经济管理学院

课　程　报　告

（复杂网络与社会计算）

题　　目：week12 课程作业

课程教师：赵吉昌

学院/专业：信息管理与信息系统

学生姓名：范春

学　　号：21377061

2024 年 5 月 14 日

北京航空航天大學
BEIHANG UNIVERSITY

作业要求如下：

1. 阅读本周参考文献。
2. 从https://data.mendeley.com/datasets/x54j8vxdmp/2下载相关评论数据，包括图像数据和评论其他属性。
(1) 利用detectron2或mmlab得到图像物体检测结果；
(2) 构建物体共现网络，并将其进行可视化，并观察随着共现阈值增加（小于阈值的边将被删除），该网络结构的变化；
(3) 从物体检测及文本的视角，计算图文一致性，结合表格数据，分析该指标和评论有用性的相关性；
(4) 思考在大模型背景下，有无其他度量图文一致性的方法?

0、数据抽取（由于电脑配置问题，任务 1 如果运行 5000 条数据会内存错误，所以最终之抽取了 3000 条数据）

```
import json
import random
from PIL import Image
import os

def check_image(img_path):
    try:
        with Image.open(img_path) as img:
            img.verify()
        return True
    except Exception:
        return False
def select_and_save(input_file, output_file, base_path):
    with open(input_file, 'r', encoding='utf-8') as f:
        data = json.load(f)

    pic_paths = data.get('pic_path',{})
    all_keys = list(pic_paths.keys())

    i = 0
    valid_keys = []
    for key in all_keys:
        if (i<3000):
            print(i)
            original_path = pic_paths[key]
            new_path = original_path.replace('I:', base_path, 1)
            if os.path.exists(new_path) and check_image(new_path):
                valid_keys.append(key)
                i +=1
            else:
                print(f"Invalid or non-existent image at path: {new_path}")
    selected_data = {top_key: {sub_key: value[sub_key] for sub_key in valid_keys} for top_key, value
in data.items()}
    with open(output_file, 'w', encoding='utf-8') as f:
        json.dump(selected_data, f, ensure_ascii=False, indent=4)
```

```
input_file = "C:\\Users\\范春\\Desktop\\week12\\tabular data\\target_comment_seed2021.json"
output_file = 'C:\\Users\\范春\\Desktop\\week12\\sample.json'
base_path = "C:\\Users\\范春\\Desktop\\20201016ImgData\\20201016ImgData"
select_and_save(input_file, output_file, base_path)
```

