Documentation Perception:

Problem 1:

Object Detection with YOLO

Below is the Python script using the Ultralytics YOLO library to perform object detection on images in a specified folder:

Importing necessary libraries, including the Ultralytics YOLO library (YOLO) and the operating system module (os).

Load the YOLO model using YOLO('yolov8n.pt'), where 'yolov8n.pt' is the pretrained model file.

Define the function getFolder_Path(folder_path) to retrieve a list of image names from a specified folder path (./All_Images).

Define the function detect_Segments() to perform object detection on all images in the specified folder.

Within the detect_Segments() function, call getFolder_Path to get the list of file names from the specified folder.

Additional Information:

Loop through each file in the folder, predict object segments using the YOLO model for each image, and save the results in the same folder (save=True). Append the results for all images to the all_results list.

The detect_Segments() function returns the all_results list.

Finally, the script checks if it is being run as the main module and calls the detect_Segments() function.

Code Snippet 1:

```
from ultralytics import YOLO
import os

# Load YOLO model
model = YOLO('yolov8n.pt')
folder_path = './All_Images'  # Path where images are stored

def getFolder_Path(folder_path):
    """
    Get a list of file names in the given folder path.

    Parameters:
        folder_path (str): Path to the folder containing files.

    Returns:
        List[str]: A list of file names.
    """
    file_list = []

    for root, dirs, files in os.walk(folder_path):
        for file_name in files:
            file_list.append(file_name)
    return file_list

def detect_Segments():
    """
```

```
•     Detect segments in images using YOLO model and save
    results.
•
•     Returns:
•         List: A list of results for each image.
•     """
•     files_in_folder = getFolder_Path(folder_path)
•     all_results = []
•     for i in files_in_folder:
•         path = f'./All_Images/{i}'
•         results = model.predict(path, save=True)
•         all_results.append(results)
•
•     return all_results
•
• if __name__ == "__main__":
•     detect_Segments()
```

Problem 2:

Calculate count of the total number of entities and group the entities by name and report the count for each entity.

How The Code Works

This Python script that uses YOLO (You Only Look Once) model from the Ultralytics library to detect objects in images and then group the images based on the detected entities. Here's a breakdown of what the script does:

Importing necessary libraries and load the YOLO model using the 'yolov8n.pt' pretrained model.

Define a function getFolder_Path that retrieves the list of all files in the given folder_path

Define a function detect_Segments that runs YOLO to detect objects in each image in the folder. It saves the results, counts the number of detected entities, groups the images, and writes the results to CSV files.

Define a function count_Entities that takes the YOLO detection results and counts the occurrences of each detected entity.

Define a function group_images that creates folders for each detected entity and moves the images to their respective entity folders.

The script calls the detect_Segments function in the __main__ block to start the process of detecting entities in the images and grouping them accordingly.

Additional Information:

The data is saved in a CSV in the same folder as detected image in ./All_Images

The data is based of the results by the YOLO model and may vary from the original sometimes

- Code Snippet 2:
- from ultralytics import YOLO
- import os
- import numpy
- import csv
- import shutil
-
- # Load YOLO model
- model = YOLO('yolov8n.pt')
- folder_path = './All_Images' # Path to folder where images are stored
- output_folder = './Grouped_Images' # Path to folder where grouped images will be saved

```

def getFolder_Path(folder_path):
    """
    Get a list of file names in the given folder path.

    Parameters:
        folder_path (str): Path to the folder containing files.

    Returns:
        List[str]: A list of file names.
    """
    file_list = []

    for root, dirs, files in os.walk(folder_path):
        for file_name in files:
            file_list.append(file_name)
    return file_list

def detect_Segments():
    """
    Detect segments in images using YOLO model, count entities, group images, and write results to CSV.

    Returns:
        None
    """
    files_in_folder = getFolder_Path(folder_path)
    for i in files_in_folder:
        path = f'./All_Images/{i}'
        results = model.predict(path, save=True)
        counts, total_objects = count_Entities(results)
        group_images(i, counts)
        write_to_CSV(i, counts, total_objects)

def count_Entities(results):
```

```
•     """
•     Count detected entities from YOLO results.
•
•     Parameters:
•         results: YOLO prediction results.
•
•     Returns:
•         dict: Entity counts.
•         int: Total number of objects detected.
•     """
•     counts = {}
•     total_objects = 0
•
•     for result in results:
•         boxes = result.boxes.cpu().numpy()
•         total_objects += len(boxes)
•         for box in boxes:
•             cls = int(box.cls[0])
•             if cls not in counts:
•                 counts[cls] = 1
•             else:
•                 counts[cls] += 1
•
•     return counts, total_objects
•
• def write_to_CSV(img_name, counts, total_objects):
•     """
•     Write entity counts to a CSV file.
•
•     Parameters:
•         img_name (str): Image name.
•         counts (dict): Entity counts.
•         total_objects (int): Total number of objects detected.
•
•     Returns:
        None
    """
    name = img_name.split('.')[0]
    csv_path = f"./runs/detect/predict/{name}.csv"

    with open(csv_path, mode='w', newline='') as csv_file:
        writer = csv.writer(csv_file)
        writer.writerow(['Entity', 'Count'])
        for key in counts:
            writer.writerow([model.names[key], counts[key]])
        writer.writerow(['Total Objects', total_objects])

    print(f"Data saved to {csv_path}")

def group_images(img_name, counts):
    """
    Group images based on detected entities.

    Parameters:
        img_name (str): Image name.
        counts (dict): Entity counts.

    Returns:
        None
    """
    for entity_id in counts:
        entity_name = model.names[entity_id]
        group_folder = f"{output_folder}/Folder-grp{entity_id}"
        if not os.path.exists(group_folder):
            os.makedirs(group_folder)

        src_path = f"./All_Images/{img_name}"
        dst_path = f"{group_folder}/{img_name}"
        shutil.copy(src_path, dst_path)

