cs512 f24 - assignment 4 (segmentation and detection)

Due by: 11/2/2024

General Instructions:

In this assignment, you will implement a basic Convolutional Neural Network (CNN) for semantic segmentation and object detection tasks. You can utilize frameworks such as Keras, TensorFlow, or PyTorch, and should run the models on a GPU (local or Google Colab).

You will use the `oxford_iiit_pet` dataset (available in Keras and PyTorch) for both the segmentation and detection tasks. For segmentation, you will implement different network architectures, and for detection, you will use a pre-trained YOLO model to detect objects and locate bounding boxes corresponding to object segments.

Ensure that your program saves and loads weights so that training can continue from the previous state. Additionally, you will log and plot metrics as a function of training and evaluation iterations.

Submission instructions:

- Follow the submission instructions of assignment 0.
- Submit a fully executed Python notebook with no gaps in execution. To receive full credit, please ensure the following:
  - The notebook includes all cell outputs and contains no error messages.
  - It is easy to match each problem with its corresponding code solution using clear markdown or comments (e.g., “Problem 2”).
  - Re-run your notebook before submitting to ensure that cells are numbered sequentially, starting at [1].

Questions:

1. Semantic Segmentation

1. **Dataset:** Use the `oxford_iiit_pet` dataset from Keras or PyTorch.
   - The dataset provides pixel-wise annotations for pets, including two classes: foreground (pet) and background, as well as an additional label for object contours. In this assignment, you will **ignore the contour labels** and focus only on the foreground segmentation.
   - The dataset also includes breed-level labels. To simplify, you will **convert the pet breed labels** into binary **cat/dog** labels. Assign all cat breeds to the "cat" category and all dog breeds to the "dog" category.

2. **Starter Code for Dataset Loading:**

   Split the dataset into training and testing subsets, then split the training st into a training and validation subsets.

```
import tensorflow_datasets as tfds
import tensorflow as tf

# Load the Oxford Pet Dataset
dataset, info = tfds.load('oxford_iiit_pet', with_info=True, as_supervised=True)

# Split into training and testing datasets
train_dataset = dataset['train']
test_dataset = dataset['test']

# Print some basic information about the dataset
print(info)

# Example: Access an image and label
for image, label in train_dataset.take(1):
    image = tf.image.convert_image_dtype(image, tf.float32)  # Convert image to float
    print(f"Image shape: {image.shape}")
    print(f"Label (breed): {label}")
```

3. **Training a Simple CNN for Semantic Segmentation (without skip connections):**
   - Train a simple convolutional neural network for supervised semantic segmentation (without skip connections).
   - Plot the training and validation loss and evaluation metric as a function of epochs.
   - Visualize some inference results.
   - Provide quantitative training, validation, and testing results.

4. **Training a CNN for Semantic Segmentation with Skip Connections (U-Net):**
   - Train a CNN for semantic segmentation with skip connections (as in U-Net).
   - Evaluate and visualize the results as before.

2. Object Detection

1. **Dataset:** Use the `oxford_iiit_pet` dataset from Keras or PyTorch.
   - For object detection, you will need to find **bounding boxes** that correspond to the pet objects in the segmentation masks. To create a bounding box for each pet, you can:
     - Identify the non-background pixels in the segmentation mask.
     - Compute the **minimum and maximum coordinates** (xmin, xmax, ymin, ymax) that enclose all the pet pixels. This will give you a bounding box for each image around the pet object.
     - Example of how to calculate bounding boxes from segmentation masks:

```
import numpy as np

def get_bounding_box(segmentation_mask):
    # Find non-zero regions in the segmentation mask (i.e., the pet area)
    pet_pixels = np.argwhere(segmentation_mask > 0)

    # Compute the bounding box coordinates
    ymin, xmin = np.min(pet_pixels, axis=0)
    ymax, xmax = np.max(pet_pixels, axis=0)

    return xmin, ymin, xmax, ymax
```

2. **Using a Pre-trained YOLO Model:**
    - Download a pre-trained YOLO model: YOLOv3 Model
    - Convert the model weights `.weights` to a Keras-compatible `.h5` file.
    - Load the model and test it on an image by normalizing the image and applying the model for predictions.
    - Show the detection results using bounding boxes drawn on the image.

3. **Starter Code for Pre-trained YOLOv3 Model Loading:**

```
from tensorflow.keras.models import load_model
import numpy as np
import cv2
import matplotlib.pyplot as plt

# Load the pre-trained YOLOv3 model
model = load_model('yolov3.h5')

# Load and preprocess an image
def load_image(img_path, target_size):
    image = cv2.imread(img_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image_resized = cv2.resize(image, target_size)
    image_resized = image_resized / 255.0  # Normalize to [0,1]
    return np.expand_dims(image_resized, axis=0), image

# Example image path and size
img_path = 'path_to_your_image.jpg'
input_size = (416, 416)  # Standard size for YOLOv3

# Load and preprocess image
input_image, original_image = load_image(img_path, input_size)

# Make predictions using YOLO
predictions = model.predict(input_image)

# Post-process and visualize the detection results
def draw_boxes(image, boxes):
    for box in boxes:
```

```
        xmin, ymin, xmax, ymax = box
        cv2.rectangle(image, (xmin, ymin), (xmax, ymax), (0, 255, 0), 2)
    return image

# Example bounding boxes (dummy values for illustration)
boxes = [[100, 150, 300, 350]]  # Replace with actual post-processed YOLO output
output_image = draw_boxes(original_image.copy(), boxes)

# Display the output
plt.imshow(output_image)
plt.axis('off')
plt.show()
```

4. **Evaluation:**
    - Evaluate the model's performance on the `oxford_iiit_pet` dataset using the known bounding boxes.
    - Calculate mAP at different IoU thresholds (mAP*{0.25}, mAP*{0.5}, mAP*{0.75}, mAP*{0.95}).