# Anomaly Detection in Traffic Signs recognition

This project aims to enhance the effectiveness of traffic sign recognition systems by incorporating anomaly detection using YOLOv8 and autoencoders.

The primary goal is to identify and flag anomalies within traffic signs, such as damaged or incorrectly placed signs, to improve road safety and the reliability of automated driving systems.

By leveraging advanced computer vision techniques, this project seeks to contribute to more accurate and robust traffic sign recognition, ultimately enhancing the safety and efficiency of transportation systems.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

## Object Detection using YOLOV8

## Libraries

```
import cv2
import glob
import random
import os
import yaml

import matplotlib.pyplot as plt
import matplotlib.image as mpimg

from collections import defaultdict
import plotly.express as px
from shutil import copyfile
from PIL import Image
import numpy as np
```

```
!pip install ultralytics
```

```
Collecting ultralytics
  Downloading ultralytics-8.0.183-py3-none-any.whl (618 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 618.1/618.1 kB 7.8 MB/s eta 0:00:00
Requirement already satisfied: matplotlib>=3.3.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (3.7.1)
Requirement already satisfied: numpy>=1.22.2 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (1.23.5)
Requirement already satisfied: opencv-python>=4.6.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (4.8.0.76)
Requirement already satisfied: pillow>=7.1.2 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (9.4.0)
Requirement already satisfied: pyyaml>=5.3.1 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (6.0.1)
Requirement already satisfied: requests>=2.23.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (2.31.0)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (1.11.2)
Requirement already satisfied: torch>=1.8.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (2.0.1+cu118)
Requirement already satisfied: torchvision>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (0.15.2+cu118)
Requirement already satisfied: tqdm>=4.64.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (4.66.1)
Requirement already satisfied: pandas>=1.1.4 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (1.5.3)
Requirement already satisfied: seaborn>=0.11.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (0.12.2)
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages (from ultralytics) (5.9.5)
Requirement already satisfied: py-cpuinfo in /usr/local/lib/python3.10/dist-packages (from ultralytics) (9.0.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (1
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (0.11.
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (23
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (3
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.1.4->ultralytics) (2023.3.po
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.23.0->ultralyt
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.23.0->ultralytics) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.23.0->ultralytics) (
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.23.0->ultralytics) (
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->ultralytics) (3.12.2)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->ultralytics) (4.5.0
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->ultralytics) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->ultralytics) (3.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->ultralytics) (3.1.2)
Requirement already satisfied: triton==2.0.0 in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->ultralytics) (2.0.0)
Requirement already satisfied: cmake in /usr/local/lib/python3.10/dist-packages (from triton==2.0.0->torch>=1.8.0->ultralytics) (3.
Requirement already satisfied: lit in /usr/local/lib/python3.10/dist-packages (from triton==2.0.0->torch>=1.8.0->ultralytics) (16.0
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib>=3.3.0->u
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch>=1.8.0->ultralytics)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch>=1.8.0->ultralytics) (1.3
```

```
Installing collected packages: ultralytics
Successfully installed ultralytics-8.0.183
```

```
# Function to convert bounding boxes in YOLO format to xmin, ymin, xmax, ymax.
def yolo2bbox(bboxes):
    xmin, ymin = bboxes[0]-bboxes[2]/2, bboxes[1]-bboxes[3]/2
    xmax, ymax = bboxes[0]+bboxes[2]/2, bboxes[1]+bboxes[3]/2
    return xmin, ymin, xmax, ymax


def plot_box(image, bboxes, labels):
    # Need the image height and width to denormalize
    # the bounding box coordinates
    h, w, _ = image.shape
    for box_num, box in enumerate(bboxes):
        x1, y1, x2, y2 = yolo2bbox(box)
        # Denormalize the coordinates.
        xmin = int(x1*w)
        ymin = int(y1*h)
        xmax = int(x2*w)
        ymax = int(y2*h)

        thickness = max(2, int(w/275))

        cv2.rectangle(
            image,
            (xmin, ymin), (xmax, ymax),
            color=(0, 0, 255),
            thickness=thickness
        )
    return image


# Function to plot images with the bounding boxes.
def plot(image_paths, label_paths, num_samples):
    all_images = []
    all_images.extend(glob.glob(image_paths+'/*.jpg'))
    all_images.extend(glob.glob(image_paths+'/*.JPG'))

    all_images.sort()

    num_images = len(all_images)

    plt.figure(figsize=(15, 12))
    for i in range(num_samples):
        j = random.randint(0,num_images-1)
        image_name = all_images[j]
        image_name = '.'.join(image_name.split(os.path.sep)[-1].split('.')[:-1])
        image = cv2.imread(all_images[j])
        with open(os.path.join(label_paths, image_name+'.txt'), 'r') as f:
            bboxes = []
            labels = []
            label_lines = f.readlines()
            for label_line in label_lines:
                label = label_line[0]
                bbox_string = label_line[2:]
                x_c, y_c, w, h = bbox_string.split(' ')
                x_c = float(x_c)
                y_c = float(y_c)
                w = float(w)
                h = float(h)
                bboxes.append([x_c, y_c, w, h])
                labels.append(label)
        result_image = plot_box(image, bboxes, labels)
        plt.subplot(2, 2, i+1)
        plt.imshow(result_image[:, :, ::-1])
        plt.axis('off')
    plt.subplots_adjust(wspace=1)
    plt.tight_layout()
    plt.show()


%%writefile traffic_signs.yaml

path: /content/drive/MyDrive/final_dataset/
train: train/images
val: test/images

# class names
names:
  - 'no_stopping'
```

```
  - 'eighty'
  - 'fifteen'
  - 'fifty'
  - 'five'
  - 'forty'
  - 'seventy'
  - 'sixty'
  - 'thirty'
  - 'twenty'


     Writing traffic_signs.yaml
```

```
# Directory paths for images and labels
data_dir = '/content/drive/MyDrive/final_dataset/'
image_dir_train = os.path.join(data_dir, 'train/images')
label_dir_train = os.path.join(data_dir, 'train/labels')

# Read class names from the YAML file
class_names_file = '/content/traffic_signs.yaml'
with open(class_names_file, 'r') as yaml_file:
    class_data = yaml.safe_load(yaml_file)
    class_names = class_data['names']

# Create a dictionary to store class counts
class_samples = defaultdict(list)

# Iterate through the label files for training images to collect one sample image per class
for label_filename in os.listdir(label_dir_train):
    label_file = os.path.join(label_dir_train, label_filename)
    if os.path.exists(label_file):
        with open(label_file, 'r') as label_f:
            for line in label_f:
                label_values = line.strip().split()
                if len(label_values) == 5:  # Check if the line has bounding box information
                    class_index = int(label_values[0])
                    image_filename = os.path.splitext(label_filename)[0] + '.jpg'
                    class_samples[class_index].append(image_filename)

# Determine how many classes to display in each row
max_classes_per_row = 5
num_classes = len(class_samples)
num_samples_per_class = 1

# Calculate the number of rows needed
num_rows = (num_classes + max_classes_per_row - 1) // max_classes_per_row

# Create subplots with the determined number of rows and columns
fig, axs = plt.subplots(num_rows, max_classes_per_row, figsize=(15, 3 * num_rows))

for class_index, image_list in class_samples.items():
    row = class_index // max_classes_per_row
    col = class_index % max_classes_per_row

    for sample_index in range(num_samples_per_class):
        if sample_index < len(image_list):
            image_filename = random.choice(image_list)  # Randomly select a sample image from the class
            image_path = os.path.join(image_dir_train, image_filename)
            img = plt.imread(image_path)

            # Display the image in the corresponding subplot
            axs[row, col].imshow(img)
            axs[row, col].set_title(class_names[class_index])
            axs[row, col].axis('off')

# Remove empty subplots
for i in range(num_classes, num_rows * max_classes_per_row):
    axs.flatten()[i].axis('off')

plt.tight_layout()
plt.show()
```

```
# Iterate through the label files for training images to collect one sample image per class
for label_filename in os.listdir(label_dir_train):
    label_file = os.path.join(label_dir_train, label_filename)
    if os.path.exists(label_file):
        with open(label_file, 'r') as label_f:
            for line in label_f:
                label_values = line.strip().split()
```

```
                if len(label_values) == 5:  # Check if the line has bounding box information
                    class_index = int(label_values[0])
                    image_filename = os.path.splitext(label_filename)[0] + '.jpg'
                    class_samples[class_index].append(image_filename)

# Determine how many classes to display in each row
max_classes_per_row = 5
num_classes = len(class_samples)
num_samples_per_class = 1  # You can change this to display more than one image per class

# Calculate the number of rows needed
num_rows = (num_classes + max_classes_per_row - 1) // max_classes_per_row

# Create subplots with the determined number of rows and columns
fig, axs = plt.subplots(num_rows, max_classes_per_row, figsize=(15, 3 * num_rows))

for class_index, image_list in class_samples.items():
    row = class_index // max_classes_per_row
    col = class_index % max_classes_per_row

    for sample_index in range(num_samples_per_class):
        if sample_index < len(image_list):
            image_filename = random.choice(image_list)  # Randomly select a sample image from the class
            image_path = os.path.join(image_dir_train, image_filename)
            img = plt.imread(image_path)

            # Display the image in the corresponding subplot
            axs[row, col].imshow(img)
            axs[row, col].set_title(class_names[class_index])
            axs[row, col].axis('off')

# Remove empty subplots
for i in range(num_classes, num_rows * max_classes_per_row):
    axs.flatten()[i].axis('off')

plt.tight_layout()
plt.show()
```

```
# Define the path to the directory containing the images
train_dir = '/content/drive/MyDrive/final_dataset/train/images'
valid_dir = '/content/drive/MyDrive/final_dataset/valid/images'
test_dir = '/content/drive/MyDrive/final_dataset/test/images'
# Use os.listdir to get the list of files in the directory
image_files1 = os.listdir(train_dir)
image_files2 = os.listdir(valid_dir)
image_files3 = os.listdir(test_dir)
# Use len() to count the number of image files
num_images1 = len(image_files1)
num_images2 = len(image_files2)
num_images3 = len(image_files3)

# Print the count
print(f"Number of images in Training set: {num_images1}")
print(f"Number of images in Validation set: {num_images2}")
print(f"Number of images in Testing set: {num_images3}")
labels = ["Training", "Validation", "Testing"]
sizes = [num_images1, num_images2, num_images3]
colors = ['gold', 'yellowgreen', 'lightcoral']
explode = (0.1, 0, 0)  # explode the 1st slice (train_dir)

plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True, startangle=140)
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

plt.title('Distribution of Images in Each Directory')
plt.show()
```

```
# Directory paths for images and labels
data_dir = '/content/drive/MyDrive/final_dataset/'
image_dir_train = os.path.join(data_dir, 'train/images')
image_dir_val = os.path.join(data_dir, 'valid/images')
image_dir_test = os.path.join(data_dir, 'test/images')
label_dir_train = os.path.join(data_dir, 'train/labels')
label_dir_val = os.path.join(data_dir, 'valid/labels')
label_dir_test = os.path.join(data_dir, 'test/labels')

# Read class names from the YAML file
class_names_file = '/content/traffic_signs.yaml'
with open(class_names_file, 'r') as yaml_file:
```

```
        class_data = yaml.safe_load(yaml_file)
        class_names = class_data['names']

    # Create dictionaries to store class counts for training, validation, and testing separately
    class_counts_train = defaultdict(int)
    class_counts_val = defaultdict(int)
    class_counts_test = defaultdict(int)

    # Iterate through the label files for training images
    for label_filename in os.listdir(label_dir_train):
        label_file = os.path.join(label_dir_train, label_filename)
        if os.path.exists(label_file):
            with open(label_file, 'r') as label_f:
                for line in label_f:
                    label_values = line.strip().split()
                    if len(label_values) == 5:  # Check if the line has bounding box information
                        class_index = int(label_values[0])
                        class_counts_train[class_names[class_index]] += 1

    # Iterate through the label files for validation images
    for label_filename in os.listdir(label_dir_val):
        label_file = os.path.join(label_dir_val, label_filename)
        if os.path.exists(label_file):
            with open(label_file, 'r') as label_f:
                for line in label_f:
                    label_values = line.strip().split()
                    if len(label_values) == 5:  # Check if the line has bounding box information
                        class_index = int(label_values[0])
                        class_counts_val[class_names[class_index]] += 1

    # Iterate through the label files for testing images
    for label_filename in os.listdir(label_dir_test):
        label_file = os.path.join(label_dir_test, label_filename)
        if os.path.exists(label_file):
            with open(label_file, 'r') as label_f:
                for line in label_f:
                    label_values = line.strip().split()
                    if len(label_values) == 5:  # Check if the line has bounding box information
                        class_index = int(label_values[0])
                        class_counts_test[class_names[class_index]] += 1

    # Extract class names and counts for training, validation, and testing
    classes_train = list(class_counts_train.keys())
    counts_train = list(class_counts_train.values())
    classes_val = list(class_counts_val.keys())
    counts_val = list(class_counts_val.values())
    classes_test = list(class_counts_test.keys())
    counts_test = list(class_counts_test.values())

    # Create separate bar charts to visualize the class distribution for training, validation, and testing
    plt.figure(figsize=(18, 6))


    # Training set
    plt.subplot(1, 3, 1)
    plt.barh(classes_train, counts_train, color='skyblue')
    plt.xlabel('Number of Images')
    plt.ylabel('Class Names')
    plt.title('Class Distribution in Training Set')
    plt.gca().invert_yaxis()

    # Add labels with counts to the bars in the training set
    for i, v in enumerate(counts_train):
        plt.text(v, i, str(v), va='center', color='black', fontsize=8)

    # Validation set
    plt.subplot(1, 3, 2)
    plt.barh(classes_val, counts_val, color='lightcoral')
    plt.xlabel('Number of Images')
    plt.ylabel('Class Names')
    plt.title('Class Distribution in Validation Set')
    plt.gca().invert_yaxis()

    for i, v in enumerate(counts_val):
        plt.text(v, i, str(v), va='center', color='black', fontsize=8)

    # Testing set
    plt.subplot(1, 3, 3)
    plt.barh(classes_test, counts_test, color='lightgreen')
    plt.xlabel('Number of Images')
    plt.ylabel('Class Names')
    plt.title('Class Distribution in Testing Set')
```

```
plt.gca().invert_yaxis()

for i, v in enumerate(counts_test):
    plt.text(v, i, str(v), va='center', color='black', fontsize=8)

plt.tight_layout()
plt.show()



import os
import random
import yaml
from collections import defaultdict
import plotly.express as px

# Directory paths for images and labels
data_dir = '/content/drive/MyDrive/final_dataset/'
image_dir_train = os.path.join(data_dir, 'train/images')
image_dir_val = os.path.join(data_dir, 'test/images')
label_dir_train = os.path.join(data_dir, 'train/labels')
label_dir_val = os.path.join(data_dir, 'test/labels')

# Read class names from the YAML file
class_names_file = '/content/traffic_signs.yaml'
with open(class_names_file, 'r') as yaml_file:
    class_data = yaml.safe_load(yaml_file)
    class_names = class_data['names']

# Create a dictionary to store class counts
class_counts = defaultdict(int)

# Iterate through the label files for validation images
for label_filename in os.listdir(label_dir_val):
    label_file = os.path.join(label_dir_val, label_filename)
    if os.path.exists(label_file):
        with open(label_file, 'r') as label_f:
            for line in label_f:
                label_values = line.strip().split()
                if len(label_values) == 5:  # Check if the line has bounding box information
                    class_index = int(label_values[0])
                    class_counts[class_names[class_index]] += 1

# Convert class_counts to a DataFrame for plotly
class_counts_df = [{'class': class_name, 'count': count} for class_name, count in class_counts.items()]

# Create an interactive treemap
fig = px.treemap(class_counts_df, path=['class'], values='count', title='Class Distribution Treemap')
fig.update_traces(textinfo='label+value')
fig.show()
```

