**Install And Import Essential Libreries**

*# Install Essential Libraries*

!pip install ultralytics

*# Import Essential Libraries*

import os

import random

import pandas as pd

from PIL import Image

import cv2

from ultralytics import YOLO

from IPython.display import Video

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

sns.set(style='darkgrid')

import pathlib

import glob

from tqdm.notebook import trange, tqdm

import warnings

warnings.filterwarnings('ignore')

*# Configure the visual appearance of Seaborn plots*

sns.set(rc={'axes.facecolor': '#eae8fa'}, style='darkgrid')

**Dataset**

**Show Some Images From TrainSet**

Image\_dir = '/kaggle/input/cardetection/car/train/images'

num\_samples = 9

image\_files = os.listdir(Image\_dir)

*# Randomly select num\_samples images*

rand\_images = random.sample(image\_files, num\_samples)

fig, axes = plt.subplots(3, 3, figsize=(11, 11))

for i **in** range(num\_samples):

image = rand\_images[i]

ax = axes[i // 3, i % 3]

ax.imshow(plt.imread(os.path.join(Image\_dir, image)))

ax.set\_title(f'Image **{**i+1**}**')

ax.axis('off')

plt.tight\_layout()

plt.show()

**Get Shape Of An Image For Using In Training Step**

*# Get the size of the image*

image = cv2.imread("/kaggle/input/cardetection/car/train/images/00000\_00000\_00012\_png.rf.23f94508dba03ef2f8bd187da2ec9c26.jpg")

h, w, c = image.shape

print(f"The image has dimensions **{**w**}**x**{**h**}** and **{**c**}** channels.")

**Try Pre-trained YOLOv8 For Detect Traffic Signs**

*# Use a pretrained YOLOv8n model*

model = YOLO("yolov8n.pt")

*# Use the model to detect object*

image = "/kaggle/input/cardetection/car/train/images/FisheyeCamera\_1\_00228\_png.rf.e7c43ee9b922f7b2327b8a00ccf46a4c.jpg"

result\_predict = model.predict(source = image, imgsz=(640))

*# show results*

plot = result\_predict[0].plot()

plot = cv2.cvtColor(plot, cv2.COLOR\_BGR2RGB)

display(Image.fromarray(plot))

**YOLOv8-Based Traffic Signs Detection Model**

**Training Step**

!pip install --upgrade ultralytics ray

*# Build from YAML and transfer weights*

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

*# Training The Final Model*

Result\_Final\_model = Final\_model.train(data="/kaggle/input/cardetection/car/data.yaml",epochs = 30, batch = -1, optimizer = 'auto')

**Validation Step**

import os

import cv2

import matplotlib.pyplot as plt

def display\_images(post\_training\_files\_path, image\_files):

for image\_file **in** image\_files:

image\_path = os.path.join(post\_training\_files\_path, image\_file)

img = cv2.imread(image\_path)

img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

plt.figure(figsize=(10, 10), dpi=120)

plt.imshow(img)

plt.axis('off')

plt.show()

*# List of image files to display*

image\_files = [

'confusion\_matrix\_normalized.png',

'F1\_curve.png',

'P\_curve.png',

'R\_curve.png',

'PR\_curve.png',

'results.png'

]

*# Path to the directory containing the images*

post\_training\_files\_path = '/kaggle/working/runs/detect/train'

*# Display the images*

display\_images(post\_training\_files\_path, image\_files)

Result\_Final\_model = pd.read\_csv('/kaggle/working/runs/detect/train/results.csv')

Result\_Final\_model.tail(10)

*# Read the results.csv file as a pandas dataframe*

Result\_Final\_model.columns = Result\_Final\_model.columns.str.strip()

*# Create subplots*

fig, axs = plt.subplots(nrows=5, ncols=2, figsize=(15, 15))

*# Plot the columns using seaborn*

sns.lineplot(x='epoch', y='train/box\_loss', data=Result\_Final\_model, ax=axs[0,0])

sns.lineplot(x='epoch', y='train/cls\_loss', data=Result\_Final\_model, ax=axs[0,1])

sns.lineplot(x='epoch', y='train/dfl\_loss', data=Result\_Final\_model, ax=axs[1,0])

sns.lineplot(x='epoch', y='metrics/precision(B)', data=Result\_Final\_model, ax=axs[1,1])

sns.lineplot(x='epoch', y='metrics/recall(B)', data=Result\_Final\_model, ax=axs[2,0])

sns.lineplot(x='epoch', y='metrics/mAP50(B)', data=Result\_Final\_model, ax=axs[2,1])

sns.lineplot(x='epoch', y='metrics/mAP50-95(B)', data=Result\_Final\_model, ax=axs[3,0])

sns.lineplot(x='epoch', y='val/box\_loss', data=Result\_Final\_model, ax=axs[3,1])

sns.lineplot(x='epoch', y='val/cls\_loss', data=Result\_Final\_model, ax=axs[4,0])

sns.lineplot(x='epoch', y='val/dfl\_loss', data=Result\_Final\_model, ax=axs[4,1])

