图片检索评估

笔记本:	我的第一个笔记本		
创建时间:	2024/11/29 9:08	**更新时间:**	2024/11/29 12:08
作者:	耿介		
URL:	about:blank		

计算评估指标:

Precision@K: 前 K 个检索结果中相关结果的比例。

Recall@K: 前 K 个检索结果中成功召回商品信息的概率。

Retrieval_time: 平均每张图片的检索时间

在本应用中只需要正确召回图片并且给出商品信息就算是成功实现功能，因此不需要计算f1-score，map等指标

代码如下:

```
import os
import cv2
import numpy as np
from tqdm import tqdm  # 用于显示进度条
from app_figure_search import search_image




def get_label_prefix(s):
    return s.split(' ')[0]




def evaluate_system(dataset_folder, K=10):
    """
    评估图像检索系统的性能。


    参数:
    - dataset_folder: 存放测试图片的文件夹路径
    - K: 评估指标中的前K个检索结果


    返回:
    - 包含评估指标的字典
    """
    # 存储每个查询的评估指标
    all_precisions = []
    all_average_precisions = []
```

```
    all_recalls = []


    # 获取测试集中的所有图片文件
    image_files = [f for f in os.listdir(dataset_folder) if f.endswith('.jpg')]


    # 遍历每张测试图片
    for image_file in tqdm(image_files, desc="Evaluating"):
        # 构建完整的图片路径
        image_path = os.path.join(dataset_folder, image_file)

        # 读取图片
        image = cv2.imread(image_path)
        #print("正在查询图片")
        if image is None:
            print(f"无法读取图片：{image_path}")
            continue

        # 将图片转换为 RGB 格式
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        # 从文件名中获取真实标签（去掉 '.jpg' 后缀）
        ground_truth_label = os.path.splitext(image_file)[0]

        # 使用现有的检索函数获取检索结果
        output_images, output_paths_str = search_image(image)

        # 从检索结果中提取标签
        # 假设 output_paths_str 的格式为：
        # "第1种可能的产品型号为： label1\n第2种可能的产品型号为： label2\n..."
        retrieved_labels = []
        lines = output_paths_str.strip().split('\n')
        for line in lines[:K]:  # 只考虑前 K 个结果
            # 提取每行中的标签
            parts = line.split('： ')
            if len(parts) == 2:
                label = parts[1].strip()
                retrieved_labels.append(label)
            else:
                print(f"无法解析行：{line}")

        # 判断每个检索结果是否与真实标签匹配
        relevant_recall = [1 if label == ground_truth_label else 0 for label in
retrieved_labels]
        relevant_pre = [
            1 if get_label_prefix(label) == get_label_prefix(ground_truth_label)
else 0
            for label in retrieved_labels
        ]


        # 计算 Precision@K
        precision_at_k = sum(relevant_pre) / K
        all_precisions.append(precision_at_k)

        # 计算 Recall@K
        # 假设每个查询在数据库中只有一个相关项
        total_relevant = 1  # 如果有多个相关项，请相应调整
        recall_at_k = sum(relevant_recall) / total_relevant
        all_recalls.append(recall_at_k)

        # 计算 Average Precision (AP)
        num_relevant = 0
        precisions = []
        for idx, rel in enumerate(relevant_recall):
            if rel:
                num_relevant += 1
                precisions.append(num_relevant / (idx + 1))
        if precisions:
```

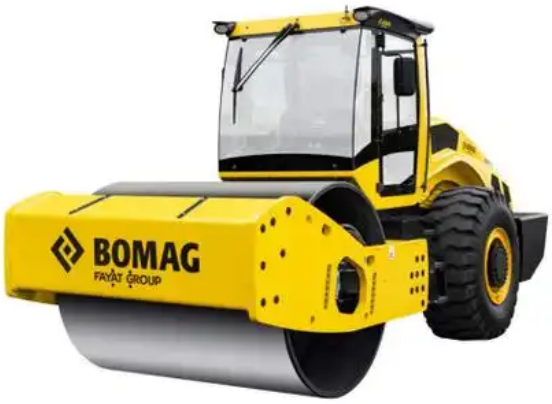
```
            average_precision = sum(precisions) / total_relevant
        else:
            average_precision = 0.0
        all_average_precisions.append(average_precision)

    # 计算平均指标
    mean_precision_at_k = np.mean(all_precisions)
    mean_recall_at_k = np.mean(all_recalls)
    mAP = np.mean(all_average_precisions)

    print(f"平均 Precision@{K}: {mean_precision_at_k:.4f}")
    print(f"平均 Recall@{K}: {mean_recall_at_k:.4f}")
    print(f"平均平均精度 (mAP): {mAP:.4f}")

    return {
        'mean_precision_at_k': mean_precision_at_k,
        'mean_recall_at_k': mean_recall_at_k,
        'mean_average_precision': mAP
    }


# 示例用法:
if __name__ == "__main__":
    dataset_folder = '/mnt/上海市交通系统交易问答框架/Figure_search/datatest'  # 将此处替换为您的 dataset 文件夹路径
    evaluate_system(dataset_folder, K=10)
```