## Training phase

```
EPOCHS = 300
!yolo task=detect mode=train model=yolov8n.pt imgsz=128 data=traffic_signs.yaml epochs={EPOCHS} batch=64 name=yolov8n_v8_50e
```

```
Downloading https://github.com/ultralytics/assets/releases/download/v0.0.0/yolov8n.pt to 'yolov8n.pt'...
100% 6.23M/6.23M [00:00<00:00, 129MB/s]
Ultralytics YOLOv8.0.183 🚀 Python-3.10.12 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)
engine/trainer: task=detect, mode=train, model=yolov8n.pt, data=traffic_signs.yaml, epochs=300, patience=50, batch=64, imgsz=128
Downloading https://ultralytics.com/assets/Arial.ttf to '/root/.config/Ultralytics/Arial.ttf'...
100% 755k/755k [00:00<00:00, 22.8MB/s]
Overriding model.yaml nc=80 with nc=10

                   from  n    params  module                                       arguments
  0                  -1  1       464  ultralytics.nn.modules.conv.Conv             [3, 16, 3, 2]
  1                  -1  1      4672  ultralytics.nn.modules.conv.Conv             [16, 32, 3, 2]
  2                  -1  1      7360  ultralytics.nn.modules.block.C2f             [32, 32, 1, True]
  3                  -1  1     18560  ultralytics.nn.modules.conv.Conv             [32, 64, 3, 2]
  4                  -1  2     49664  ultralytics.nn.modules.block.C2f             [64, 64, 2, True]
  5                  -1  1     73984  ultralytics.nn.modules.conv.Conv             [64, 128, 3, 2]
  6                  -1  2    197632  ultralytics.nn.modules.block.C2f             [128, 128, 2, True]
  7                  -1  1    295424  ultralytics.nn.modules.conv.Conv             [128, 256, 3, 2]
  8                  -1  1    460288  ultralytics.nn.modules.block.C2f             [256, 256, 1, True]
  9                  -1  1    164608  ultralytics.nn.modules.block.SPPF            [256, 256, 5]
 10                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 11             [-1, 6]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 12                  -1  1    148224  ultralytics.nn.modules.block.C2f             [384, 128, 1]
 13                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 14             [-1, 4]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 15                  -1  1     37248  ultralytics.nn.modules.block.C2f             [192, 64, 1]
```

```
 16                -1  1     36992  ultralytics.nn.modules.conv.Conv             [64, 64, 3, 2]
 17           [-1, 12]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 18                -1  1    123648  ultralytics.nn.modules.block.C2f             [192, 128, 1]
 19                -1  1    147712  ultralytics.nn.modules.conv.Conv             [128, 128, 3, 2]
 20            [-1, 9]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 21                -1  1    493056  ultralytics.nn.modules.block.C2f             [384, 256, 1]
 22       [15, 18, 21]  1    753262  ultralytics.nn.modules.head.Detect           [10, [64, 128, 256]]
Model summary: 225 layers, 3012798 parameters, 3012782 gradients

Transferred 319/355 items from pretrained weights
TensorBoard: Start with 'tensorboard --logdir runs/detect/yolov8n_v8_50e', view at http://localhost:6006/
Freezing layer 'model.22.dfl.conv.weight'
AMP: running Automatic Mixed Precision (AMP) checks with YOLOv8n...
AMP: checks passed ✅
train: Scanning /content/drive/MyDrive/final_dataset/train/labels.cache... 800 images, 0 backgrounds, 0 corrupt: 100% 800/800 [0(
albumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)), ToGray(p=0.01), CLAHE(p=0.01, clip_limit
val: Scanning /content/drive/MyDrive/final_dataset/test/labels.cache... 230 images, 0 backgrounds, 0 corrupt: 100% 230/230 [00:0(
Plotting labels to runs/detect/yolov8n_v8_50e/labels.jpg...
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' and 'momentur
optimizer: AdamW(lr=0.000714, momentum=0.9) with parameter groups 57 weight(decay=0.0), 64 weight(decay=0.0005), 63 bias(decay=0
Image sizes 128 train, 128 val
Using 2 dataloader workers
Logging results to runs/detect/yolov8n_v8_50e
Starting training for 300 epochs...

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      1/300     0.688G     0.8706      4.195       1.17         80        128: 100% 13/13 [02:27<00:00, 11.33s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:04<00:00,  2.08s/it]
                   all        230        231    0.00722      0.792     0.0955     0.0741

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
```

```
import matplotlib.pyplot as plt

# Number of epochs and corresponding mAP values
epochs = [100, 200, 226]
map_values = [0.868, 0.884, 0.899]

# Create a line graph to visualize mAP vs. number of epochs
plt.figure(figsize=(8, 6))
plt.plot(epochs, map_values, marker='o', linestyle='-')
plt.title('mAP vs. Number of Epochs')
plt.xlabel('Number of Epochs')
plt.ylabel('mAP')
plt.grid(True)
plt.xticks(epochs)
plt.show()
```

```
!cp runs/detect/yolov8n_v8_50e/weights/best.pt /content/drive/MyDrive/final_dataset/yolov8_model.pt
```

