**Import Random Image from Image Directory:**

import os

import random

import cv2

import matplotlib.pyplot as plt

def select\_random\_image\_from\_folder(folder\_path):

    # Verify if the folder exists

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

        raise FileNotFoundError(f"The folder path {folder\_path} does not exist.")

    # Get a list of all files in the folder

    files = os.listdir(folder\_path)

    # Filter out only image files (optional step)

    image\_files = [f for f in files if f.lower().endswith(('png', 'jpg', 'jpeg', 'bmp', 'gif', 'tiff'))]

    if not image\_files:

        raise FileNotFoundError(f"No image files found in the folder {folder\_path}.")

    # Select a random image

    random\_image = random.choice(image\_files)

    # Load the selected image

    image\_path = os.path.join(folder\_path, random\_image)

    image = cv2.imread(image\_path)

    return image, random\_image

# Example usage

folder\_path = r"F:\Bird\_Data\Bird\images\train"  # Replace with your absolute folder path

image, image\_name = select\_random\_image\_from\_folder(folder\_path)

# Display the randomly selected image

image\_rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

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

plt.imshow(image\_rgb)

plt.title(f"Randomly Selected Image: {image\_name}")

plt.axis('off')

plt.show()



**Load TFlite model**

import numpy as np

import tensorflow as tf

import matplotlib.pyplot as plt

# Load TFLite model and allocate tensors

interpreter = tf.lite.Interpreter(model\_path=r"F:\Bird\_Data\Bird\trained\_models\newModel\my\_model.tflite")

interpreter.allocate\_tensors()

# Get input and output tensors

input\_details = interpreter.get\_input\_details()

output\_details = interpreter.get\_output\_details()

**Run Inference on Randomly selected image**

import cv2

import numpy as np

def run\_inference\_tflite(interpreter, image):

    # Get input details

    input\_details = interpreter.get\_input\_details()

    # Resize the image to the expected input size

    input\_shape = input\_details[0]['shape']

    height, width = input\_shape[1], input\_shape[2]

    resized\_image = cv2.resize(image, (width, height))

    # Preprocess the image

    input\_data = np.expand\_dims(resized\_image, axis=0).astype(np.float32)

    # Set the input tensor

    interpreter.set\_tensor(input\_details[0]['index'], input\_data)

    # Run inference

    interpreter.invoke()

    # Get output details

    output\_details = interpreter.get\_output\_details()

    # Extract predictions

    bbox\_pred = interpreter.get\_tensor(output\_details[1]['index'])

    class\_pred = interpreter.get\_tensor(output\_details[0]['index'])

    return bbox\_pred[0], class\_pred[0]

# Example: Running inference on the first test image

bbox\_pred, class\_pred = run\_inference\_tflite(interpreter, image)

**Visualize the prediction:**

def visualize\_tflite\_output(image, bbox, class\_pred, class\_names):

    # Denormalize the bounding box

    h, w, \_ = image.shape

    x\_min, y\_min, x\_max, y\_max = bbox

    x\_min = int(x\_min \* w)

    y\_min = int(y\_min \* h)

    x\_max = int(x\_max \* w)

    y\_max = int(y\_max \* h)

    # Plot the image

    plt.figure(figsize=(8, 8))

    plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

    # Plot the bounding box

    plt.gca().add\_patch(plt.Rectangle((x\_min, y\_min), x\_max - x\_min, y\_max - y\_min,

                                      edgecolor='red', facecolor='none', linewidth=2))

    # Plot the label

    label\_idx = np.argmax(class\_pred)  # Get the index of the class

    class\_name = class\_names[label\_idx]

    plt.text(x\_min, y\_min - 10, class\_name, color='red', fontsize=12, weight='bold')

    plt.axis('off')

    plt.show()

# Visualize the result

visualize\_tflite\_output(image, bbox\_pred, class\_pred, class\_names={0: 'SuperBird', 1: 'Bird', 2: 'Background'})