```

- print(f"Images grouped for {img_name}")
-
- if __name__ == "__main__":
-     detect_Segments()

Problem 3:

Grouping Images by Detected Entities

Import Required Libraries:

- **from ultralytics import YOLO**
- **import os**
- **import numpy as np**
- **import csv**
- **import shutil**
- **from PIL import Image**
- **from skimage.metrics import structural_similarity as ssim**

- This section imports various libraries used throughout the code:
- **`ultralytics.YOLO`**: This is the class provided by the Ultralytics library that allows you to work with YOLO models for object detection.
- **`os`**: This module provides functions to interact with the operating system, like file operations and directory navigation.
- **`numpy`**: A powerful library for numerical operations in Python.
- **`csv`**: Library for working with CSV (Comma Separated Values) files.
- **`shutil`**: Provides higher-level file operations like moving, copying, and deleting files.
- **`PIL.Image`**: The Python Imaging Library, used for image processing.
- **`skimage.metrics.structural_similarity`**: A function from the scikit-image library for calculating structural similarity index between images.

Load YOLO Model:

- `model = YOLO('yolov8n.pt')  # Using yolov8n pretrained model`

- This creates an instance of the YOLO model using the Ultralytics library. It loads a pretrained YOLO model named 'yolov8n.pt'.
- **Define Paths:**

```
folder_path = './All_Images'     input images
output_folder = './Grouped_Images'  where grouped images will be saved
```

- These variables store the paths to the input folder containing images and the output folder where grouped images will be saved.

- **Function: `getFolder_Path(folder_path)`**
- pythonCopy code

-
  ```
  def getFolder_Path(folder_path):
      file_list = []
      for root, dirs, files in os.walk(folder_path):
          for file_name in files:
              file_list.append(file_name)
      return file_list
  ```

- This function retrieves a list of file names in the specified folder using the **`os.walk()`** function. It returns a list of filenames.
- **Function: `detect_Segments()`**
-

-
  ```
  def detect_Segments():
      files_in_folder = getFolder_Path(folder_path)
      for i in files_in_folder:
          path = f'./All_Images/{i}'
          results = model.predict(path, save=True)
          counts, total_objects = count_Entities(results)
          group_images(i, counts)
          write_to_CSV(i, counts, total_objects)
  ```

- This function coordinates the object detection and grouping process for all images in the **`folder_path`**. It iterates through the list of file names obtained from

getFolder_Path(), performs object detection using YOLO, counts entities, groups images, and writes data to CSV.

