import tensorflow as tf

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense, Dropout

from tensorflow.keras.optimizers import Adam

import numpy as np

import matplotlib.pyplot as plt

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

(x\_train, y\_train), (x\_test, y\_test) = tf.keras.datasets.cifar10.load\_data()

# Normalize the data

x\_train = x\_train.astype('float32') / 255.0

x\_test = x\_test.astype('float32') / 255.0

# Convert labels to one-hot encoding

y\_train\_onehot = tf.keras.utils.to\_categorical(y\_train, 10)

y\_test\_onehot = tf.keras.utils.to\_categorical(y\_test, 10)

# Define class names for CIFAR-10

class\_names = [

"airplane", "automobile", "bird", "cat", "deer",

"dog", "frog", "horse", "ship", "truck"

]

# Define the RCNN-like model

def rcnn\_model():

input\_layer = Input(shape=(32, 32, 3))

# Convolutional base

x = Conv2D(32, (3, 3), activation='relu', padding='same')(input\_layer)

x = MaxPooling2D((2, 2))(x)

x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)

x = MaxPooling2D((2, 2))(x)

x = Flatten()(x)

# Fully connected layers for classification

classification\_output = Dense(128, activation='relu')(x)

classification\_output = Dense(10, activation='softmax', name="classification")(classification\_output)

# Fully connected layers for bounding box regression

bounding\_box\_output = Dense(128, activation='relu')(x)

bounding\_box\_output = Dense(4, activation='sigmoid', name="bounding\_box")(bounding\_box\_output)

# Combine outputs

model = Model(inputs=input\_layer, outputs=[classification\_output, bounding\_box\_output])

return model

model = rcnn\_model()

# Compile the model

model.compile(

optimizer=Adam(learning\_rate=0.001),

loss={

"classification": "categorical\_crossentropy", # For classification output

"bounding\_box": "mean\_squared\_error" # For bounding box regression

},

metrics={

"classification": "accuracy", # Track accuracy for classification

"bounding\_box": "mse" # Track MSE for bounding box

}

)

# Generate synthetic bounding boxes (normalized format: x\_min, y\_min, x\_max, y\_max)

# In a real dataset, use actual bounding box annotations

def generate\_bounding\_boxes(labels):

bounding\_boxes = []

for label in labels:

x\_min = np.random.uniform(0.1, 0.4)

y\_min = np.random.uniform(0.1, 0.4)

x\_max = x\_min + np.random.uniform(0.2, 0.5)

y\_max = y\_min + np.random.uniform(0.2, 0.5)

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

return np.array(bounding\_boxes)

y\_train\_boxes = generate\_bounding\_boxes(y\_train)

y\_test\_boxes = generate\_bounding\_boxes(y\_test)

# Train the model

history = model.fit(

x\_train,

{

"classification": y\_train\_onehot, # One-hot encoded classification labels

"bounding\_box": y\_train\_boxes # Bounding box labels (normalized)

},

validation\_data=(

x\_test,

{

"classification": y\_test\_onehot, # Validation classification labels

"bounding\_box": y\_test\_boxes # Validation bounding box labels

}

),

epochs=10,

batch\_size=64

)

# Visualize predictions

def visualize\_predictions(images, labels, boxes, predictions):

for i in range(5):

plt.figure(figsize=(4, 4))

plt.imshow(images[i])

true\_label = class\_names[np.argmax(labels[i])]

predicted\_label = class\_names[np.argmax(predictions[0][i])]

box = predictions[1][i] \* 32 # Scale bounding box back to pixel dimensions

x\_min, y\_min, x\_max, y\_max = box

plt.gca().add\_patch(plt.Rectangle(

(x\_min, y\_min), x\_max - x\_min, y\_max - y\_min,

fill=False, edgecolor='red', linewidth=2

))

plt.title(f"True: {true\_label}, Pred: {predicted\_label}")

plt.axis('off')

plt.show()

predictions = model.predict(x\_test[:5])

visualize\_predictions(x\_test[:5], y\_test\_onehot[:5], y\_test\_boxes[:5], predictions)

import numpy as np

from tqdm import tqdm

from sklearn.metrics import average\_precision\_score

import time

def calculate\_map(predictions, ground\_truth\_boxes, ground\_truth\_labels, num\_classes=10, iou\_threshold=0.5):

"""

Calculates mean Average Precision (mAP) for object detection.

Args:

predictions: A list of predictions, each containing bounding boxes and class probabilities.

ground\_truth\_boxes: A list of ground truth bounding boxes.

ground\_truth\_labels: A list of ground truth class labels.

num\_classes: The number of classes in the dataset.

iou\_threshold: The IoU threshold for considering a detection as a true positive.

Returns:

The mAP score.

"""

average\_precisions = []

# predictions[0] contains classification probabilities

# predictions[1] contains bounding box coordinates

for class\_id in tqdm(range(num\_classes), desc="Calculating mAP"):

class\_predictions = []

class\_ground\_truths = []

for i in range(len(predictions[0])): # Iterate over the number of predictions

# Filter predictions and ground truths for the current class

# Access predictions using indices directly

if np.argmax(predictions[0][i]) == class\_id:

class\_predictions.extend([(predictions[1][i], predictions[0][i][class\_id])])

if ground\_truth\_labels[i][0] == class\_id: # Assuming ground\_truth\_labels is a list of lists or a 2D array

class\_ground\_truths.extend([(ground\_truth\_boxes[i], 1)])

# If no predictions or ground truths for this class, skip

if not class\_predictions or not class\_ground\_truths:

continue

# ... (rest of the mAP calculation logic, similar to standard implementations) ...

# Calculate and return mAP

map\_score = np.mean(average\_precisions)

return map\_score

def calculate\_fps(model, images, num\_frames=100):

"""

Calculates Frames Per Second (FPS) for a model.

Args:

model: The object detection model.

images: A list of images to process.

num\_frames: The number of frames to use for FPS calculation.

Returns:

The FPS value.

"""

start\_time = time.time()

for i in range(num\_frames):

\_ = model.predict(images[i:i + 1]) # Run inference on a single image

end\_time = time.time()

fps = num\_frames / (end\_time - start\_time)

return fps

predictions = model.predict(x\_test) # Get predictions for the test set

# Adapt the following lines to match your data format:

ground\_truth\_boxes = y\_test\_boxes # Replace with your ground truth bounding boxes

ground\_truth\_labels = y\_test # Replace with your ground truth class labels

# Calculate mAP

map\_score = calculate\_map(predictions, ground\_truth\_boxes, ground\_truth\_labels)

print(f"mAP: {map\_score}")

# Calculate FPS

fps = calculate\_fps(model, x\_test)

print(f"FPS: {fps}")

# ... (Your existing code) ...