# Install pycocotools first

!pip install pycocotools

# Script to download COCO dataset

from pycocotools.coco import COCO

import requests

import os

# Define URLs for the COCO dataset annotations and images

annotations\_url = 'http://images.cocodataset.org/annotations/annotations\_trainval2017.zip'

images\_url = 'http://images.cocodataset.org/zips/train2017.zip'

# Create directories for the dataset

os.makedirs('annotations', exist\_ok=True)

os.makedirs('images', exist\_ok=True)

# Download annotations and extract

!wget {annotations\_url} -O annotations.zip

!unzip -q annotations.zip -d annotations

# Download images and extract

!wget {images\_url} -O train2017.zip

!unzip -q train2017.zip -d images

# Load the COCO annotations

coco = COCO('annotations/instances\_train2017.json')

# Get category IDs for 'animal' and 'vehicle'

cat\_ids = coco.getCatIds(catNms=['animal', 'vehicle'])

img\_ids = coco.getImgIds(catIds=cat\_ids)

# Download the first 10 images for demonstration

for img\_id in img\_ids[:10]:

    img\_info = coco.loadImgs(img\_id)[0]

    img\_url = img\_info['coco\_url']

    img\_data = requests.get(img\_url).content

    with open(os.path.join('images/', img\_info['file\_name']), 'wb') as handler:

        handler.write(img\_data)

print("Images downloaded successfully.")

# Step 1: Install necessary libraries (if not already installed)

!pip install pandas matplotlib seaborn openpyxl

# Step 2: Import libraries

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Step 3: Load the dataset

df = pd.read\_excel('/content/object\_distribution\_dataset.xlsx')

# Step 4: Count the occurrences of each category

object\_counts = df['category\_name'].value\_counts()

# Step 5: Create the bar plot

plt.figure(figsize=(10, 6))

sns.barplot(x=object\_counts.index, y=object\_counts.values, palette='viridis')

plt.title('Distribution of Objects in the Dataset', fontsize=16)

plt.xlabel('Object Category', fontsize=14)

plt.ylabel('Frequency', fontsize=14)

plt.xticks(rotation=45, ha='right')  # Rotate x labels for better visibility

plt.tight\_layout()  # Adjust layout to prevent overlap

plt.show()

# Let's create a sample dataset structure for demonstration purposes.

import os

import numpy as np

from PIL import Image

# Create directory structure

base\_dir = '/mnt/data/train'

os.makedirs(base\_dir, exist\_ok=True)

# Create sample classes

classes = ['cats', 'dogs']

for cls in classes:

    os.makedirs(os.path.join(base\_dir, cls), exist\_ok=True)

# Generate sample images for each class

for cls in classes:

    for i in range(5):  # Creating 5 sample images per class

        img = Image.fromarray((np.random.rand(224, 224, 3) \* 255).astype('uint8'))

        img.save(os.path.join(base\_dir, cls, f'{cls}\_{i}.jpg'))

# Zip the created dataset directory for download

import shutil

shutil.make\_archive('/mnt/data/train\_dataset', 'zip', base\_dir)

# Step 1: Install necessary libraries (if not already installed)

!pip install tensorflow

# Step 2: Import libraries

from google.colab import files

import os

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Step 3: Create a directory to hold the dataset

os.makedirs('train', exist\_ok=True)

# Step 4: Upload dataset files (optional, if you don't have files in 'train/' yet)

# This will allow you to upload your dataset files from your local machine.

uploaded = files.upload()

# Step 5: Setup your directory structure in the 'train' folder

# Assuming you have subdirectories for each class after uploading the images.

# Step 6: Initialize the ImageDataGenerator

datagen = ImageDataGenerator(

    rescale=1./255,

    rotation\_range=30,

    width\_shift\_range=0.2,

    height\_shift\_range=0.2,

    horizontal\_flip=True,

    fill\_mode='nearest'

)

# Step 7: Load the training data

train\_data = datagen.flow\_from\_directory('train/', target\_size=(224, 224), class\_mode='categorical')