- **Function: count_Entities(results)**
-

```
def count_Entities(results):
    counts = {}
    total_objects = 0

    for result in results:
        boxes = result.boxes.cpu().numpy()
        total_objects += len(boxes)
        for box in boxes:
            cls = int(box.cls[0])
            if cls not in counts:
                counts[cls] = 1
            else:
                counts[cls] += 1

    return counts, total_objects
```

- This function takes a list of YOLO detection results and counts the detected entities. It returns a dictionary with entity counts and the total number of detected objects.
- **Function: write_to_CSV(img_name, counts, total_objects)**
-

```
def write_to_CSV(img_name, counts, total_objects):
    name = img_name.split('.')[0]
    csv_path = f"./runs/detect/predict/{name}.csv"

    # Write entity counts and total object count to a CSV file
    with open(csv_path, mode='w', newline='') as csv_file:
        writer = csv.writer(csv_file)
        writer.writerow(['Entity', 'Count'])
        for key in counts:
            writer.writerow([model.names[key], counts[key]])
        writer.writerow(['Total Objects', total_objects])

    print(f"Data saved to {csv_path}")
```

- This function writes entity counts and the total object count to a CSV file using the provided file name. It uses the entity IDs to get the entity names from the YOLO model's **`model.names`**.
- **Function: `calculate_image_similarity(image1_path, image2_path)`**
- pythonCopy code

-
  ```
  def calculate_image_similarity(image1_path, image2_path):
      image1 = Image.open(image1_path).convert("L")
      image2 = Image.open(image2_path).convert("L")
      similarity = ssim(np.array(image1), np.array(image2))
      return similarity
  ```
- This function calculates the structural similarity index between two grayscale images. It uses the **`skimage.metrics.structural_similarity`** function from the scikit-image library.
- **Function: `group_images(img_name, counts)`**

-
  ```
  def group_images(img_name, counts):
      for entity_id in counts:
          entity_name = model.names[entity_id]
          group_folder = f"{output_folder}/Folder-
  grp{entity_id}_{entity_name}"
          if not os.path.exists(group_folder):
              os.makedirs(group_folder)

          src_path = f"./All_Images/{img_name}"
          dst_path = f"{group_folder}/{img_name}"
          shutil.copy(src_path, dst_path)

      print(f"Images grouped for {img_name}")
  ```
- This function groups images into folders based on detected entities. It creates folders using the entity ID and name, then moves the image to the appropriate group folder.
- **Main Execution Block:**

-
  ```
  if __name__ == "__main__":
      detect_Segments()
  ```

- This block ensures that the **`detect_Segments()`** function is executed when the script is run directly (not imported as a module).

This script processes images using a YOLO model for object detection, counts the detected entities, groups the images based on entities, and saves the results in CSV files. The code is designed to be modular and maintainable, allowing you to adapt it for your specific use case.

Problem 4:

Object Detection and Comparison

Object Detection and Comparison

This HTML page provides an overview of the Python script that uses the Ultralytics YOLO library for object detection and entity comparison.

Import the necessary libraries: The script starts by importing the Ultralytics YOLO library (ultralytics.YOLO), the operating system module (os), and the CSV module (csv).

Load YOLO Model: The YOLO model is loaded using the pretrained model file named 'yolov8n.pt'.

Main Function: The main() function is defined as the entry point of the script. It processes subfolders within the main folder and performs comparisons.

Get Subfolders: The get_subfolders(parent_folder) function retrieves the paths of subfolders within a specified main folder.

Get Paths of Ideal and Other Images: The get_ideal_images(folder_path) function retrieves paths of ideal images from the "ideal" subfolder, and the get_other_images(folder_path) function retrieves paths of other images from the "other" subfolder.

Ideal Images Processing: The ideal_images(selected_folder_path) function processes ideal images, predicts entities using the YOLO model, and stores the results.

Counting Detected Entities and Coordinates: The count_Entities(results) function counts detected entities and extracts their coordinates from the YOLO results.

Comparing Entities: The compare_entities(ideal_results, other_result) function compares entities detected in ideal and other images, identifying missing and extra entities along with their coordinates.

Writing Data to CSV: The perform_comparison_and_write_csv(ideal_results, other_path, selected_folder_path) function performs entity comparison, writes results to a CSV file named after the other image, and includes information about missing and extra entities.

Calling Functions Inside main(): The main function iterates through subfolders, processes images, performs comparisons, and writes results to CSV files.

Calling Main Function: The main function is executed when the script is run as the main module.

This Python script efficiently processes images, performs entity comparison, and writes comparison results to CSV files using the Ultralytics YOLO library.

Import Required Modules:

- import os
- import csv
- from ultralytics import YOLO

- **os**: This module provides a way to interact with the operating system, including tasks like file and folder operations.
- **csv**: This module is used to work with CSV (Comma Separated Values) files.

- **`ultralytics.YOLO`**: This is the class provided by the Ultralytics library that allows you to work with YOLO models.

