metrics served as indicators, monitoring the performance of the training process.

## 6. Performance Evaluation

A summary was displayed when the training was completed.



```
100 epochs completed in 0.646 hours.
Optimizer stripped from runs/detect/train/weights/last.pt, 6.3MB
Optimizer stripped from runs/detect/train/weights/best.pt, 6.3MB

Validating runs/detect/train/weights/best.pt...
Ultralytics YOLOv8.0.219 🚀 Python-3.10.12 torch-2.1.0+cu118 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 3005843 parameters, 0 gradients, 8.1 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 2/2 [00:01<00:00,  1.82it/s]
                   all         56         58      0.998      0.966      0.994      0.889
Speed: 0.3ms preprocess, 2.6ms inference, 0.0ms loss, 2.6ms postprocess per image
Results saved to runs/detect/train
ultralytics.utils.metrics.DetMetrics object with attributes:
```

***Figure 6:*** *Model summary*

As shown above, the model in this project was trained within 100 epochs in 0.646 hours. 56 images were used for validation, in which 58 instances were recognized. The **P (Precision)** value is 0.998, which means more than 99.8% of detected car plate are correct. **R (Recall)** value is 0.966, presenting the model can correctly identify all relevant instances from the total actual positive instances. The **mAP50-95** value is 0.889, indicating that the model can correctly detect target objects at varying IoU(intersection over union) thresholds, ranging from 0.50 to 0.95.

Additionally, a series of detailed metrics were available in the "runs" directory.

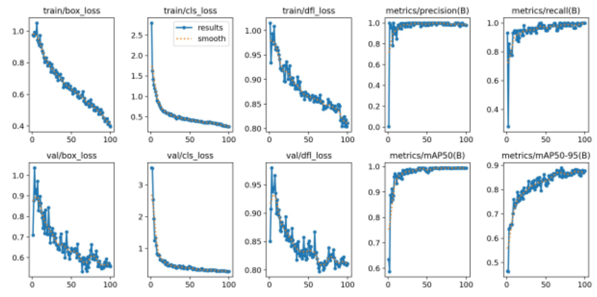


***Figure 7:*** *Metric against epoch*

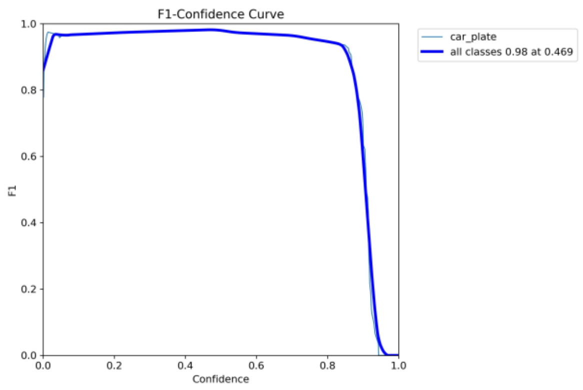


***Figure 8:*** *F1-Confidence*

F1 score, a metric that combines precision and recall into a single value, is close to 1 in the range of 0 and 0.8 confidence, indicating that within this range confidence the model has a good overall performance.

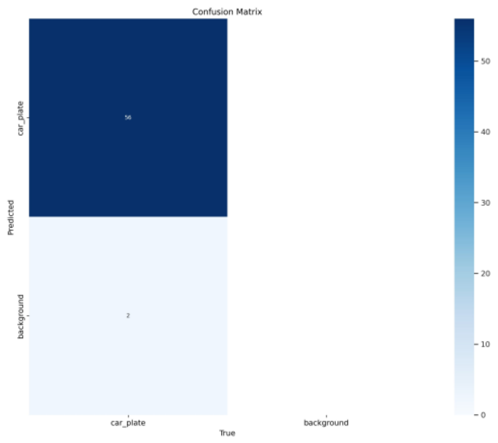


***Figure 9:*** *Confusion Matrix*

The confusion matrix presents the custom model correctly identified 56 car plates but misidentified 2 car plates as background.

## 7. Predict Using the Custom YOLO Model

After the custom model's training was completed, it was ready to predict images within the "pred" folders to present the model's practical application. The prediction results demonstrated that the custom model could identify and locate the position of car plates within the images.