## Evaluation phase

```
!yolo task=detect mode=val model=/content/drive/MyDrive/final_dataset/yolov8_model.pt name=yolov8n_eval data=traffic_signs.yaml
```

```
Ultralytics YOLOv8.0.183 🚀 Python-3.10.12 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 3007598 parameters, 0 gradients
Downloading https://ultralytics.com/assets/Arial.ttf to '/root/.config/Ultralytics/Arial.ttf'...
100% 755k/755k [00:00<00:00, 12.5MB/s]
val: Scanning /content/drive/MyDrive/final_dataset/test/labels.cache... 230 images, 0 backgrounds, 0 corrupt: 100% 230/230 [00:00<?
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 15/15 [00:04<00:00,  3.68it/s]
                   all        230        231      0.85      0.875      0.899      0.861
           no_stopping        230          2          1      0.924      0.995      0.895
                eighty        230         12       0.95      0.833      0.937      0.928
               fifteen        230         10       0.87          1      0.995      0.995
                 fifty        230         22      0.798      0.955      0.952      0.938
                  five        230         30      0.963      0.867      0.942      0.835
                 forty        230         33      0.777      0.879       0.89      0.864
               seventy        230         29      0.921      0.809      0.855      0.761
                 sixty        230         28      0.805      0.592      0.655      0.635
                thirty        230         52      0.624      0.895      0.793      0.788
                twenty        230         13      0.788          1      0.974      0.974
Speed: 0.2ms preprocess, 3.2ms inference, 0.0ms loss, 3.1ms postprocess per image
Results saved to runs/detect/yolov8n_eval
💡 Learn more at https://docs.ultralytics.com/modes/val
```