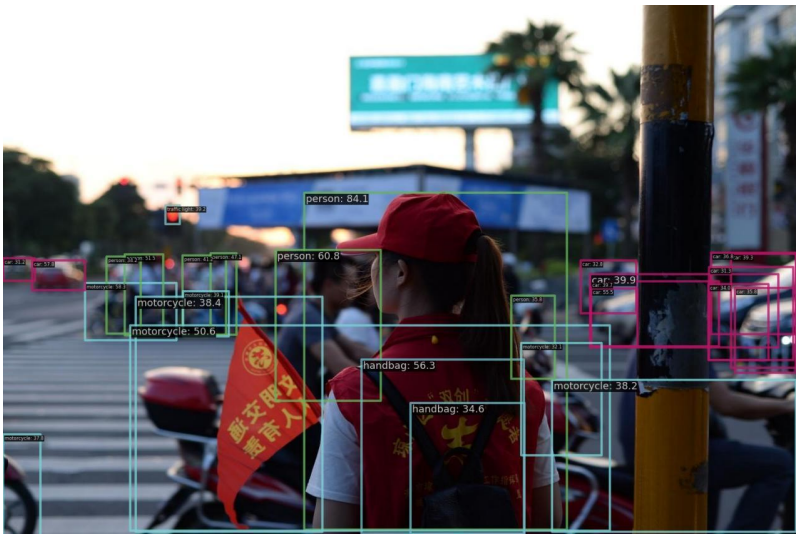
1、图像物体检测

代码如下：

```
import json
import os
from mmengine.logging import print_log
from mmdet.apis import DetInferencer
from mmdet.evaluation import get_classes

def run_simple_image_inference(input_json, output_dir, output_json, model_config, weights,
device, base_path, alpha):
    classes = get_classes('coco')

    with open(input_json, 'r', encoding='utf-8') as f:
        input_data = json.load(f)
    image_counts = input_data.get('imageCount', {})

    with open(output_json, 'w', encoding='utf-8') as f_out:
        i = 0
        for pic_id, pic_path in input_data['pic_path'].items():
            pic_path = pic_path.replace('I:', base_path, 1)
            i += 1
            inferencer = DetInferencer(
                model=model_config,
                weights=weights,
                device=device,
                palette=None
            )
            labels_count = [0] * len(classes)
            num_images = image_counts.get(pic_id, 1)
            for img_index in range(num_images):
                img_path = pic_path.replace('\\0.jpg', f'\\{img_index}.jpg')
                if not os.path.exists(img_path):
                    print(f"Image file {img_path} does not exist. Skipping...")
                    continue
                try:
                    result = inferencer(
                        inputs=img_path,
                        out_dir=os.path.join(output_dir, pic_id),
                        show=False,
                        no_save_vis=False,
```

```
                    no_save_pred=False,
                    print_result=True,
                    batch_size=1,
                    pred_score_thr=0.3
                )
            except Exception as e:
                print(f"Error processing image {img_path}: {str(e)}")
                continue
            for item in result['predictions']:
                for label, score in zip(item['labels'], item['scores']):
                    if score > alpha:  # Filter based on threshold
                        labels_count[label] += 1
        labels_binary = [1 if count > 0 else 0 for count in labels_count]
        results_dict = dict(zip(classes, labels_binary))
        f_out.write(f'{pic_id}: {json.dumps(results_dict)}\n')
        print(f'{i} is ok')
if __name__ == '__main__':
    input_json_path = "C:\\Users\\范春\\Desktop\\week12\\sample.json"
    output_directory = 'C:\\Users\\范春\\Desktop\\week12\\output'
    output_json_path = 'C:\\Users\\范春\\Desktop\\week12\\output\\data\\output0.txt'
    model_config_path =
"C:\\MMDetection_GPU\\mmdetection-main\\mmdetection-main\\rtmdet_tiny_8xb32-300e_coco.py"
    weights_path =
"C:\\MMDetection_GPU\\mmdetection-main\\mmdetection-main\\rtmdet_tiny_8xb32-300e_coco_20220902
_112414-78e30dcc.pth"
    device_type = 'cpu'
    base_data_path = "C:\\Users\\范春\\Desktop\\20201016ImgData\\20201016ImgData"
    alpha_value = 0  # Not filtering here, to be filtered when building the co-occurrence network
    run_simple_image_inference(input_json_path, output_directory, output_json_path,
model_config_path, weights_path, device_type, base_data_path, alpha_value)
```

