Image Similarity and OCR-Based Analysis

Overview

This project aims to perform two main tasks:

1. **Image Similarity Calculation**: For sets 1 to 7, the project calculates the similarity between a main image and two test images using a pre-trained VGG16 model.
2. **OCR-Based Value Extraction**: For sets 8 and 9, the project extracts specific numerical values ("TOTAL WIN" and "BET") from two test images using OCR (Optical Character Recognition) with PaddleOCR.

Intuition Behind the Approach

1. **Image Similarity Calculation**:
    - **Model**: We use the VGG16 model pre-trained on ImageNet for feature extraction.
    - **Process**: The images are pre-processed and fed into the VGG16 model to extract feature vectors. Cosine similarity is then calculated between the feature vectors of the main image and the test images to determine their similarity percentages.
2. **OCR-Based Value Extraction**:
    - **Model**: PaddleOCR is used for extracting text from images.
    - **Process**: The OCR model detects text regions in the images, and specific target words ("TOTAL WIN" and "BET") are identified. The nearest numerical value to these target words is extracted and cleaned (commas removed) for further use.
    -

Step-by-Step Guide to Run the Project

1. Project Structure

Ensure your project directory has the following structure:

```
project_directory/
│
├── main.py
├── Handler.py
├── Solution.py
├── ProblemSetList.txt
├── Problems/
│   ├── Set1/
│   │   ├── Image.png
│   │   ├── Test1.png
│   │   └── Test2.png
│   ├── Set2/
│   │   ├── Image.png
│   │   ├── Test1.png
│   │   └── Test2.png
│   └── ...
│       ├── Set8/
│       │   ├── Test1.png
│       │   └── Test2.png
│       ├── Set9/
│       │   ├── Test1.png
│       │   └── Test2.png
│       └── ...
└── Results.txt
```

2. Setting Up the Environment

Ensure you have the necessary libraries installed:

```
pip install certifi paddleocr tensorflow scikit-learn pillow
```

3. Running the Project

Follow these steps to execute the project:

1. **Initialization**:
    - Ensure the `Results.txt` file is empty or non-existent.
    - The `ProblemSetList.txt` file should list all the sets to be processed.
2. **Execution**:
    - Run the `main.py` script to start the process.

```
python main.py
```

4. Output

- The `Results.txt` file will be populated with the results.
    - For sets 1 to 7: Image similarity percentages between the main image and test images.
    - For sets 8 and 9: Extracted numerical values for "TOTAL WIN" and "BET".

Code Files

main.py

```
from Handler

import start, execute

def run():
    start()
    execute()

if __name__ == '__main__':
    run()
```

Handler.py

```
from Solution import Solution

def save(output, filename='Results.txt'):
      with open(filename, 'a') as file:
            file.write(output + '\n')

def start():
      filename = 'Results.txt'
      open(filename, 'w').close()

def execute():
      problem_sets = []
      with open('ProblemSetList.txt', 'r') as file:
             for line in file:
                  problem_sets.append(line.strip())

for problem in problem_sets:
      solution = Solution()
      result = solution.get_answer(problem)
```

```
      if len(result) == 2:
            save(','.join(map(str, result)))
```

