**浙江工商大学计算机科学与技术学院**

**《视觉计算方向专业实践》报告**

**专 业： 计算机科学与技术**

**班 级： 计科2203班**

**学 号： 2212190505**

**姓 名： 王倩**

**2025 年 3月14日**

1. 环境搭建

用给定的图像完成SIFT,ORB或其他基于关键点的特征提取，将特征通过聚类算法得出很多聚类中心，形成字典，每个特征能在字典中找到最相似的聚类中心，统计这些聚类中心出现的次数，得到向量后，通过对比向量距离，完成图像的匹配，并比较两种算法的时间性能。设计GUI界面，在界面上可以选择待匹配图像，在给定路径下的图像中找出最匹配的图像并显示在界面上，界面上还可以展示匹配时间等实验结果。

1. 代码

import sys  
import cv2  
import os  
import numpy as np  
import time  
from sklearn.cluster import KMeans # 使用sklearn中的KMeans  
from PyQt5.QtCore import Qt  
from PyQt5.QtGui import QPixmap, QImage  
from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QPushButton, QLabel, QFileDialog  
# 限制线程数，避免内存泄漏警告  
os.environ["OMP\_NUM\_THREADS"] = "1"  
  
class ImageMatcher(QWidget):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
  
 self.setWindowTitle("图像匹配")  
 self.setGeometry(100, 100, 800, 600)  
  
 self.layout = QVBoxLayout()  
  
 # 按钮和标签  
 self.select\_folder\_button = QPushButton("选择文件夹")  
 self.select\_folder\_button.clicked.connect(self.select\_folder)  
 self.layout.addWidget(self.select\_folder\_button)  
  
 self.select\_button = QPushButton("选择待匹配图像")  
 self.select\_button.clicked.connect(self.select\_image)  
 self.layout.addWidget(self.select\_button)  
  
 self.result\_label = QLabel("匹配结果将显示在这里")  
 self.layout.addWidget(self.result\_label)  
  
 self.time\_label = QLabel("匹配时间：")  
 self.layout.addWidget(self.time\_label)  
  
 self.setLayout(self.layout)  
  
 self.folder\_path = None # 用来存储选择的文件夹路径  
 self.dictionary = None # 视觉词典  
  
 def select\_folder(self):  
 # 选择文件夹路径  
 folder = QFileDialog.getExistingDirectory(self, "选择文件夹")  
 if folder:  
 self.folder\_path = folder  
 print(f"已选择文件夹：{self.folder\_path}")  
 else:  
 print("未选择文件夹")  
  
 def select\_image(self):  
 try:  
 if not self.folder\_path:  
 self.result\_label.setText("请先选择文件夹！")  
 return  
  
 # 选择待匹配图像  
 options = QFileDialog.Options()  
 file, \_ = QFileDialog.getOpenFileName(self, "选择图像", self.folder\_path, "Images (\*.png \*.xpm \*.jpg);;All Files (\*)", options=options)  
  
 if not file:  
 print("未选择文件")  
 return  
  
 # 选择待匹配的图像  
 img1 = cv2.imread(file, cv2.IMREAD\_GRAYSCALE)  
 if img1 is None:  
 print(f"无法读取图像: {file}")  
 return  
  
 # 如果还没有构建视觉词典，则从文件夹中构建词典  
 if self.dictionary is None:  
 self.build\_dictionary(self.folder\_path)  
  
 # 使用ORB提取特征并生成向量  
 start\_time = time.time()  
 img1\_vector = self.get\_feature\_vector(img1)  
 end\_time = time.time()  
 match\_time = end\_time - start\_time  
  