## Inference phase

```
!yolo task=detect \
mode=predict \
model=/content/drive/MyDrive/final_dataset/yolov8_model.pt\
source="/content/drive/MyDrive/final_dataset/test/images" \
imgsz=128 \
name=yolov8n_v8_50e_infer1280 \
show_labels=True \
save_txt=True \
save_conf=True
```

```
Ultralytics YOLOv8.0.183 🚀 Python-3.10.12 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 3007598 parameters, 0 gradients

image 1/230 /content/drive/MyDrive/final_dataset/test/images/00001_00001_00012_png_jpg.rf.a828911891672abce3d764e07d75d852.jpg:
image 2/230 /content/drive/MyDrive/final_dataset/test/images/000_0001_png.rf.8e1f9b5af2ee0dd06fa23de276aa2027.jpg: 128x128 1 fiv
image 3/230 /content/drive/MyDrive/final_dataset/test/images/000_0001_png_jpg.rf.959b4f70b4089b81ad8d1d03c1694542.jpg: 128x128 1
image 4/230 /content/drive/MyDrive/final_dataset/test/images/000_0029_png.rf.6bee78a5cd93e8b68801b6592eefe7ee.jpg: 128x128 1 no_
image 5/230 /content/drive/MyDrive/final_dataset/test/images/000_1_0005_png.rf.f45d9bfbb89f1935d08b8f2e10fd19f8.jpg: 128x128 1 f
image 6/230 /content/drive/MyDrive/final_dataset/test/images/000_1_0020_png.rf.d24ff5f4ab2068f2ead6cdb8e11160c0.jpg: 128x128 1 f
image 7/230 /content/drive/MyDrive/final_dataset/test/images/000_1_0023_png.rf.21cf4753cd40320976367b02b2351ddc.jpg: 128x128 1 f
image 8/230 /content/drive/MyDrive/final_dataset/test/images/000_1_0054_png.rf.f5f6f5027fc2d99c4a690564f0b93eb0.jpg: 128x128 2 f
image 9/230 /content/drive/MyDrive/final_dataset/test/images/001_0001_png.rf.4365b90a8049274ef988cf76026f212e.jpg: 128x128 1 fif
image 10/230 /content/drive/MyDrive/final_dataset/test/images/001_1_0004_png.rf.86b5c5d0187821a9ba3d9d128ab31eaa.jpg: 128x128 1
image 11/230 /content/drive/MyDrive/final_dataset/test/images/001_1_0005_png.rf.357e0877864e7090b90ed7fdd6eafacd.jpg: 128x128 1
image 12/230 /content/drive/MyDrive/final_dataset/test/images/001_1_0009_png.rf.a96fabe46d693a462d96e14ecbb2dda0.jpg: 128x128 1
image 13/230 /content/drive/MyDrive/final_dataset/test/images/001_1_0010_png.rf.c7f2b45456d075be0e7f91fd5f822604.jpg: 128x128 1
image 14/230 /content/drive/MyDrive/final_dataset/test/images/001_1_0013_png.rf.bfddf14c2ab63859e88ec3f324b25328.jpg: 128x128 1
image 15/230 /content/drive/MyDrive/final_dataset/test/images/001_1_0016_png.rf.2002d3f72255e50eb891a3e316d11775.jpg: 128x128 1
image 16/230 /content/drive/MyDrive/final_dataset/test/images/003_0121_png.rf.aeb83a508e058a96aa7e9e5934d42718.jpg: 128x128 1 fo
image 17/230 /content/drive/MyDrive/final_dataset/test/images/003_1_0021_png.rf.bd4c6c8d82a2ca1072cc5646bacbf9f9.jpg: 128x128 1
image 18/230 /content/drive/MyDrive/final_dataset/test/images/003_1_0023_png.rf.4d90041240a95951328badee8b0ac684.jpg: 128x128 1
image 19/230 /content/drive/MyDrive/final_dataset/test/images/003_1_0030_png.rf.ff217d472ace6610ca46eb5d64c6537d.jpg: 128x128 1
image 20/230 /content/drive/MyDrive/final_dataset/test/images/003_1_0040_png.rf.55e7fe982e8aa2a2bd4beb8266a0df91.jpg: 128x128 1
image 21/230 /content/drive/MyDrive/final_dataset/test/images/003_1_0041_png.rf.4754947c4368564d8a5b3cf4c149ede1.jpg: 128x128 1
image 22/230 /content/drive/MyDrive/final_dataset/test/images/003_1_0045_png.rf.ae5e7e670ef80cb3d7dee3ce923dfc0c.jpg: 128x128 1
image 23/230 /content/drive/MyDrive/final_dataset/test/images/003_1_0056_png.rf.2694277de8f9328146e2fbc4af415917.jpg: 128x128 1
image 24/230 /content/drive/MyDrive/final_dataset/test/images/003_1_0057_png.rf.9f7d6b0137ec354c3273637fa3fdda03.jpg: 128x128 1
image 25/230 /content/drive/MyDrive/final_dataset/test/images/003_1_0103_png.rf.53ec85f99e64e09ddd6c1ba201c903fc.jpg: 128x128 1
image 26/230 /content/drive/MyDrive/final_dataset/test/images/004_1_0011_png.rf.d9e6aad8b1f6f3c2698d14879672ca67.jpg: 128x128 1
image 27/230 /content/drive/MyDrive/final_dataset/test/images/004_1_0014_png.rf.4c2d339b2470ef9c1b026f8566909877.jpg: 128x128 1
image 28/230 /content/drive/MyDrive/final_dataset/test/images/004_1_0015_png.rf.427d8dcfb1589b2bbbf383997731b557.jpg: 128x128 1
image 29/230 /content/drive/MyDrive/final_dataset/test/images/004_1_0027_png.rf.1b45b4ebad8125e4ff6c0d2c477ccb0c.jpg: 128x128 1
image 30/230 /content/drive/MyDrive/final_dataset/test/images/004_1_0028_png.rf.8ae8dc86841e01cd6eccf3716048009d.jpg: 128x128 1
image 31/230 /content/drive/MyDrive/final_dataset/test/images/004_1_0033_png.rf.5b98fd7bd697ebe37c696409cac73f01.jpg: 128x128 1
image 32/230 /content/drive/MyDrive/final_dataset/test/images/005_0011_png.rf.ba910d93b377536755460bb4d6372d0b.jpg: 128x128 1 si
image 33/230 /content/drive/MyDrive/final_dataset/test/images/005_0073_png.rf.7e9a857393cd3a093a47ff111890343e.jpg: 128x128 1 si
image 34/230 /content/drive/MyDrive/final_dataset/test/images/005_1_0003_png.rf.e2c2d5d1e9f83685a3a01e2ff731a2d6.jpg: 128x128 1
image 35/230 /content/drive/MyDrive/final_dataset/test/images/005_1_0027_png.rf.fc9381dbdc274159566dc7e37566b751.jpg: 128x128 1
image 36/230 /content/drive/MyDrive/final_dataset/test/images/005_1_0032_png.rf.c7a5322676fd1038b63d0873ac7aeed8.jpg: 128x128 1
image 37/230 /content/drive/MyDrive/final_dataset/test/images/005_1_0057_png.rf.2c56d9ea9c0c215793f7a8701bfd1ecc.jpg: 128x128 1
image 38/230 /content/drive/MyDrive/final_dataset/test/images/006_1_0015_png.rf.ea39d6e8e9a16a327a3151632884518e.jpg: 128x128 1
image 39/230 /content/drive/MyDrive/final_dataset/test/images/006_1_0016_png.rf.6918188bb5c37bf9008f408bb1f783d6.jpg: 128x128 1
image 40/230 /content/drive/MyDrive/final_dataset/test/images/006_1_0033_png.rf.7401e478358ea9050115ce6ca47ed4cb.jpg: 128x128 1
image 41/230 /content/drive/MyDrive/final_dataset/test/images/006_1_0034_png.rf.aa89b41e5363d6ef22c75eb6dad38ad3.jpg: 128x128 1
image 42/230 /content/drive/MyDrive/final_dataset/test/images/006_1_0036_png.rf.17888b3607535c5143198dfaf66e0445.jpg: 128x128 1
image 43/230 /content/drive/MyDrive/final_dataset/test/images/007_0004_png.rf.6b1fe382e7357c51af8e3fef03be1e68.jpg: 128x128 1 ei
image 44/230 /content/drive/MyDrive/final_dataset/test/images/007_0057_png.rf.b8a62e2d4c491075da04aac268ea73c9.jpg: 128x128 1 ei
image 45/230 /content/drive/MyDrive/final_dataset/test/images/007_1_0026_1_j_png.rf.23bc25b2e36a78c1dea9eb7d84f6c112.jpg: 128x12
image 46/230 /content/drive/MyDrive/final_dataset/test/images/007_1_0029_png.rf.39241cac3dc6672cbae93c45ceddc038.jpg: 128x128 1
image 47/230 /content/drive/MyDrive/final_dataset/test/images/007_1_0032_png.rf.4085780a49a44e4cff51ef5e02b078ac.jpg: 128x128 1
image 48/230 /content/drive/MyDrive/final_dataset/test/images/007_1_0040_png.rf.7995c391876ce20c344409c49507287b.jpg: 128x128 1
image 49/230 /content/drive/MyDrive/final_dataset/test/images/007_1_0042_png.rf.f3016385e0547b4df5de99d2b0f0ab6a.jpg: 128x128 1
image 50/230 /content/drive/MyDrive/final_dataset/test/images/007_1_0048_png.rf.7e2244a14e51484b74ca2c8326aec7b4.jpg: 128x128 1
image 51/230 /content/drive/MyDrive/final_dataset/test/images/007_1_0067_png.rf.6ca3de3c37c0853e4b946731f2defabb.jpg: 128x128 1
image 52/230 /content/drive/MyDrive/final_dataset/test/images/007_1_0068_png.rf.af4a6a9b98bcc9e66875253c5adb3b35.jpg: 128x128 1
image 53/230 /content/drive/MyDrive/final_dataset/test/images/112_jpg.rf.1074e17d17c19270f88174bf2cb0b9b7.jpg: 128x128 1 thirty,
image 54/230 /content/drive/MyDrive/final_dataset/test/images/115_jpg.rf.36c42138855cb4405067963a6de5818c.jpg: 128x128 1 thirty,
```

```
# Collect confidence scores from the text files
files = glob.glob('/content/drive/MyDrive/final_dataset/runs/detect/yolov8n_v8_50e_infer1280/labels/*.txt')
confidences = []

for f in files:
    with open(f) as file:
        for line in file:
            conf = line.split()[-1]
            confidences.append(float(conf))

# Create a histogram to visualize the distribution of confidence scores
plt.figure(figsize=(8, 6))
plt.hist(confidences, bins=30, range=(0, 1), color='skyblue', edgecolor='black')
plt.title('Distribution of Confidence Scores')
plt.xlabel('Confidence Score')
plt.ylabel('Frequency')
```

```
plt.grid(True)
plt.show()
```

```
# Count the number of confidence scores less than 0.5 and greater than or equal to 0.5
count_below_0_5 = sum(1 for conf in confidences if conf < 0.5)
print(f'Number of confidence scores less than 0.5: {count_below_0_5}')
```

```
Number of confidence scores less than 0.5: 27
```

```
# Define the directory paths
label_dir = '/content/drive/MyDrive/final_dataset/runs/detect/yolov8n_v8_50e_infer1280/labels/'
output_dir = '/content/drive/MyDrive/final_dataset/low_confidence_images/'

# Create the output directory if it doesn't exist
os.makedirs(output_dir, exist_ok=True)

# Iterate through label files
for label_file in glob.glob(os.path.join(label_dir, '*.txt')):
    with open(label_file, 'r') as file:
        lines = file.readlines()

    # Extract confidence values from each line
    for line in lines:
        confidence = float(line.strip().split()[-1])

        # Check if confidence is less than 0.5
        if confidence < 0.5:
            # Extract image filename from label filename
            image_file = os.path.splitext(os.path.basename(label_file))[0] + '.jpg'

            # Copy the image to the output directory
            image_path_src = os.path.join('/content/drive/MyDrive/final_dataset/runs/detect/yolov8n_v8_50e_infer1280/', image_file)
            image_path_dst = os.path.join(output_dir, image_file)
            copyfile(image_path_src, image_path_dst)
```

```
#Low confidence images
# Get a list of image files in the output directory
image_files = glob.glob(os.path.join(output_dir, '*.jpg'))

# Shuffle the list of image files
random.shuffle(image_files)

# Display 10 random images from the directory
num_images_to_display = 10
plt.figure(figsize=(15, 8))
for i in range(num_images_to_display):
    if i < len(image_files):
        image_path = image_files[i]
        image = Image.open(image_path)
        plt.subplot(2, 5, i + 1)
        plt.imshow(image)
        plt.axis('off')
plt.tight_layout()
plt.show()
```

```
# Define the directory paths
label_dir = '/content/drive/MyDrive/final_dataset/runs/detect/yolov8n_v8_50e_infer1280/labels/'
output_dir = '/content/drive/MyDrive/final_dataset/low_confidence_cropped_images/'

# Create the output directory if it doesn't exist
os.makedirs(output_dir, exist_ok=True)

# Set the confidence threshold
confidence_threshold = 0.5

# Iterate through label files
for label_file in glob.glob(os.path.join(label_dir, '*.txt')):
    with open(label_file, 'r') as file:
        lines = file.readlines()

    # Extract image filename from label filename
    image_file = os.path.splitext(os.path.basename(label_file))[0] + '.jpg'
    image_path = os.path.join('/content/drive/MyDrive/final_dataset/runs/detect/yolov8n_v8_50e_infer1280/', image_file)
```

```
        # Load the image
        image = Image.open(image_path)
        width, height = image.size

        # Process each line in the label file
        for line in lines:
            values = line.strip().split()
            confidence = float(values[4])

            # Check if confidence is less than the threshold
            if confidence < confidence_threshold:
                # Extract bounding box coordinates
                x_center = float(values[1]) * width
                y_center = float(values[2]) * height
                box_width = float(values[3]) * width
                box_height = float(values[4]) * height

                # Calculate bounding box coordinates
                x_min = int(x_center - (box_width / 2))
                y_min = int(y_center - (box_height / 2))
                x_max = int(x_center + (box_width / 2))
                y_max = int(y_center + (box_height / 2))

                # Crop the region from the image
                cropped_region = image.crop((x_min, y_min, x_max, y_max))

                # Save the cropped region to the output directory
                output_path = os.path.join(output_dir, f'cropped_{image_file}')
                cropped_region.save(output_path)
```

```
# Define the directory path
output_dir = '/content/drive/MyDrive/final_dataset/low_confidence_cropped_images/'

# Get a list of image files in the output directory
image_files = glob.glob(os.path.join(output_dir, '*.jpg'))

# Shuffle the list of image files
random.shuffle(image_files)

# Display 5 random images from the directory
num_images_to_display = 3
plt.figure(figsize=(15, 6))
for i in range(num_images_to_display):
    if i < len(image_files):
        image_path = image_files[i]
        image = Image.open(image_path)
        plt.subplot(1, 5, i + 1)
        plt.imshow(image)
        plt.axis('off')
plt.tight_layout()
plt.show()
```

```
# Define the directory paths
label_dir = '/content/drive/MyDrive/final_dataset/runs/detect/yolov8n_v8_50e_infer1280/labels/'
output_dir = '/content/drive/MyDrive/final_dataset/all_cropped_images/'

# Create the output directory if it doesn't exist
os.makedirs(output_dir, exist_ok=True)

# Set the confidence threshold
confidence_threshold = 0.5

# Iterate through label files
for label_file in glob.glob(os.path.join(label_dir, '*.txt')):
    with open(label_file, 'r') as file:
        lines = file.readlines()

    # Extract image filename from label filename
    image_file = os.path.splitext(os.path.basename(label_file))[0] + '.jpg'
    image_path = os.path.join('/content/drive/MyDrive/final_dataset/runs/detect/yolov8n_v8_50e_infer1280/', image_file)

    # Load the image
    image = Image.open(image_path)
    width, height = image.size

    # Process each line in the label file
```

```
    for line in lines:
        values = line.strip().split()
        confidence = float(values[4])

        # Extract bounding box coordinates
        x_center = float(values[1]) * width
        y_center = float(values[2]) * height
        box_width = float(values[3]) * width
        box_height = float(values[4]) * height

        # Calculate bounding box coordinates
        x_min = int(x_center - (box_width / 2))
        y_min = int(y_center - (box_height / 2))
        x_max = int(x_center + (box_width / 2))
        y_max = int(y_center + (box_height / 2))

        # Crop the region from the image
        cropped_region = image.crop((x_min, y_min, x_max, y_max))

        # Save the cropped region to the output directory
        output_path = os.path.join(output_dir, f'cropped_{image_file}')
        cropped_region.save(output_path)
```

```
# Define the directory path
output_dir = '/content/drive/MyDrive/final_dataset/all_cropped_images/'

# Get a list of image files in the output directory
image_files = glob.glob(os.path.join(output_dir, '*.jpg'))

# Shuffle the list of image files
random.shuffle(image_files)

# Display 5 random images from the directory
num_images_to_display = 5
plt.figure(figsize=(15, 6))
for i in range(num_images_to_display):
    if i < len(image_files):
        image_path = image_files[i]
        image = Image.open(image_path)
        plt.subplot(1, 5, i + 1)
        plt.imshow(image)
        plt.axis('off')
plt.tight_layout()
plt.show()
```

```
yolo_confidence=np.average(confidences)
```