Solution.py

```
import certifi
from paddleocr import PaddleOCR
import os
import re
import numpy as np
from PIL import Image
import tensorflow as tf
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model
from sklearn.metrics.pairwise import cosine_similarity

os.environ['SSL_CERT_FILE'] = certifi.where()

class Solution:
    def __init__(self):
        self.base_model = VGG16(weights='imagenet', include_top=False, pooling='avg')
        self.model = Model(inputs=self.base_model.input,
outputs=self.base_model.output)
        self.ocr = PaddleOCR(use_angle_cls=True, lang='en')

    def preprocess_image(self, img_path):
        img = image.load_img(img_path, target_size=(224, 224))
        img_array = image.img_to_array(img)
        img_array = np.expand_dims(img_array, axis=0)
        img_array = preprocess_input(img_array)
        return img_array

    def extract_features(self, img_path):
        img_array = self.preprocess_image(img_path)
        features = self.model.predict(img_array)
        return features

    def calculate_similarity(self, main_image_path, test_image_path1,
test_image_path2):
        main_features = self.extract_features(main_image_path)
        test1_features = self.extract_features(main_image_path)
        test2_features = self.extract_features(test_image_path2)
        similarity1 = cosine_similarity(main_features, test1_features)[0][0]
        similarity2 = cosine_similarity(main_features, test2_features)[0][0]
        percentage1 = similarity1 * 100
        percentage2 = similarity2 * 100
        return [percentage1, percentage2]

    def process_image_sets(self, base_path, set_number):
        main_image_path = f'{base_path}/Set{set_number}/Image.png'
        test_image_path1 = f'{base_path}/Set{set_number}/Test1.png'
        test_image_path2 = f'{base_path}/Set{set_number}/Test2.png'
        return self.calculate_similarity(main_image_path, test_image_path1,
test_image_path2)

    def extract_numerical_value(self, text):
        pattern = r'[-+]?\d{1,3}(?:,\d{3})*(?:\.\d+)?'
        match = re.search(pattern, text)
        if match:
            return match.group().replace(',', '')  # Remove commas from the extracted
number
        else:
            return None

    def find_nearest_numerical_value(self, target_word, ocr_result, max_distance):
        target_box = None
```

```
        for (box, text) in zip(ocr_result['boxes'], ocr_result['texts']):
            if target_word in text:
                target_box = box
                break
        if target_box is None:
            return ""
        target_x, target_y = target_box[0][0], target_box[0][1]
        target_center = np.array([target_x + (target_box[1][0] - target_x) / 2,
                                  target_y + (target_box[2][1] - target_y) / 2])
        closest_value = None
        closest_distance = float('inf')
        for box, text in zip(ocr_result['boxes'], ocr_result['texts']):
            numerical_value = self.extract_numerical_value(text)
            if numerical_value is not None:
                box_x, box_y = box[0][0], box[0][1]
                box_center = np.array([box_x + (box[1][0] - box_x) / 2,
                                       box_y + (box[2][1] - box_y) / 2])
                distance = np.linalg.norm(target_center - box_center)
                if distance < closest_distance and distance <= max_distance:
                    closest_distance = distance
                    closest_value = numerical_value
        return closest_value if closest_value is not None else ""

    def find_nearest_values_for_targets(self, target_words, ocr_result, max_distance):
        results = {}
        for word in target_words:
            results[word] = self.find_nearest_numerical_value(word, ocr_result,
max_distance)
        return results

    def process_images_for_sets(self, base_path, set_number, target_words,
max_distance):
        results = {word: [] for word in target_words}
        image_files = ['Test1.png', 'Test2.png']
        for image_file in image_files:
            img_path = os.path.join(base_path, f'Set{set_number}', image_file)
            if not os.path.exists(img_path):
                print(f"File {img_path} not found. Skipping.")
                continue
            try:
                img = Image.open(img_path)
            except Exception as e:
                print(f"Error loading image {img_path}: {e}")
                continue
            ocr_result = self.ocr.ocr(img_path, cls=True)
            parsed_result = {'boxes': [], 'texts': [], 'scores': []}
            for line in ocr_result:
                for word_info in line:
                    parsed_result['boxes'].append(word_info[0])
                    parsed_result['texts'].append(word_info[1][0])
                    parsed_result['scores'].append(word_info[1][1])
            nearest_values = self.find_nearest_values_for_targets(target_words,
parsed_result, max_distance)
            for word in target_words:
                results[word].append(nearest_values[word])
        return results

    def get_answer(self, problem):
        base_path = './Problems'
        if problem == 'Set8':
            target_words = ["TOTAL WIN"]
            return self.process_images_for_sets(base_path, 8, target_words, 150)["TOTAL
WIN"]
        elif problem == 'Set9':
            target_words = ["BET"]
            return self.process_images_for_sets(base_path, 9, target_words, 150)["BET"]
        else:
```

```
            set_number = int(problem.split('Set')[1])
            return self.process_image_sets(base_path, set_number)
```

ProblemSetList.txt

```
Set1
Set2
Set3
Set4
Set5
Set6
Set7
Set8
Set9
```

Key Points

- **Image Similarity**: Uses VGG16 for feature extraction and cosine similarity for comparison.
- **OCR**: PaddleOCR for text detection and extraction, targeting specific words and nearest numerical values.
- **Flexibility**: The solution is adaptable for different sets based on the problem requirements.

By following this guide, you should be able to run the project and understand the intuition behind the approach. The provided code processes images according to their respective requirements, and the results are saved in a structured manner.