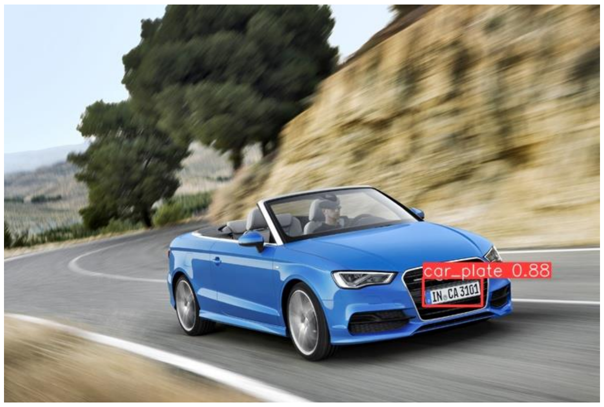


***Figure 10:*** *Prediction*

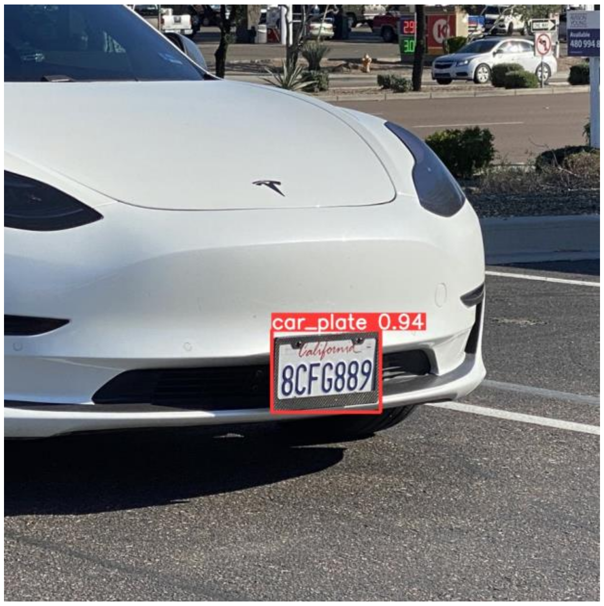


***Figure 11:*** *Prediction*

## 8. Pipeline to Train YOLO Model

A pipeline to efficiently train a custom YOLO model has been created. Please visit ***Appendix: Pipeline*** for more details.

## 9. Summary

The project introduces YOLOv8, an advanced object detection algorithm in computer vision, focusing on creating a custom model for car plate prediction.

This project went through:

- Data collection, a step that involved using a Google image downloader.

- Data processing, the step included cleansing and annotation using "labelImg."
- Creation of environment, utilizing virtual environment.
- Yolo model configuration, emphasizing the need for a specific dataset architecture.
- Training the custom YOLOv8 detector, involving monitoring key metrics
- Performance evaluation, concentrating the key indicators such as precision, recall, and mAP50-95.
- Car plate prediction with new images.
- And a streamlined pipeline for custom YOLO model training.

With this project, I learned:

- the steps involved in training custom YOLO model,
- data processing methods for image-based data,
- parameters used to train a YOLO model,
- and the indicators to evaluate a custom model.

## References

Glenn, J., Abirami, V. (2023, November 18). *Performance Metrics Deep Dive.* Ultralytics. https://docs.ultralytics.com/guides/yolo-performance-metrics/

Glenn, J. (2023, November 25). *Model Training with Ultralytics YOLO*. Ultralytics. https://docs.ultralytics.com/modes/train/

## Appendix: Glossary

**YOLO (You Only Look Once)**: an end-to-end neural network that makes predictions of bounding boxes and class probabilities all at once.

**simple-image-download:** a Python script to search for and download images from Google Images.

**LabelImg:** a Python graphical image annotation tool.

**Ultralytics**: a Python library for YOLOv8 model that builds upon the success of previous YOLO versions and introduces new features and improvements to further boost performance and flexibility.

**Epoch**: In deep learning, an epoch refers to one complete pass through the entire training dataset during the training phase of a neural network.

**Batch**: a batch refers to a subset of the training data that is used during one iteration of model training.

**Optimizer**: In the context of deep learning, an optimizer refers to an algorithm that adjusts the weights and biases of a neural network during training to minimize the error or loss function.

**Early stopping:** a method to halt the training process once the performance on a validation dataset stops improving or starts deteriorating.

**Patience**: the number of epochs to wait before early stop if no progress on the validation set.

**box_loss**: a measure of how well the model is predicting the positions and sizes of bounding boxes around objects in the training data, and it is one of the factors used to guide the optimization process during training.

**cls_loss**: classification loss, which is a component of the overall loss function used during the training of the model.

**Precision**: a metric that measures the accuracy of positive predictions made by a classification model.

**Recall**: a metric that measures the ability of a classification model to capture all the relevant instances of a particular class.