Load YOLO

- model = YOLO('yolov8n.pt')
-
- This line loads a YOLO model using the Ultralytics library. It loads a pretrained model named 'yolov8n.pt'.
- Set Folder Path:
- pythonCopy code
- `folder_path = './Ads_problem4'`

- This is the main folder path that contains subfolders with images for comparison.
- Define the **`main()`** Function:
- pythonCopy code
-
```
def main():
    folders_to_process = get_subfolders(folder_path)

    for selected_folder_path in folders_to_process:
        # ... (rest of the code)
```

- This is the main function that coordinates the processing of subfolders. It calls other functions to perform operations on the images.
- Get Subfolders:
- pythonCopy code

```
def get_subfolders(parent_folder):
    subfolders = []
    for item in os.listdir(parent_folder):
        item_path = os.path.join(parent_folder, item)
        if os.path.isdir(item_path):
            subfolders.append(item_path)
    return subfolders
```

This function reads all subfolders within the main folder and returns a list of their paths.
Get Paths of Ideal Images:

pythonCopy code

-
  ```
  def get_ideal_images(folder_path):
      ideal_folder = os.path.join(folder_path, "ideal")
      ideal_images = [os.path.join(ideal_folder, image) for image
  in os.listdir(ideal_folder)]
      return ideal_images
  ```
- This function constructs paths to the images located in the "ideal" subfolder within the given folder path.
- Get Paths of Other Images:
- pythonCopy code

-
  ```
  def get_other_images(folder_path):
      other_folder = os.path.join(folder_path, "other")
      other_images = [os.path.join(other_folder, image) for image
  in os.listdir(other_folder)]
      return other_images
  ```
- This function constructs paths to the images located in the "other" subfolder within the given folder path.
- Process Ideal Images:
- pythonCopy code

-
  ```
  def ideal_images(selected_folder_path):
      all_results = []
      ideal_images = get_ideal_images(selected_folder_path)
      for path in ideal_images:
          results = model.predict(path, save=True)
          all_results.append(results)
      return all_results
  ```
- This function processes ideal images using the YOLO model to predict objects present in the images.
- Count Detected Entities:
- pythonCopy code

-
  ```
  def count_Entities(results):
      counts = {}
      cords = []

      for result in results:
          boxes = result[0].boxes.cpu().numpy()
          cords.extend(result[0].boxes.xyxy.cpu().numpy())
  ```

```
    # ... (rest of the code)
```

- This function counts detected entities (objects) in the YOLO prediction results and collects their coordinates.
- Compare Entities:
- pythonCopy code

-
  ```
  def compare_entities(ideal_results, other_result):
      ideal_counts, ideal_cords = count_Entities(ideal_results)
      other_counts, other_cords = count_Entities(other_result)

      # ... (rest of the code)
  ```
- This function compares the entities (objects) detected in ideal and other images and identifies missing and extra entities.
- Perform Comparison and Write CSV:
- pythonCopy code

-
  ```
  def perform_comparison_and_write_csv(ideal_results, other_path,
  selected_folder_path):
      other_results = model.predict(other_path, save=True)

      # ... (rest of the code)
  ```
- This function performs a comparison between ideal and other images, identifies differences, and writes the comparison results to a CSV file.
- Call Functions Inside **`main()`**:
- pythonCopy code
  ```
  if __name__ == "__main__":
  ```
- This block of code ensures that the **`main()`** function is executed when the script is run directly, not when it's imported as a module.

This script mainly focuses on:

- Reading and processing images from the "ideal" and "other" subfolders.
- Using the YOLO model to predict objects in images.
- Comparing the objects detected in ideal and other images.
- Writing the comparison results to CSV files.

Problem 5:

Object Detection with YOLO - Documentation

Below is the documentation for the Python script using the Ultralytics YOLO library to perform object detection on images in a specified folder:

Code Overview

Import necessary libraries including os, shutil, itertools, matplotlib.pyplot, YOLO from Ultralytics, and clip from OpenAI.

Define the function detect_and_save_objects(yolo_model, input_folder, output_folder) to perform object detection on images and save detected object crops.

Define the function compare_images_using_clip(clip_model, output_folder) to compare images using the CLIP model and save similar object crops.

Define the main function main() to orchestrate the execution of object detection and image comparison.

Call the main() function when the script is run as the main module.

Execution Flow

Create a YOLO model instance using the pretrained model file yolov8m.pt.

Specify input and output directories for images and results.

Call the detect_and_save_objects function to perform object detection and save object crops.

Call the compare_images_using_clip function to compare images and save similar object crops.