# You can now proceed with your model training using train\_data

import pandas as pd

# Create a sample dataset for environmental monitoring using object detection

data = {

    'image\_id': [f'image\_{i}' for i in range(1, 101)],

    'image\_path': [f'/path/to/images/image\_{i}.jpg' for i in range(1, 101)],

    'label': ['vehicle' if i % 5 == 0 else 'animal' for i in range(1, 101)],

    'xmin': [i % 416 for i in range(1, 101)],

    'ymin': [i % 416 for i in range(1, 101)],

    'xmax': [(i % 416) + 50 for i in range(1, 101)],

    'ymax': [(i % 416) + 50 for i in range(1, 101)],

}

# Create a DataFrame

df = pd.DataFrame(data)

# Save the DataFrame to a CSV file

file\_path = '/mnt/data/environmental\_monitoring\_object\_detection\_data.csv'

df.to\_csv(file\_path, index=False)

file\_path

# Creating a synthetic dataset based on the user's previous requests and data context.

import pandas as pd

import numpy as np

# Create a synthetic dataset for environmental monitoring with object detection

data = {

    'image\_path': [

        'images/image1.jpg',

        'images/image2.jpg',

        'images/image3.jpg',

        'images/image4.jpg',

        'images/image5.jpg'

    ],

    'label': [

        'vehicle',

        'animal',

        'vehicle',

        'animal',

        'animal'

    ],

    'coordinates': [

        '100,150,200,250',

        '50,60,150,180',

        '120,130,220,250',

        '30,40,90,110',

        '55,65,155,175'

    ]

}

# Convert to DataFrame

dataset = pd.DataFrame(data)

# Save to a CSV file

csv\_file\_path = '/mnt/data/environmental\_monitoring\_dataset.csv'

dataset.to\_csv(csv\_file\_path, index=False)

csv\_file\_path

!wget https://github.com/pjreddie/darknet/blob/master/cfg/yolov3.cfg?raw=true -O yolov3.cfg

!wget https://github.com/pjreddie/darknet/releases/download/v1.1/yolov3.weights -O yolov3.weights

# Importing necessary libraries

import requests

# URLs for YOLOv3 configuration and weights files

config\_url = "https://github.com/pjreddie/darknet/raw/master/cfg/yolov3.cfg"

weights\_url = "https://pjreddie.com/media/files/yolov3.weights"

# Downloading the YOLOv3 configuration file

config\_response = requests.get(config\_url)

with open("/mnt/data/yolov3.cfg", "wb") as config\_file:

    config\_file.write(config\_response.content)

# Downloading the YOLOv3 weights file

weights\_response = requests.get(weights\_url)

with open("/mnt/data/yolov3.weights", "wb") as weights\_file:

    weights\_file.write(weights\_response.content)

"/mnt/data/yolov3.cfg", "/mnt/data/yolov3.weights"

# Install OpenCV if not already installed

!pip install opencv-python

!pip install pandas

import cv2

import numpy as np

import pandas as pd

import os

def load\_yolo\_model(config\_path='yolov3.cfg', weights\_path='yolov3.weights'):

    """

    Load the YOLO model using the specified configuration and weights files.

    :param config\_path: Path to the YOLO configuration file.

    :param weights\_path: Path to the YOLO weights file.

    :return: Loaded YOLO model.

    """

    if not os.path.exists(config\_path) or not os.path.exists(weights\_path):

        raise FileNotFoundError("YOLO configuration or weights file not found. Please check the paths.")

    # Load the YOLO model

    net = cv2.dnn.readNetFromDarknet(config\_path, weights\_path)

    return net

def prepare\_data(dataset\_path):

    """

    Load and prepare data from the specified dataset path.

    :param dataset\_path: Path to the dataset CSV file.

    :return: Loaded data as a DataFrame.

    """

    if not os.path.exists(dataset\_path):

        raise FileNotFoundError("Dataset file not found. Please check the path.")