结果示例：



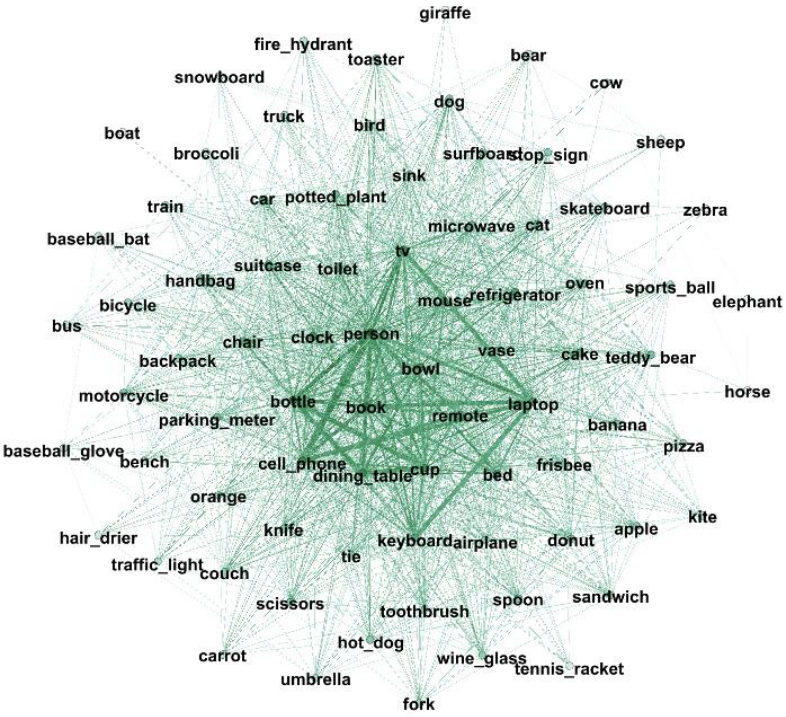
2、构建物体共现网络并可视化

代码如下：

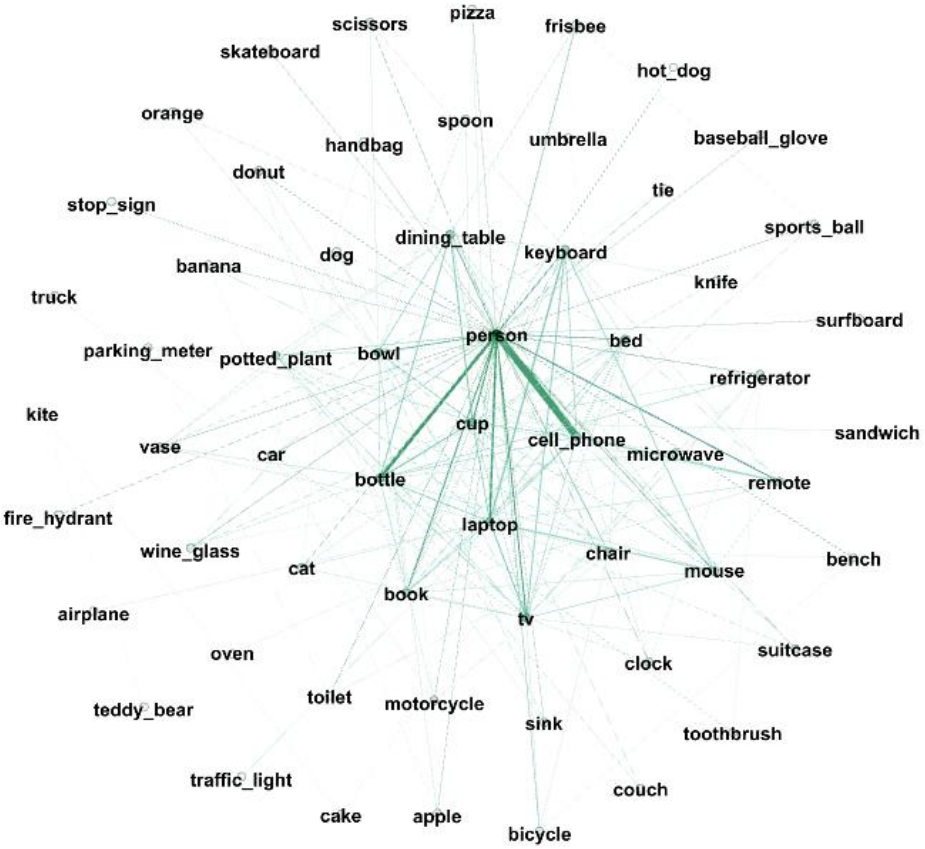
```
import json
import os
from collections import defaultdict
import itertools
import networkx as nx
import matplotlib.pyplot as plt
from mmdet.evaluation import get_classes

class ObjectCoOccurrenceNetwork:
    def __init__(self, inference_output_dir, output_file, score_threshold):
        self.inference_output_dir = inference_output_dir
        self.co_occurrence_matrix = defaultdict(lambda: defaultdict(int))
        self.classes = get_classes('coco')
        self.output_file = output_file
        self.score_threshold = score_threshold
    def parse_inference_results(self):
        for root, _, files in os.walk(self.inference_output_dir):
            for file in files:
                if file.endswith('.json'):
                    file_path = os.path.join(root, file)
                    with open(file_path, 'r', encoding='utf-8') as f:
                        result = json.load(f)
                        labels = result.get("labels", [])
                        scores = result.get("scores", [])
                        filtered_labels = [label for label, score in zip(labels, scores) if score >
self.score_threshold]
                        self.update_co_occurrence(filtered_labels)
    def update_co_occurrence(self, labels):
        unique_labels = set(labels)
        for label1, label2 in itertools.combinations(unique_labels, 2):
            self.co_occurrence_matrix[label1][label2] += 1
            self.co_occurrence_matrix[label2][label1] += 1
    def save_co_occurrence_matrix(self):
        with open(self.output_file, 'w', encoding='utf-8') as f:
            json.dump(self.co_occurrence_matrix, f, ensure_ascii=False, indent=4)
    def display_network(self):
        G = nx.Graph()
        for label1, related_labels in self.co_occurrence_matrix.items():
            for label2, count in related_labels.items():
                if count > 0:
                    G.add_edge(self.classes[label1], self.classes[label2], weight=count)
        pos = nx.spring_layout(G)
```

```
        edge_labels = nx.get_edge_attributes(G, 'weight')
        nx.draw(G, pos, with_labels=True, node_color='skyblue', node_size=7000, font_size=10,
font_weight='bold')
        nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)
        plt.show()
    def save_network_gexf(self, gexf_path):
        G = nx.Graph()
        for label1, related_labels in self.co_occurrence_matrix.items():
            for label2, count in related_labels.items():
                if count > 0:
                    G.add_edge(self.classes[label1], self.classes[label2], weight=count)
        largest_cc = max(nx.connected_components(G), key=len)
        H = G.subgraph(largest_cc).copy()
        nx.write_gexf(H, gexf_path)
if __name__ == '__main__':
    inference_output_dir = 'C:\\Users\\范春\\Desktop\\week12\\output'
    output_file = 'C:\\Users\\范春\\Desktop\\week12\\output\\co_occurrence_matrix.json'
    gexf_path = 'C:\\Users\\范春\\Desktop\\week12\\output\\network0_3.gexf'
    score_threshold = 0.3
    co_occur_network = ObjectCoOccurrenceNetwork(inference_output_dir, output_file,
score_threshold)
    co_occur_network.parse_inference_results()
    co_occur_network.save_co_occurrence_matrix()
    co_occur_network.display_network()
    co_occur_network.save_network_gexf(gexf_path)
```

