**Python Environment:**

* Install Python
* Create Python environment using requirement.txt

**Load custom data and its annotation file to create Custom Dataset:**

**Load Annotation File**

import json

import cv2

import numpy as np

from tensorflow.keras.utils import to\_categorical

# Load the JSON file

with open('Bird/annotations/train\_annotations.json', 'r') as f:

    annotations = json.load(f)

**Create Dataset:**

def load\_dataset(annotations, image\_dir, input\_shape=(224, 224), num\_classes=3):

    images = []

    bboxes = []

    labels = []

    for annotation in annotations['annotations']:

        image\_id = annotation['image\_id']

        bbox = annotation['bbox']  # [x, y, width, height]

        category\_id = annotation['category\_id']

        # Load the image

        image\_info = next(item for item in annotations['images'] if item['id'] == image\_id)

        image\_path = f"{image\_dir}/{image\_info['file\_name']}"

        image = cv2.imread(image\_path)

        image = cv2.resize(image, input\_shape[:2])

        # Normalize the bounding box

        x\_min, y\_min, width, height = bbox

        x\_max = x\_min + width

        y\_max = y\_min + height

        x\_min /= image\_info['width']

        y\_min /= image\_info['height']

        x\_max /= image\_info['width']

        y\_max /= image\_info['height']

        # Append the data

        images.append(image)

        bboxes.append([x\_min, y\_min, x\_max, y\_max])

        labels.append(category\_id)

    images = np.array(images)

    bboxes = np.array(bboxes)

    labels = to\_categorical(labels, num\_classes=num\_classes)

    return images, bboxes, labels

# Specify the directory where images are stored

image\_dir = r'Bird/images/train'

# Load the dataset

X, y\_bboxes, y\_labels = load\_dataset(annotations, image\_dir, input\_shape=(224, 224), num\_classes=3)

**Split dataset in training and validation:**

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

X\_train, X\_val, y\_bboxes\_train, y\_bboxes\_val, y\_labels\_train, y\_labels\_val = train\_test\_split(

    X, y\_bboxes, y\_labels, test\_size=0.2, random\_state=42)

**Visualize the dataset**

import matplotlib.pyplot as plt

# Class names (replace these with your actual class names)

class\_names = {0: 'SuperBird', 1: 'Bird', 2: 'Background'}

def visualize\_dataset(images, bboxes, labels, num\_images=2):

    for i in range(num\_images):

        image = images[i]

        bbox = bboxes[i]

        label = labels[i]

        # Denormalize 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(label)  # 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')

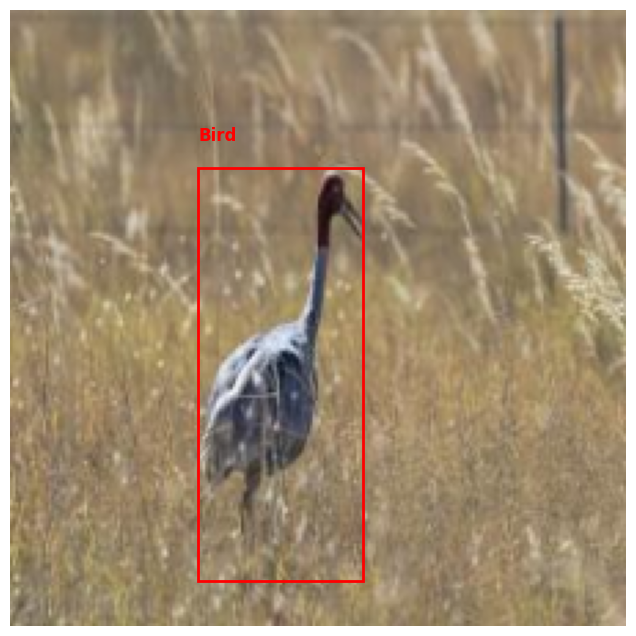
        plt.axis('off')

        plt.show()

# Visualize 5 images

visualize\_dataset(X, y\_bboxes, y\_labels, num\_images=5)