Main Execution

Check if the script is being run as the main module using if __name__ == "__main__":

Call the main() function to initiate the execution of object detection and comparison.

Handle exceptions and print error messages in case of failures.

This documentation provides an overview of the code's functionality and execution process.

Here is the Code :

```
import os

import shutil

import itertools

import matplotlib.pyplot as plt

from ultralytics import YOLO

import clip




# Function to perform YOLO object detection and save cropped images

def detect_and_save_objects(yolo_model, input_folder, output_folder):

    try:
        # Get YOLO detections function from the model


        # Get a list of image paths in the input folder

        image_paths = [os.path.join(input_folder, img) for img in
os.listdir(input_folder) if img.lower().endswith(('.jpeg', '.jpg'))]



        for img_path in image_paths:

            # Extract image name without extension

            image_name = os.path.splitext(os.path.basename(img_path))[0]

            # Create output folder for the image's detections

            output_img_folder = os.path.join(output_folder, image_name)

            os.makedirs(output_img_folder, exist_ok=True)



            # Perform object detection using YOLO

            detected_objects = yolo_model.predict(img_path, save=False)

            object_counts = {}  # Track counts of different objects in the image



            for idx, detected_obj in enumerate(detected_objects[0].boxes):

                obj_class = detected_obj.cls[0].item()

                object_counts[obj_class] = object_counts.get(obj_class, 0) + 1
                class_label = detected_objects[0].names[obj_class]

                entity_folder = os.path.join(output_img_folder,
f"{class_label}_{object_counts[obj_class]}")

                os.makedirs(entity_folder, exist_ok=True)



                # Extract object's bounding box coordinates

                x_min, y_min, x_max, y_max = detected_obj.xyxy[0]

                x_min, y_min, x_max, y_max = int(x_min), int(y_min), int(x_max),
int(y_max)



                # Crop the object from the image

                cutout = plt.imread(img_path)[y_min:y_max, x_min:x_max]

                output_filename =
f"{class_label}_{object_counts[obj_class]}_crop.jpg"

                plt.imsave(os.path.join(entity_folder, output_filename), cutout)


    except Exception as e:

        print("Error occurred during object detection and saving:", e)



# Function to compare images using the CLIP model

def compare_images_using_clip(clip_model, output_folder):

    try:

        # Load CLIP model and preprocessing function
        clip_model, clip_preprocess = clip_model.load("ViT-B/32")

        # Get matched image paths for CLIP comparison

        matched_image_paths = [os.path.join(root, file) for root, _, files in
os.walk(output_folder) for file in files if file.endswith('.jpg')]




        for root, _, files in os.walk(output_folder):

            if files:

                # Get the path of the reference image for comparison

                reference_img_path = os.path.join(root, files[0])

                reference_entity = ''.join(filter(lambda z: not z.isdigit(),
files[0].split('_')[0]))

                reference_image =
clip_preprocess(plt.imread(reference_img_path))[None]

                reference_image_features =
clip_model.encode_image(reference_image)

                similarity_scores = {}  # Store similarity scores for images




                for img_path in matched_image_paths:

                    if reference_img_path == img_path:

                        continue

                    target_image = clip_preprocess(plt.imread(img_path))[None]

                    target_image_features = clip_model.encode_image(target_image)

                    # Calculate similarity score using cosine similarity
                    similarity_score = (1 + (target_image_features @
reference_image_features.T) / 2).item()



                    similarity_scores[img_path] = similarity_score



                # Sort the similarity scores and get top similar images

                sorted_scores = dict(sorted(similarity_scores.items(), key=lambda
x: x[1], reverse=True))

                top_similar_images = dict(itertools.islice(sorted_scores.items(),
3))



                print(top_similar_images)

                temp_counter = 1



                for image_path in top_similar_images.keys():

                    # Create new filename for copied similar images

                    new_filename = f"top{temp_counter}_crop.jpeg"

                    destination_path = os.path.join(root, new_filename)

                    shutil.copy(image_path, destination_path)

                    temp_counter += 1


    except Exception as e:
        print("Error occurred during image comparison:", e)


def main():

    try:

        # Initialize YOLO model

        yolo = YOLO("yolov8m.pt")

        input_directory = "./All_Images"

        output_directory = "output/problem5"


        # Call functions to perform object detection and image comparison

        detect_and_save_objects(yolo, input_directory, output_directory)

        compare_images_using_clip(clip, output_directory)


    except Exception as e:

        print("An error occurred during execution:", e)


if __name__ == "__main__":

    main()
```