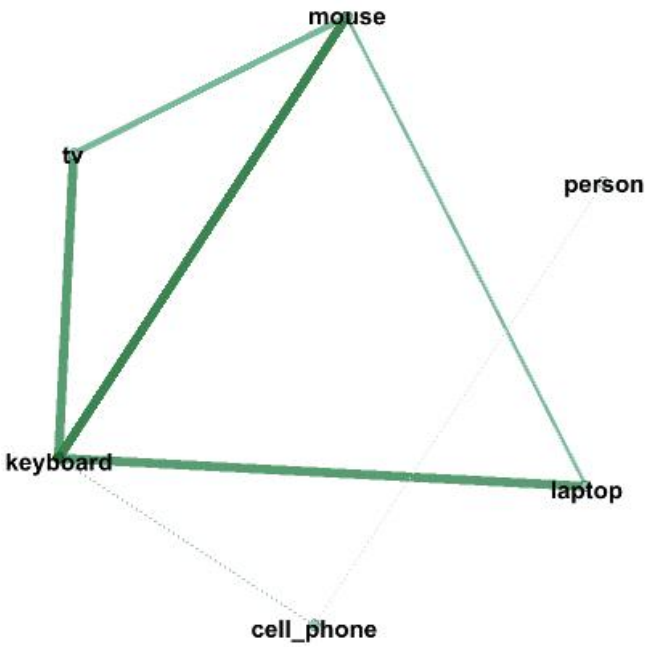
(1) 阈值为 0.3



(2) 阈值为 0.5



(3) 阈值为 0.8



3、一致性分析

代码如下：

```
import json
import os
import re
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from mmdet.evaluation import get_classes

class ImageTextConsistency:
    def __init__(self, inference_output_dir, sample_json_path, score_threshold):
        self.inference_output_dir = inference_output_dir
        self.sample_json_path = sample_json_path
        self.score_threshold = score_threshold
        self.classes = get_classes('coco')
        self.image_results = {}
        self.comments = {}
    def load_sample_json(self):
        with open(self.sample_json_path, 'r', encoding='utf-8') as f:
            data = json.load(f)
            self.comments = data['content']
            self.image_paths = data['pic_path']
    def load_inference_results(self):
        for root, _, files in os.walk(self.inference_output_dir):
            for file in files:
                if file.endswith('.json'):
                    file_path = os.path.join(root, file)
                    with open(file_path, 'r', encoding='utf-8') as f:
                        result = json.load(f)
                        labels = result.get("labels", [])
                        scores = result.get("scores", [])
                        filtered_labels = [label for label, score in zip(labels, scores) if score >
self.score_threshold]
                        pic_id = os.path.basename(file_path).split('.')[0]
                        self.image_results[pic_id] = filtered_labels
    def calculate_consistency(self, comment, detected_labels):
        detected_objects = set([self.classes[label] for label in detected_labels])
        words = set(re.findall(r'\b\w+\b', comment.lower()))
        common_objects = detected_objects & words
        return len(common_objects) / len(detected_objects) if detected_objects else 0
    def analyze_consistency(self):
        consistencies = []
        for pic_id, comment in self.comments.items():
            detected_labels = self.image_results.get(pic_id, [])
            consistency = self.calculate_consistency(comment, detected_labels)
            consistencies.append((pic_id, consistency))
        return consistencies
    def save_consistency_results(self, output_file, consistencies):
```

```
        df = pd.DataFrame(consistencies, columns=['pic_id', 'consistency'])
        df.to_csv(output_file, index=False)
if __name__ == '__main__':
    inference_output_dir = 'C:\\Users\\范春\\Desktop\\week12\\output'
    sample_json_path = 'C:\\Users\\范春\\Desktop\\week12\\sample.json'
    output_consistency_file = 'C:\\Users\\范春
\\Desktop\\week12\\output\\consistency_results.csv'
    score_threshold = 0.6
    consistency_analyzer = ImageTextConsistency(inference_output_dir, sample_json_path,
score_threshold)
    consistency_analyzer.load_sample_json()
    consistency_analyzer.load_inference_results()
    consistencies = consistency_analyzer.analyze_consistency()
    consistency_analyzer.save_consistency_results(output_consistency_file, consistencies)
```

通过上述代码计算了图文一致性，但结果显示均为 0，原因有待进一步思考。

4、在大模型背景下，其他度量图文一致性的方法

(1) 基于语义相似度的度量：利用大型预训练模型（如 BERT、GPT 等）提取文本和图像的语义表示，并计算它们之间的相似度。可以使用文本嵌入和图像特征之间的余弦相似度或其他相似性度量来衡量它们之间的一致性。

(2) 多模态表示学习：通过训练一个联合的多模态模型，将文本和图像嵌入到一个共同的表示空间中。然后，可以使用这个共同的表示空间来衡量图像和文本之间的一致性，例如计算它们之间的距离或相似性。