**First two images from Dataset and their bounding box:**



**Create Model:**

**Define function to create model:**

import tensorflow as tf

from tensorflow.keras import layers, models, Input

def create\_model(input\_shape=(224, 224, 3), num\_classes=3):

    inputs = Input(shape=input\_shape)

    # Convolutional layers

    x = layers.Conv2D(32, (3, 3), activation='relu')(inputs)

    x = layers.MaxPooling2D((2, 2))(x)

    x = layers.Conv2D(64, (3, 3), activation='relu')(x)

    x = layers.MaxPooling2D((2, 2))(x)

    x = layers.Conv2D(128, (3, 3), activation='relu')(x)

    x = layers.MaxPooling2D((2, 2))(x)

    x = layers.Conv2D(256, (3, 3), activation='relu')(x)

    x = layers.MaxPooling2D((2, 2))(x)

    x = layers.Flatten()(x)

    x = layers.Dense(512, activation='relu')(x)

    # Output layers

    bbox\_output = layers.Dense(4, activation='sigmoid', name='bbox\_output')(x)

    class\_output = layers.Dense(num\_classes, activation='softmax', name='class\_output')(x)

    # Model

    model = models.Model(inputs=inputs, outputs=[bbox\_output, class\_output])

    return model

**Create callbacks to monitor the progress**

from tensorflow.keras.callbacks import TensorBoard

# Define TensorBoard callback

tensorboard\_callback = TensorBoard(log\_dir='./logs', histogram\_freq=1)

**Create Model**

model = create\_model(input\_shape=(224, 224, 3), num\_classes=3)

model.compile(optimizer='adam',

              loss={'bbox\_output': 'mean\_squared\_error', 'class\_output': 'categorical\_crossentropy'},

              metrics={'bbox\_output': 'mse', 'class\_output': 'accuracy'})

**Train the model**

# Assuming X\_train is your input images, y\_bbox\_train is the bounding box labels, and y\_class\_train is the class labels

history = model.fit(X\_train,

                    {'bbox\_output': y\_bboxes\_train, 'class\_output': y\_labels\_train},

                    epochs=20,

                    batch\_size=32,

                    validation\_data=(X\_val, {'bbox\_output': y\_bboxes\_val, 'class\_output': y\_labels\_val})

                    callbacks=[tensorboard\_callback])

**Plot the Metrics**:

import matplotlib.pyplot as plt

def plot\_metrics(history):

    # Plot bounding box loss

    plt.figure(figsize=(12, 6))

    plt.plot(history.history['bbox\_output\_loss'], label='Bounding Box Loss (Train)')

    plt.plot(history.history['val\_bbox\_output\_loss'], label='Bounding Box Loss (Val)')

    plt.title('Bounding Box Loss')

    plt.xlabel('Epoch')

    plt.ylabel('Loss')

    plt.legend()

    plt.grid()

    plt.show()

    # Plot class accuracy

    plt.figure(figsize=(12, 6))

    plt.plot(history.history['class\_output\_accuracy'], label='Class Accuracy (Train)')

    plt.plot(history.history['val\_class\_output\_accuracy'], label='Class Accuracy (Val)')

    plt.title('Class Accuracy')

    plt.xlabel('Epoch')

    plt.ylabel('Accuracy')

    plt.legend()

