%pip install ultralytics

!pip install ultralytics

from ultralytics import YOLO

import matplotlib.pyplot as plt

import cv2

import os

import random

import pandas as pd

import matplotlib.image as mpimg

import seaborn as sns

import torch

import numpy as np

sns.set\_style('darkgrid')

%matplotlib inline

img0 = "train/y104.jpg"

\_ = plt.figure(figsize = (5,5))

\_ = plt.axis('off')

\_ = plt.imshow(mpimg.imread(img0))

a=mpimg.imread(img0)

print(a.shape)

import json

train\_img = "train"

test\_img = "tests"

valid\_img = "valid"

image\_files = os.listdir(train\_img)

random\_images = random.sample(image\_files, 16)

# Set up the plot

fig, axs = plt.subplots(4, 4, figsize=(16, 16))

# Loop over the random images and plot the object detections

for i, image\_file in enumerate(random\_images):

    row = i // 4

    col = i % 4

    # Load the image

    image\_path = os.path.join(train\_img, image\_file)

    image = cv2.imread(image\_path)

    with open("train/annotations\_train.json", "r") as f:

      labeling = json.load(f)

    for i in labeling:

      annotations=labeling[str(i)]

      if str(annotations["filename"])==str(image\_file) and str(annotations["regions"][0]['shape\_attributes']["name"])=="polygon":

        a = annotations["regions"][0]['shape\_attributes']["all\_points\_x"]

        b = annotations["regions"][0]['shape\_attributes']["all\_points\_y"]

        annotations\_x = a

        annotations\_y = b

        image\_width = annotations["size"]

        image\_height = annotations["size"]

        x\_center = sum(annotations\_x)/len(annotations\_x)

        y\_center = sum(annotations\_y)/len(annotations\_y)

        width = max(annotations\_x) - min(annotations\_x)

        height = max(annotations\_y) - min(annotations\_y)

        x\_min = int((x\_center - width/2))

        y\_min = int((y\_center - height/2))

        x\_max = int((x\_center + width/2))

        y\_max = int((y\_center + height/2))

        cv2.rectangle(image, (x\_min, y\_min), (x\_max, y\_max), (0, 255, 0), 3)

      elif str(annotations["filename"])==str(image\_file) and str(annotations["regions"][0]['shape\_attributes']["name"])=="ellipse":

        cx=int(annotations["regions"][0]['shape\_attributes']["cx"])

        cy=int(annotations["regions"][0]['shape\_attributes']["cy"])

        rx=int(annotations["regions"][0]['shape\_attributes']["rx"])

        ry=int(annotations["regions"][0]['shape\_attributes']["ry"])

        theta=int(annotations["regions"][0]['shape\_attributes']["theta"])

        theta\_rad = np.radians(theta)

        cv2.ellipse(image, (cx, cy),(rx, ry),theta\_rad,0, 360, (0, 255, 0), 2)

    axs[row, col].imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

    axs[row, col].axis('off')

plt.show()

bounds=[]

images=[]

from pathlib import Path

from PIL import Image

for imgt in image\_files:

  for it in labeling:

      annotationst=labeling[str(it)]

      if str(annotationst["filename"])==str(imgt) and str(annotationst["regions"][0]['shape\_attributes']["name"])=="polygon":

        a = annotationst["regions"][0]['shape\_attributes']["all\_points\_x"]

        b = annotationst["regions"][0]['shape\_attributes']["all\_points\_y"]

        image\_patht = os.path.join(train\_img, imgt)

        image1 = cv2.imread(image\_patht)

        h,w,\_ = image1.shape

        file\_name = "images/t1/{}"

        new\_file = file\_name.format(imgt)

        cv2.imwrite(new\_file ,image1)

        annotations\_xt = a

        annotations\_yt = b

        image\_widtht = annotationst["size"]

        image\_heightt = annotationst["size"]

        x\_centert = sum(annotations\_xt)/len(annotations\_xt)

        y\_centert = sum(annotations\_yt)/len(annotations\_yt)

        widtht = max(annotations\_xt) - min(annotations\_xt)

        heightt = max(annotations\_yt) - min(annotations\_yt)

        x\_min = int((x\_centert - widtht/2))

        y\_min = int((y\_centert - heightt/2))

        bounds.append([imgt,x\_centert/w,y\_centert/h,widtht/w,heightt/h])

        images.append(imgt)

for j in images:

  for bound in bounds:

    if bound[0] == j:

      bound[0] = 0

      t=j.replace(".jpg",".txt")

      file\_name = "labels/t1/{}"

      file\_real = file\_name.format(t)

      with open(file\_real,"w") as f:

        line = " ".join([str(x) for x in bound])

        f.write(line + "\n")

print(bounds)

image\_files\_v = os.listdir(valid\_img)

bounds\_v=[]

images\_v=[]