验证数据集准备

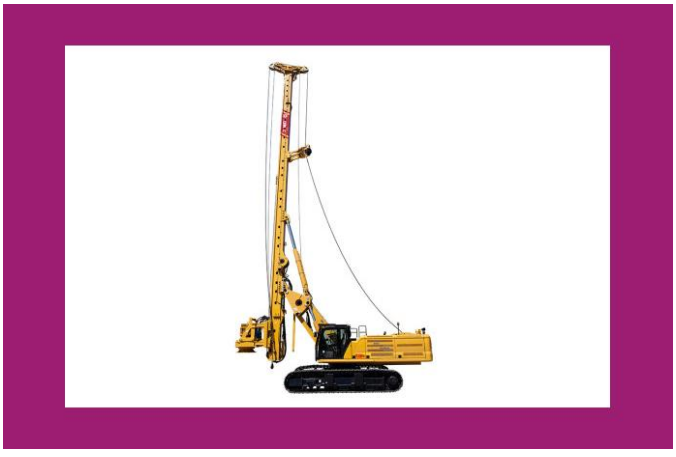
一部分数据是真实的商品图片（带有背景），由于这个数据太少，补充了人为的带有破坏性的数据，比如背景随机扩充，随机翻转，随机增加噪声

干扰性验证集数据

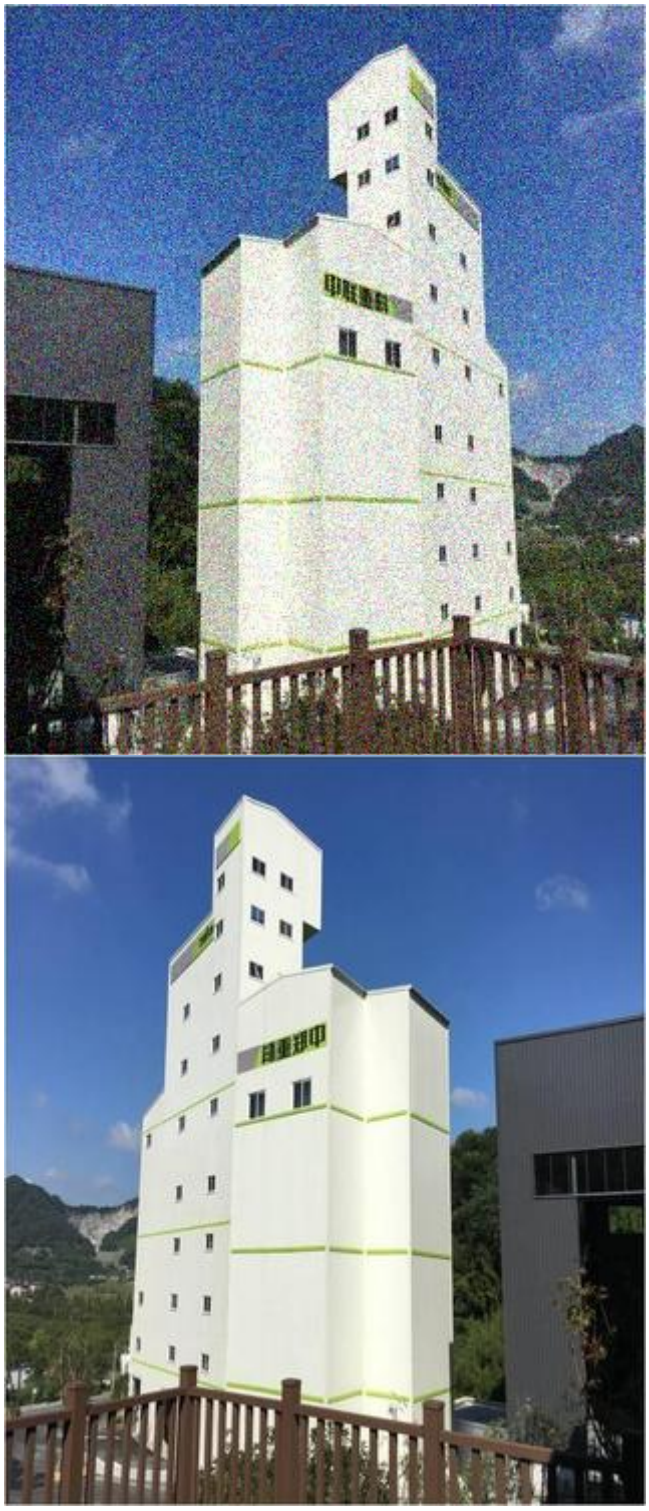
数据集中商品图片 vs 真实场景商品图片：



数据集中商品图片 vs 背景填充商品图片：



水平翻转和噪声示例：



直接训练后的评估指标

平均 Precision@10: 0.2324

平均 Recall@10: 0.3826
平均检索时间: 0.4057秒

对于有干扰的商品图片，召回概率很低，说明在训练时没有真正提取出图片中鲁棒性的特征

因此在重新训练模型，对于训练集进行数据增强，包括水平翻转和随机增加噪声，逼迫模型学习鲁棒性特征而非表面特征。

数据增强

修改了dataset_v2.py, 加入翻转噪声

代码如下：

```
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True
from torchvision.transforms import functional as F
import torch


class DataSet_V2:
    def __init__(self, root_dir, batch_size, shuffle, num_workers, istrainning):
        super(DataSet_V2, self).__init__()
        self.istrainning = istrainning
        self.dataset = datasets.ImageFolder(root=root_dir,
                                            transform=self.get_transforms())
        self.loader = DataLoader(
            dataset=self.dataset,
            batch_size=batch_size,
            shuffle=shuffle,
            num_workers=num_workers,
            prefetch_factor=None if num_workers == 0 else num_workers *
batch_size
        )


    def get_transforms(self):
        if not self.istrainning:
            return transforms.Compose([
                transforms.Lambda(self.convert_img),
                transforms.ToTensor(),
                transforms.Resize(size=(224, 224), antialias=True),
            ])
        else:
            return transforms.Compose([
                transforms.Lambda(self.convert_img),