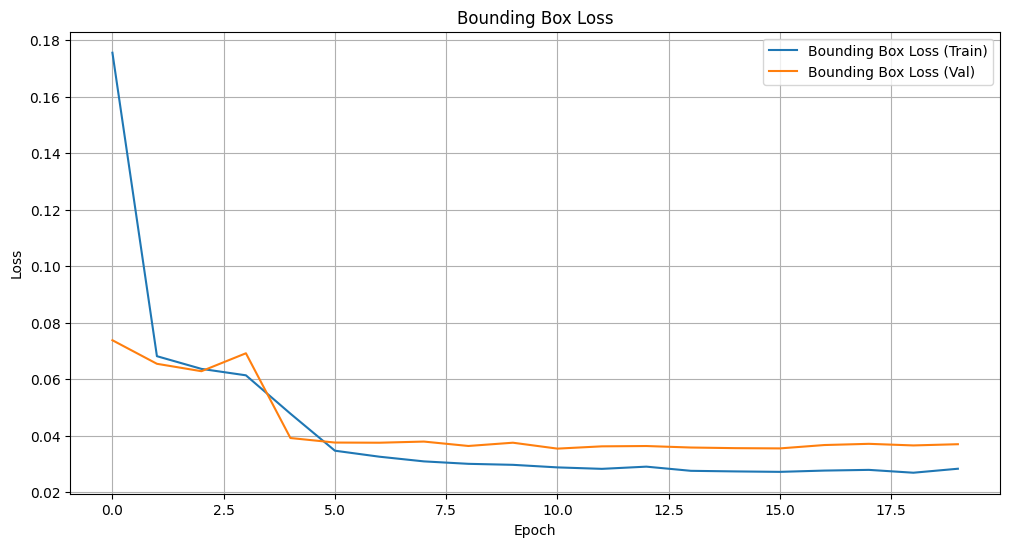
    plt.grid()

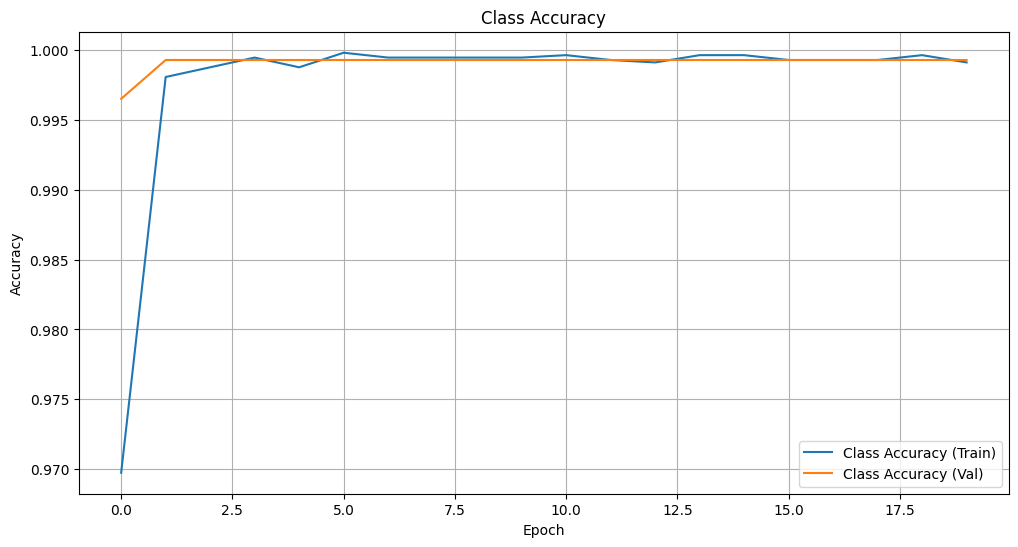
    plt.show()

# Plot the metrics

plot\_metrics(history)

**Metrics result:**





**Save the model to .h5 format:**

# Save the entire model to a file

model.save('Bird/trained\_models/newModel/my\_model.h5')

**Creation of TFLite Model:**

**Load Model**

from tensorflow.keras.models import load\_model

# Load the previously saved model

loaded\_model = load\_model('Bird/trained\_models/newModel/my\_model.h5')

**Convert and save to TFLite format**

import tensorflow as tf

# Convert the model to TFLite format

converter = tf.lite.TFLiteConverter.from\_keras\_model(loaded\_model)

converter.optimizations = [tf.lite.Optimize.DEFAULT]

tflite\_model = converter.convert()

# Save the TFLite model to a file

with open('Bird/trained\_models/newModel/my\_model.tflite', 'wb') as f:

    f.write(tflite\_model)