    # Read data from the CSV file

    data = pd.read\_csv(dataset\_path)

    return data

if \_\_name\_\_ == "\_\_main\_\_":

    # Upload your dataset file to the Colab environment or specify the correct path in Google Drive

    dataset\_path = 'path/to/your/dataset.csv'  # Replace with your dataset path

    try:

        # Load the YOLO model

        yolo\_model = load\_yolo\_model()

        # Prepare data

        train\_data = prepare\_data(dataset\_path)

        # Implement your training logic or inference using yolo\_model here

        print("Data loaded successfully:")

        print(train\_data.head())  # Display the first few rows of the dataset

    except FileNotFoundError as e:

        print(e)

    except Exception as e:

        print("An error occurred:", e)

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

import tensorflow as tf

from tensorflow.keras.preprocessing.image import img\_to\_array

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Flatten, Dense

from tensorflow.keras.utils import to\_categorical

import matplotlib.pyplot as plt

# Simulated dataset creation

data\_dict = {

    'Image\_ID': [f'img\_{i+1}.jpg' for i in range(10)],

    'Category': ['cat', 'dog', 'cat', 'dog', 'cat',

                 'dog', 'cat', 'dog', 'cat', 'dog'],

    'BoundingBox\_X1': np.random.randint(0, 50, size=10),

    'BoundingBox\_Y1': np.random.randint(0, 50, size=10),

    'BoundingBox\_X2': np.random.randint(100, 150, size=10),

    'BoundingBox\_Y2': np.random.randint(100, 150, size=10),

    'Confidence': np.random.rand(10)

}

# Create DataFrame from the simulated data

data = pd.DataFrame(data\_dict)

# Check the columns in the DataFrame

print("Columns in the dataset:", data.columns)

# Generate random images for demonstration

def generate\_random\_images(num\_images):

    images = []

    for \_ in range(num\_images):

        # Create a random image (224x224 pixels, 3 color channels)

        random\_image = np.random.rand(224, 224, 3)

        images.append(random\_image)

    return np.array(images)

# Create random images based on the number of samples

X = generate\_random\_images(len(data))

# Map categories to integers

category\_map = {category: idx for idx, category in enumerate(data['Category'].unique())}

y = data['Category'].map(category\_map).values

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Convert labels to categorical one-hot encoding

y\_train = to\_categorical(y\_train, num\_classes=len(category\_map))

y\_test = to\_categorical(y\_test, num\_classes=len(category\_map))

# Model training setup

def create\_model():

    model = Sequential([

        Flatten(input\_shape=(224, 224, 3)),  # Input shape for the model

        Dense(128, activation='relu'),

        Dense(len(category\_map), activation='softmax')  # Output layer

    ])

    return model

# Instantiate and compile the model

model = create\_model()

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Train the model

model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=20, batch\_size=2)

# Optionally visualize a random image from the dataset

plt.imshow(X[0])  # Display the first generated random image

plt.title('Sample Random Image')

plt.axis('off')

plt.show()

from sklearn.metrics import precision\_score, recall\_score, average\_precision\_score

import numpy as np

# Make predictions on the test set

predictions = model.predict(X\_test)

# Convert predictions to binary labels based on a threshold

threshold = 0.5  # Adjust this threshold as needed

predicted\_labels = (predictions > threshold).astype(int)

# Ensure y\_test is in binary format (if it's not already)

y\_test\_binary = np.array(y\_test)  # Assuming y\_test is in a format suitable for conversion

# Calculate evaluation metrics with zero\_division set to 0

precision = precision\_score(y\_test\_binary, predicted\_labels, average='weighted', zero\_division=0)

recall = recall\_score(y\_test\_binary, predicted\_labels, average='weighted', zero\_division=0)

# Calculate mAP (Mean Average Precision)

def calculate\_mAP(y\_true, y\_scores):

    """

    Calculate mean Average Precision (mAP)

    :param y\_true: True binary labels

    :param y\_scores: Predicted probabilities for the positive class

    :return: mAP value

    """

    # Calculate average precision for each class

    average\_precisions = []

    for i in range(y\_true.shape[1]):  # Iterate through each class

        average\_precisions.append(average\_precision\_score(y\_true[:, i], y\_scores[:, i]))

    # Return the mean of average precisions

    return np.mean(average\_precisions)

# Calculate mAP using predicted probabilities

mAP = calculate\_mAP(y\_test\_binary, predictions)

print(f'Precision: {precision:.4f}')

print(f'Recall: {recall:.4f}')

print(f'mAP: {mAP:.4f}')