## grouping

```
# Define the directory paths
label_dir = '/content/drive/MyDrive/final_dataset/runs/detect/yolov8n_v8_50e_infer1280/labels/'
image_dir = '/content/drive/MyDrive/final_dataset/runs/detect/yolov8n_v8_50e_infer1280/'
output_dir = '/content/drive/MyDrive/final_dataset/grouped_images/'

# Create the output directory if it doesn't exist
os.makedirs(output_dir, exist_ok=True)

# Class names based on your YAML file
class_names = [
    'no_stopping',
    'eighty',
    'fifteen',
    'fifty',
    'five',
    'forty',
    'seventy',
    'sixty',
    'thirty',
    'twenty'
]

# Iterate through label files
for label_file in glob.glob(os.path.join(label_dir, '*.txt')):
```

```
    with open(label_file, 'r') as file:
        lines = file.readlines()

    # Extract image filename from label filename
    image_file = os.path.splitext(os.path.basename(label_file))[0] + '.jpg'
    class_label = None

    # Process each line in the label file
    for line in lines:
        values = line.strip().split()
        class_label = int(values[0])

    if class_label is not None:
        # Get the class name based on the label
        class_name = class_names[class_label]

        # Create a folder for the class if it doesn't exist
        class_folder = os.path.join(output_dir, class_name)
        os.makedirs(class_folder, exist_ok=True)

        # Copy the image to the class folder
        image_src_path = os.path.join(image_dir, image_file)
        image_dst_path = os.path.join(class_folder, image_file)
        copyfile(image_src_path, image_dst_path)
```

```
yolo_confidence
```

```
 0.9300451182795699
```

```
# Plot and visualize images in a 2x2 grid.
def visualize(result_dir, num_samples=4):
    """
    Function accepts a list of images and plots
    them in a 2x2 grid.
    """
    plt.figure(figsize=(30, 20))
    image_names = glob.glob(os.path.join(result_dir, '*.jpg'))
    random.shuffle(image_names)
    for i, image_name in enumerate(image_names):
        image = plt.imread(image_name)
        plt.subplot(2, 2, i+1)
        plt.imshow(image)
        plt.axis('off')
        if i == num_samples-1:
            break
    plt.tight_layout()
    plt.show()
```

```
visualize('/content/drive/MyDrive/final_dataset/runs/detect/yolov8n_v8_50e_infer1280')
```

## Autoencoder

```
import os
from operator import add
from matplotlib import pyplot as plt

import torch
import torch.nn as nn
import torch.nn.functional as F

from   torch.optim      import Adam
from torch.utils.data import Dataset
from torch.utils.data import DataLoader

import albumentations as A
from   albumentations.pytorch import ToTensorV2

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KernelDensity


from tqdm import tqdm


def save_checkpoint(save_dict, filename=""):
    torch.save(save_dict, filename)
```

```
def load_checkpoint(checkpoint_address, model, optimizer,verbose=False):
    model.load_state_dict(checkpoint_address["state_dict"])
    optimizer.load_state_dict(checkpoint_address["optimizer"])
    if verbose:
        print("=> Checkpoint Loaded")
```