from pathlib import Path

with open("valid/annotations\_val.json","r") as f\_v:

  labeling\_v = json.load(f\_v)

for img\_v in image\_files\_v:

  for i in labeling\_v:

      annotations\_v=labeling\_v[str(i)]

      if str(annotations\_v["filename"])==str(img\_v) and str(annotations\_v["regions"][0]['shape\_attributes']["name"])=="polygon":

        av = annotations\_v["regions"][0]['shape\_attributes']["all\_points\_x"]

        bv = annotations\_v["regions"][0]['shape\_attributes']["all\_points\_y"]

        print("hi")

        image\_path\_v = os.path.join(valid\_img, img\_v)

        image1\_v = cv2.imread(image\_path\_v)

        h\_v,w\_v,\_ = image1\_v.shape

        file\_name\_v = "images/v1/{}"

        new\_file\_v = file\_name\_v.format(img\_v)

        cv2.imwrite(new\_file\_v, image1\_v)

        annotations\_xv = av

        annotations\_yv = bv

        image\_widthv = annotations\_v["size"]

        image\_heightv = annotations\_v["size"]

        x\_centerv = sum(annotations\_xv)/len(annotations\_xv)

        y\_centerv = sum(annotations\_yv)/len(annotations\_yv)

        widthv = max(annotations\_xv) - min(annotations\_xv)

        heightv = max(annotations\_yv) - min(annotations\_yv)

        x\_minv = int((x\_centerv - widthv/2))

        y\_minv = int((y\_centerv - heightv/2))

        bounds\_v.append([img\_v,x\_centerv/w\_v,y\_centerv/h\_v,widthv/w\_v,heightv/h\_v])

        images\_v.append(img\_v)

print(bounds\_v)

for j\_v in images\_v:

  for bound\_v in bounds\_v:

    if bound\_v[0] == j\_v:

      bound\_v[0] = 0

      t=j\_v.replace(".jpg",".txt")

      file\_name\_v = "labels/v1/{}"

      file\_real\_v = file\_name\_v.format(t)

      with open(file\_real\_v,"w") as f:

        line = " ".join([str(x) for x in bound\_v])

        f.write(line + "\n")

CONFIG = """

# train and val datasets (image directory or \*.txt file with image paths)

train: /content/images/t1

val: /content/images/v1

# number of classes

nc: 1

# class names

names: ['MRI']

"""

with open("data.yaml", "w") as f:

    f.write(CONFIG)

import torch

torch.cuda.empty\_cache()

model = YOLO("yolov8m.pt")

model.train(data='data.yaml', epochs=3,imgsz=640,batch=10,cache=False)

%matplotlib inline

# read in the results.csv file as a pandas dataframe

df = pd.read\_csv('/content/runs/detect/train12/results.csv')

df.columns = df.columns.str.strip()

# create subplots using seaborn

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=df, ax=axs[0,0])

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

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

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

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

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

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

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

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

sns.lineplot(x='epoch', y='val/dfl\_loss', data=df, 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')

# add suptitle and subheader

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

# adjust top margin to make space for suptitle

plt.subplots\_adjust(top=0.8)

# adjust spacing between subplots

plt.tight\_layout()

plt.show()

%matplotlib inline

# Reading the confusion matrix image file

img = mpimg.imread('/content/runs/detect/train12/confusion\_matrix.png')

# Plotting the confusion matrix image

fig, ax = plt.subplots(figsize = (15, 15))

ax.imshow(img)

ax.axis('off');

def ship\_detect(img\_path):

    # Read the image

    img = cv2.imread(img\_path)

    # Pass the image through the detection model and get the result

    detect\_result = model(img)

    # Plot the detections

    detect\_img = detect\_result[0].plot()

    # Convert the image to RGB format

    detect\_img = cv2.cvtColor(detect\_img, cv2.COLOR\_BGR2RGB)

    return detect\_img

import random

# Define the directory where the custom images are stored

custom\_image\_dir = 'train'

# Get the list of image files in the directory

image\_files = os.listdir(custom\_image\_dir)

# Select 16 random images from the list

selected\_images = random.sample(image\_files, 16)

# Create a figure with subplots for each image

fig, axes = plt.subplots(nrows=4, ncols=4, figsize=(15, 15))

# Iterate over the selected images and plot each one

for i, img\_file in enumerate(selected\_images):

    # Compute the row and column index of the current subplot

    row\_idx = i // 4

    col\_idx = i % 4

    # Load the current image and run object detection

    img\_path = os.path.join(custom\_image\_dir, img\_file)

    detect\_img = ship\_detect(img\_path)

    # Plot the current image on the appropriate subplot

    axes[row\_idx, col\_idx].imshow(detect\_img)

    axes[row\_idx, col\_idx].axis('off')

# Adjust the spacing between the subplots

plt.subplots\_adjust(wspace=0.05, hspace=0.05)