 # 遍历文件夹中的所有图像，计算匹配分数  
 best\_match\_image, best\_match\_score = self.find\_best\_match(img1\_vector, self.folder\_path)  
  
 # 显示最匹配的图像  
 self.show\_result(best\_match\_image, best\_match\_score, match\_time)  
  
 except Exception as e:  
 print(f"错误：{str(e)}")  
 self.result\_label.setText(f"发生错误: {str(e)}")  
  
 def build\_dictionary(self, folder\_path, num\_clusters=50):  
 """  
 从给定文件夹中的图像中提取特征并使用KMeans聚类构建视觉词典。  
 """  
 sift = cv2.SIFT\_create()  
 all\_descriptors = []  
  
 for filename in os.listdir(folder\_path):  
 file\_path = os.path.join(folder\_path, filename)  
 if not filename.lower().endswith(('.png', '.jpg', '.jpeg')):  
 continue  
 img = cv2.imread(file\_path, cv2.IMREAD\_GRAYSCALE)  
 if img is None:  
 continue  
 kp, des = sift.detectAndCompute(img, None)  
 if des is not None:  
 all\_descriptors.append(des)  
  
 all\_descriptors = np.vstack(all\_descriptors) # 将所有描述符合并  
 print(f"提取了 {len(all\_descriptors)} 个特征描述符，开始聚类...")  
  
 # 使用KMeans聚类  
 kmeans = KMeans(n\_clusters=num\_clusters) # 使用sklearn的KMeans  
 kmeans.fit(all\_descriptors)  
  
 # 保存词典  
 self.dictionary = kmeans.cluster\_centers\_  
  
 def get\_feature\_vector(self, img):  
 """  
 提取图像特征并生成与视觉词典的向量表示  
 """  
 sift = cv2.SIFT\_create()  
 kp, des = sift.detectAndCompute(img, None)  
  
 if des is None:  
 return np.zeros(len(self.dictionary)) # 如果没有特征，返回零向量  
  
 # 为每个描述符找到最近的聚类中心  
 labels = KMeans(n\_clusters=len(self.dictionary)).fit\_predict(des)  
  
 # 统计每个聚类中心的出现次数  
 feature\_vector = np.zeros(len(self.dictionary))  
 for label in labels:  
 feature\_vector[label] += 1  
  
 return feature\_vector  
  
 def find\_best\_match(self, img1\_vector, folder\_path):  
 """  
 在文件夹中找到与图像特征向量最匹配的图像  
 """  
 best\_match\_score = float('inf')  
 best\_match\_image = None  
 best\_match\_img\_path = None  
  
 # 遍历文件夹中的所有图像文件  
 for filename in os.listdir(folder\_path):  
 file\_path = os.path.join(folder\_path, filename)  
 if not filename.lower().endswith(('.png', '.jpg', '.jpeg')):  
 continue  
  
 img = cv2.imread(file\_path, cv2.IMREAD\_GRAYSCALE)  
 if img is None:  
 continue  
  
 # 获取图像的特征向量  
 img\_vector = self.get\_feature\_vector(img)  
  
 # 计算欧氏距离  
 score = np.linalg.norm(img1\_vector - img\_vector)  
  
 if score < best\_match\_score:  
 best\_match\_score = score  
 best\_match\_image = img  
 best\_match\_img\_path = file\_path  
  
 return best\_match\_image, best\_match\_score  
  
 def show\_result(self, best\_match\_image, best\_match\_score, match\_time):  
 """  
 在界面上展示最匹配的图像和匹配得分  
 """  
 # 转换为RGB格式  
 best\_match\_rgb = cv2.cvtColor(best\_match\_image, cv2.COLOR\_BGR2RGB)  
  
 # 将OpenCV图像转换为QPixmap  
 match\_pixmap = QPixmap.fromImage(QImage(best\_match\_rgb.data, best\_match\_rgb.shape[1], best\_match\_rgb.shape[0], best\_match\_rgb.strides[0], QImage.Format\_RGB888))  
  
 # 创建标签来显示图像  
 match\_label = QLabel(self)  
 match\_label.setPixmap(match\_pixmap)  
  
 # 清空旧的内容并显示新的结果  
 self.result\_label.setText(f"最匹配图像（匹配得分：{best\_match\_score:.2f}）")  
 self.layout.addWidget(match\_label)  
  
 # 显示匹配时间  
 self.time\_label.setText(f"匹配时间：{match\_time:.4f}秒")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app = QApplication(sys.argv)  
 window = ImageMatcher()  
 window.show()  
 sys.exit(app.exec\_())

1. 运行结果