## Model Definition

```
class Block(nn.Module):
    def __init__(self, in_features, out_features, down=True):
        super().__init__()

        self.conv = nn.Sequential(
                                    nn.Conv2d(in_features, out_features, kernel_size=4,stride=2,padding=1) if down
                                    else
                                    nn.ConvTranspose2d(in_features, out_features, kernel_size=4,stride=2,padding=1),
                                    nn.LeakyReLU()
        )

    def forward(self, x):
        return self.conv(x)

class AutoEncoder(nn.Module):
    def __init__(self, in_channels=3, features=[3,64,128,256,512,1024], return_bottleneck = False):
        super().__init__()

        self.return_bottleneck = return_bottleneck
        downlayers = []
        for feature in features[1:]:
            downlayers.append(Block(in_channels,feature))
            in_channels=feature

        uplayers = []
        for feature in features[::-1][1:]:
            uplayers.append(Block(in_channels,feature,down=False))
            in_channels=feature

        self.encoder = nn.Sequential(*downlayers)
        self.decoder = nn.Sequential(*uplayers)

    def forward(self,x):
        x = self.encoder(x)

        if self.return_bottleneck:
            bottleneck = x.clone()
            return self.decoder(x), bottleneck
        else:
            return self.decoder(x)
```

## Defining Image Paths

```
main_path = "/content/drive/MyDrive/yolo_data/traffic_signs/"
print(sorted(list( map(add, [main_path]*len(os.listdir(main_path)), os.listdir(main_path) ))))
```

```
['/content/drive/MyDrive/yolo_data/traffic_signs/eighty', '/content/drive/MyDrive/yolo_data/traffic_signs/fifteen', '/content/drive
```

```
list_of_image_paths = []

train_paths = []
val_paths   = []
test_paths  = []

for path in sorted(os.listdir(main_path)):
  list_of_image_paths+=sorted(list( map(add, [main_path+path+"/"]*len(os.listdir(main_path+path)), os.listdir(main_path+path) )))

  class_paths = sorted(list( map(add, [main_path+path+"/"]*len(os.listdir(main_path+path)), os.listdir(main_path+path) )))

  class_train_paths, class_valTest_paths = train_test_split(class_paths        , test_size = 0.30, random_state = 42)
  class_val_paths  , class_test_paths    = train_test_split(class_valTest_paths, test_size = 0.33, random_state = 42)

  train_paths += class_train_paths
  val_paths   += class_val_paths
  test_paths  += class_test_paths

  print(len(class_train_paths), len(class_val_paths), len(class_test_paths),"\t:",path)
```

```
list_of_image_paths = sorted(list_of_image_paths)
```

```
     175 50 26       : eighty
     42 12 6         : fifteen
     109 32 16       : fifty
     98 28 15        : five
     240 69 34       : forty
     77 22 11        : no_stopping
     320 92 46       : seventy
     53 15 8         : sixty
     105 30 15       : thirty
     147 42 21       : twenty
```

```
print(len(train_paths),"+",len(val_paths),"+",len(test_paths),"=",len(list_of_image_paths))
```

```
     1366 + 392 + 198 = 1956
```

```
 anomaly_paths = "/content/drive/MyDrive/final_dataset/low_confidence_images/"
 anomaly_paths = sorted(list( map(add, [anomaly_paths]*len(os.listdir(anomaly_paths)), os.listdir(anomaly_paths) )))
```

## Writing Dataset Class

```
from albumentations.core.transforms_interface  import ImageOnlyTransform, DualTransform,to_tuple

def make_normal(temp):
  return (temp - temp.min())/(temp.max()-temp.min())

class myNormalize(ImageOnlyTransform):
    """Normalization is applied by the formula: `img = (img - img.min) / (img.min - img.max)`
    Args:
        no args, we use the min and max of the image
    Targets:
        image
    Image types:
        uint8, float32
    """

    def __init__( self, always_apply=False, p=1.0,):
        super(myNormalize, self).__init__(always_apply, p)

    def apply(self, image, **params):
        return make_normal(image)



class TrafficDataset(Dataset):
  def __init__(self, paths, transform):
    self.paths     = paths
    self.transform = transform

  def __len__(self):
    return len(self.paths)

  def __getitem__(self, i):
    image = cv2.imread(self.paths[i])
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    image = make_normal(image)

    if self.transform:
      image = self.transform(image=image)['image'].float()

    return image


transform_train = A.Compose([
                                A.Resize(256,256),
                                A.VerticalFlip(p=0.3),
                                A.Rotate(10,p=0.3),
                                myNormalize(),
                                ToTensorV2()
])

transform_valTest = A.Compose([
                                A.Resize(256,256),
                                myNormalize(),
```

```
                                ToTensorV2()
])


train_dataset = TrafficDataset(paths = train_paths, transform = transform_train  )
val_dataset   = TrafficDataset(paths = val_paths  , transform = transform_valTest)
test_dataset  = TrafficDataset(paths = test_paths , transform = transform_valTest)

anomaly_dataset = TrafficDataset(paths = anomaly_paths, transform = transform_valTest)
all_dataset = TrafficDataset(paths = list_of_image_paths, transform = transform_train)



loader = DataLoader(dataset = anomaly_dataset, batch_size = 1, shuffle=False)
try:
  for idx, (temp_image) in enumerate(loader):
    # print(temp_image.shape)
    plt.figure(figsize=(4,4))
    plt.imshow(temp_image[0].transpose(0,2).transpose(1,0))
    plt.axis('off')
    plt.title(temp_image[0].shape)
    plt.show()
except KeyboardInterrupt:
  pass
```

## Defining Hyperparameters

```
batch_size    = 16
learning_rate = 1e-3
num_epochs    = 100


device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

checkpoint_address = "/content/drive/MyDrive/yolo_data/AutoEncoder_Reconstruction.pt"
grid_address       = "/content/drive/My Drive/yolo_data/grid/Grid_"

model = AutoEncoder()
optimizer = Adam(model.parameters(), lr= learning_rate)

loss_fn = nn.L1Loss()

load_model = True


model_with_btn = AutoEncoder(return_bottleneck=True)
if load_model:
  model_with_btn = model_with_btn.to(device)
  load_checkpoint(model=model_with_btn, optimizer=optimizer, checkpoint_address = torch.load(checkpoint_address), verbose=True)

     => Checkpoint Loaded


if load_model:
  model = model.to(device)
  load_checkpoint(model=model, optimizer=optimizer, checkpoint_address = torch.load(checkpoint_address), verbose=True)

     => Checkpoint Loaded


train_loader = DataLoader(dataset = train_dataset, batch_size = batch_size, shuffle=True )
val_loader   = DataLoader(dataset = val_dataset  , batch_size = batch_size, shuffle=False)
```