*# Set titles and axis labels for each subplot*

axs[0,0].set(title='Train Box Loss')

axs[0,1].set(title='Train Class Loss')

axs[1,0].set(title='Train DFL Loss')

axs[1,1].set(title='Metrics Precision (B)')

axs[2,0].set(title='Metrics Recall (B)')

axs[2,1].set(title='Metrics mAP50 (B)')

axs[3,0].set(title='Metrics mAP50-95 (B)')

axs[3,1].set(title='Validation Box Loss')

axs[4,0].set(title='Validation Class Loss')

axs[4,1].set(title='Validation DFL Loss')

plt.suptitle('Training Metrics and Loss', fontsize=24)

plt.subplots\_adjust(top=0.8)

plt.tight\_layout()

plt.show()

**Validation of the model by testset**

*# Loading the best performing model*

Valid\_model = YOLO('/kaggle/working/runs/detect/train/weights/best.pt')

*# Evaluating the model on the validset*

metrics = Valid\_model.val(split = 'val')

*# final results*

print("precision(B): ", metrics.results\_dict["metrics/precision(B)"])

print("metrics/recall(B): ", metrics.results\_dict["metrics/recall(B)"])

print("metrics/mAP50(B): ", metrics.results\_dict["metrics/mAP50(B)"])

print("metrics/mAP50-95(B): ", metrics.results\_dict["metrics/mAP50-95(B)"])

**Making prediction on test images**

*# Normalization function*

def normalize\_image(image):

return image / 255.0

*# Image resizing function*

def resize\_image(image, size=(640, 640)):

return cv2.resize(image, size)

*# Path to validation images*

dataset\_path = '/kaggle/input/cardetection/car' *# Place your dataset path here*

valid\_images\_path = os.path.join(dataset\_path, 'test', 'images')

*# List of all jpg images in the directory*

image\_files = [file for file **in** os.listdir(valid\_images\_path) if file.endswith('.jpg')]

*# Check if there are images in the directory*

if len(image\_files) > 0:

*# Select 9 images at equal intervals*

num\_images = len(image\_files)

step\_size = max(1, num\_images // 9) *# Ensure the interval is at least 1*

selected\_images = [image\_files[i] for i **in** range(0, num\_images, step\_size)]

*# Prepare subplots*

fig, axes = plt.subplots(3, 3, figsize=(20, 21))

fig.suptitle('Validation Set Inferences', fontsize=24)

for i, ax **in** enumerate(axes.flatten()):

if i < len(selected\_images):

image\_path = os.path.join(valid\_images\_path, selected\_images[i])

*# Load image*

image = cv2.imread(image\_path)

*# Check if the image is loaded correctly*

if image **is** **not** None:

*# Resize image*

resized\_image = resize\_image(image, size=(640, 640))

*# Normalize image*

normalized\_image = normalize\_image(resized\_image)

*# Convert the normalized image to uint8 data type*

normalized\_image\_uint8 = (normalized\_image \* 255).astype(np.uint8)

*# Predict with the model*

results = Valid\_model.predict(source=normalized\_image\_uint8, imgsz=640, conf=0.5)

*# Plot image with labels*

annotated\_image = results[0].plot(line\_width=1)

annotated\_image\_rgb = cv2.cvtColor(annotated\_image, cv2.COLOR\_BGR2RGB)

ax.imshow(annotated\_image\_rgb)

else:

print(f"Failed to load image **{**image\_path**}**")

ax.axis('off')

plt.tight\_layout()

plt.show()

**Try Pre-trained YOLOv8 For Detect Traffic Signs From Video**

*# Convert mp4*

!ffmpeg -y -loglevel panic -i /kaggle/input/cardetection/video.mp4 output.mp4## <b>6 <span style='color:#e61227'>|</span> Export The Final Model Of Detect Traffic Signs </b>

\*\*Tip:\*\* The ultimate goal of training a model **is** to deploy it for real-world applications. Export mode **in** Ultralytics YOLOv8 offers a versatile range of options for exporting your trained model to different formats, making it deployable across various platforms **and** devices.

*# Display the video*

Video("output.mp4", width=960)

*# Use the model to detect signs*

Valid\_model.predict(source="/kaggle/input/cardetection/video.mp4", show=True,save = True)

*# show result*

*# Convert format*

!ffmpeg -y -loglevel panic -i /kaggle/working/runs/detect/predict/video.avi result\_out.mp4

*# Display the video*

Video("result\_out.mp4", width=960)