                transforms.RandomHorizontalFlip(p=0.4),
                transforms.Lambda(lambda img: self.add_random_noise(img)),
                transforms.ToTensor(),
                transforms.Resize(size=(224, 224), antialias=True),
            ])


    @staticmethod
    def add_random_noise(img):
        """
```

```
        向图像添加随机噪声
        :param img: PIL.Image 输入图像
        :return: 含噪声的 PIL.Image 图像
        """
        img_tensor = F.to_tensor(img)
        noise = torch.randn_like(img_tensor) * 0.1
        noisy_img_tensor = img_tensor + noise
        noisy_img_tensor = torch.clamp(noisy_img_tensor, 0, 1)
        return F.to_pil_image(noisy_img_tensor)


    @staticmethod
    def convert_img(img):
        return img.convert("RGB")


    def __len__(self):
        return len(self.dataset.imgs)


    def __iter__(self):
        for data in self.loader:
            yield data



if __name__ == '__main__':
    batch_size = 8
    num_workers = 0
    train_dataset = DataSet_V2('./data/train', batch_size, True, num_workers,
istrainning=True)
    test_dataset = DataSet_V2('./data/test', batch_size, False, num_workers,
istrainning=False)


    for inputs, labels in train_dataset:
        print(labels[0].item())
```

重新训练后得到新的best.pth,计算出新的pca权重，将数据重新存入数据库，collection的名字为image_final

数据增强训练后的评估指标

平均 Precision@10: 0.4660
平均 Recall@10: 0.9717
平均检索时间: 0.4073 秒

这个结果就是正确检索图片的概率是百分之97，召回的十张图片有46%的概率是相关的商品。这个结果是满意的。

这个检索方式（提取特征和降维）牺牲了precision和少量的召回率，但是获得了极短的检索时间，这个交换是值得的。