## Training

```
[ ] ↳ 3 cellules masquées
```

## Test Inference

```
train_loader_batchsize_1 = DataLoader(dataset = train_dataset  , batch_size = 1, shuffle=True )
val_loader_batchsize_1   = DataLoader(dataset = val_dataset    , batch_size = 1, shuffle=False)
test_loader              = DataLoader(dataset = test_dataset   , batch_size = 1, shuffle=False)
anomaly_loader           = DataLoader(dataset = anomaly_dataset, batch_size = 1, shuffle=False)



def get_list_of_latents(loader, model, device):
    model = model.to(device)
```

```
    loop = tqdm(loader)

    list_of_latents = []
    for idx, (image) in enumerate(loop):
        image = image.to(device)

        _, latent = model(image)

        list_of_latents.append(latent.flatten().detach().cpu().numpy())

    return list_of_latents



list_of_latents_train = get_list_of_latents(loader = train_loader_batchsize_1, model = model_with_btn, device = device)
kde = KernelDensity(kernel="gaussian", bandwidth=0.2).fit(list_of_latents_train)
```

```
 100%|██████████| 1366/1366 [00:23<00:00, 57.40it/s]
```

```
def get_recon_loss_and_density(loader, model, loss_fn, kde, device):
    model = model.to(device)
    total_loss    = []
    total_density = []
    loop = tqdm(loader)

    model.eval()
    with torch.no_grad():
      for batch_idx, (image) in enumerate(loop):
          image = image.to(device)
          reconstructed, latent = model(image)
          loss    = loss_fn(image, reconstructed)
          density = kde.score_samples(latent.flatten(start_dim=1).detach().cpu().numpy())[0]

          total_loss.append(loss.item())
          total_density.append(density)
          loop.set_postfix(loss = loss)

    model.train()
    return total_loss, total_density



total_loss        , total_density         =  get_recon_loss_and_density(loader = val_loader_batchsize_1, model = model_with_btn, loss_fn
total_loss_anomaly, total_density_anomaly =  get_recon_loss_and_density(loader = anomaly_loader        , model = model_with_btn, loss_fn
```

```
 100%|██████████| 392/392 [01:21<00:00,  4.82it/s, loss=tensor(0.0154, device='cuda:0')]
 100%|██████████| 71/71 [00:14<00:00,  4.95it/s, loss=tensor(0.0551, device='cuda:0')]
```

```
print("Normal Images")
print("\tDensity Mean\t\t:"             , np.mean(total_density))
print("\tDensity SD  \t\t:"             , np.std(total_density))
print("\tReconstruction Loss Mean:"     , np.mean(total_loss))
print("\tReconstruction Loss SD  :"     , np.std(total_loss))

print("Anomaly Images")
print("\tDensity Mean\t\t:"        , np.mean(total_density_anomaly))
print("\tDensity SD  \t\t:"        , np.std(total_density_anomaly))
print("\tReconstruction Loss Mean:", np.mean(total_loss_anomaly))
print("\tReconstruction Loss SD  :", np.std(total_loss_anomaly))
```

```
 Normal Images
        Density Mean            : 43000.00550364287
        Density SD              : 3245.393880222268
        Reconstruction Loss Mean: 0.015278786530109997
        Reconstruction Loss SD  : 0.00878220650556563
 Anomaly Images
        Density Mean            : 40251.45357854997
        Density SD              : 4408.7365934820045
        Reconstruction Loss Mean: 0.03585165369153862
        Reconstruction Loss SD  : 0.016070802715882418
```

```
density_threshold    = np.mean(total_density)
recon_loss_threshold = np.mean(total_loss_anomaly)



def check_anomaly(loader, model, loss_fn, device, density_thres, recon_loss_thres, make_plot = True):
    model = model.to(device)
    loop = tqdm(loader)
    true_count = 0
```

```
    false_count = 0
    model.eval()
    with torch.no_grad():
      for batch_idx, (image) in enumerate(loop):
          image = image.to(device)
          reconstructed, latent = model(image)
          loss    = loss_fn(image, reconstructed)
          density = kde.score_samples(latent.flatten(start_dim=1).detach().cpu().numpy())[0]

          if  density < density_thres or loss > recon_loss_thres:
          #if  density < density_thres:
          #if  loss > recon_loss_thres:
              flag = True
              true_count +=1

          else:
              flag = False
              false_count +=1

          if make_plot:
              plt.figure()
              plt.subplot(1,2,1)
              plt.imshow(image.squeeze().transpose(0,2).transpose(0,1).detach().cpu())
              plt.axis('off')
              plt.title("Original")

              plt.subplot(1,2,2)
              plt.imshow(reconstructed.squeeze().transpose(0,2).transpose(0,1).detach().cpu())
              plt.axis('off')
              plt.title(f"l={round(loss.item(),6)} | d={density} | anomaly={flag}")
              plt.show()
    print()
    print(true_count,false_count)
```

## using density and loss

```
check_anomaly(loader = train_loader_batchsize_1, model = model_with_btn, loss_fn = loss_fn, device = device, density_thres = density_thre
```

```
100%|██████████| 1366/1366 [04:09<00:00,  5.46it/s]
289 1077
```

```
check_anomaly(loader = test_loader, model = model_with_btn, loss_fn = loss_fn, device = device, density_thres = density_threshold, recon_
```

```
100%|██████████| 198/198 [01:24<00:00,  2.35it/s]
70 128
```

```
check_anomaly(loader = anomaly_loader, model = model_with_btn, loss_fn = loss_fn, device = device, density_thres = density_threshold, rec
```

```
100%|██████████| 71/71 [00:19<00:00,  3.68it/s]
53 18
```

```
#old results
#Using Density & Loss
#                       Actual Positive  Actual Negative
# Predicted Positive :      1077                    289
# Predicted Negative :      139                     59

#Using Density Only
#          TP  FP
# Anomaly: 1096 270
# Normal : 142 56

#Using Loss Only
#          TP  FP
# Anomaly: 1076 290
# Normal : 125 105


#FN(ANOMALY= TRUE)
# TN (ANOMALY = FALSE)
check_anomaly(loader = anomaly_loader, model = model_with_btn, loss_fn = loss_fn, device = device, density_thres = 35123.12160878582, rec
```

```
#TP(ANOMALY= FALSE)
# FP (ANOMALY = TRUE)


check_anomaly(loader = test_loader, model = model_with_btn, loss_fn = loss_fn, device = device, density_thres = 35123.12160878582, recon_


import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Provided TP and FP values
anomaly_tp = [201,142, 125]
anomaly_fp = [29, 88, 105]
methods = ['Density & Loss', 'Density Only', 'Loss Only']

# Create confusion matrix data
confusion_matrix_data = np.array([anomaly_tp, anomaly_fp])

# Plot the heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix_data, annot=True, fmt='d', cmap='Blues', xticklabels=methods, yticklabels=['Anomaly (TP)', 'Normal (FP)'])
plt.xlabel('Methods')
plt.ylabel('True/False Positive')
plt.title('Confusion Matrix Heatmap')
plt.show()



import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Confusion matrix values
confusion_matrix_values = np.array([[1076, 290],
                                    [125, 105]])
#                       Actual Positive  Actual Negative
# Predicted Positive :      1096                   270
# Predicted Negative :      125                    105
# Normal : 142 56
# Labels for rows and columns
row_labels = ['Predicted Positive', 'Predicted Negative']
col_labels = ['Actual Positive', 'Actual Negative']

# Create the heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix_values, annot=True, fmt='d', cmap='Blues', xticklabels=col_labels, yticklabels=row_labels)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Confusion Matrix Heatmap using Loss')
plt.show()
```