人工智能学院

机器视觉技术实验报告

# 基于传统方法的手势识别

学院：人工智能学院

专业：智能科学与技术

姓名：潘一铭

学号：2111368

# 目录

# 1 实验目的

使用$HOG$（方向梯度直方图）特征提取与$SVM$（支持向量机）分类器相结合的方法，实现手势识别任务。探索基于计算机视觉技术的手势识别方法，并评估其在实际应用中的性能。

# 2 基于传统方法手势识别原理

## 2.1 $HOG$特征检测原理

HOG（方向梯度直方图）特征是一种用于图像识别和目标检测的特征描述子。它的实现原理如下：

- 图像预处理：将输入的图像转换为灰度图像，并加入去噪，平滑，提取前景处理。
- 计算图像梯度：使用一维差分滤波器在水平和垂直方向上计算图像的梯度。划分图像为小区域：将图像划分为连续的小区域，每个小区域包含若干个像素。
- 计算梯度直方图：对于每个小区域内的像素，根据其梯度幅值和方向，统计梯度方向的分布情况
- 归一化梯度直方图：将每个小区域的梯度直方图进行归一化，以消除光照等因素对特征的影响。
- 构建特征向量：将所有小区域的归一化梯度直方图串联起来，形成最终的特征向量。

HOG 特征的优势在于它对于目标的形状和纹理有较好的描述能力。在目标检测任务中，可以将 HOG 特征与机器学习算法（如支持向量机）相结合，进行目标的训练和分类。通过训练得到的模型，可以在新的图像中进行目标检测和识别，从而实现手势识别等应用。

## 2.2 $SVM$分类器原理

支持向量机（Support Vector Machine，SVM）是一种常用的监督学习算法，用于分类和回归任务。SVM 的分类器原理如下：

- 特征映射：SVM 通过将输入的特征向量映射到高维特征空间中来解决非线性分类问题。这种映射可以通过核函数来实现，常用的核函数有线性核、多项式核和径向基函数（RBF）核等。

- 寻找最优超平面：目标是找到一个最优的超平面，将正类和负类样本分隔开来，并使得两类样本中距离超平面最近的样本点到超平面的间隔最大化。
- 最大化间隔：SVM 通过最大化支持向量到超平面的间隔来提高分类器的泛化能力。
- 软间隔与正则化：在实际应用中，数据往往不是完全线性可分的，此时可以引入软间隔，允许一些样本点被错分。同时，通过正则化项来平衡间隔最大化和误分类的权重，以避免过拟合问题。
- 预测与分类：对于新的未标记样本，利用训练得到的 SVM 模型，通过计算其特征向量与超平面的距离，来进行分类预测。

SVM 具有较好的泛化性能和对高维数据的处理能力，适用于线性和非线性分类问题。它在图像分类、文本分类、手写体识别等领域有广泛应用，并且可以通过调整核函数和正则化参数进行灵活的模型优化。

# 3 实验步骤

## 3.1 导入所需的库与头文件

## 3.2 数据准备阶段

- 设置手势类别和图像数据的路径
- 初始化训练集和测试集的数据结构
- 加载训练集图像和分配标签

## 3.3 图像处理阶段

- 定义读取图像和计算 HOG 特征的函数

此处利用cvtColor(resizedImage, grayscaleImage, COLOR_BGR2GRAY)输入图片转化为灰度图，fastNlMeansDenoising去噪，GaussianBlur加入高斯平滑，threshold设置阈值提取前景

## 3.4 训练集处理部分

- 提取训练集的 HOG 特征
- 创建 SVM 分类器并进行训练
- 对训练集进行预测并计算准确率

## 3.5 测试集处理部分

- 提取测试集的 HOG 特征，并对对测试集进行预测并计算准确率

# 4 程序代码

## 4.1 图像预处理

```
Mat preprocessImage(const Mat &inputImage)
{
    Mat resizedImage, grayscaleImage;
    resize(inputImage, resizedImage, Size(128, 128));
    cvtColor(resizedImage, grayscaleImage, COLOR_BGR2GRAY);
    // 去噪
    fastNlMeansDenoising(grayscaleImage, grayscaleImage);
    // 平滑
    GaussianBlur(grayscaleImage, grayscaleImage, Size(5, 5), 0,
0);
    // 提取前景
    threshold(grayscaleImage, grayscaleImage, 0, 255,
THRESH_BINARY + THRESH_OTSU);
    return grayscaleImage;
}
```

## 4.2 图像读取

```
vector<Mat> readImagesFromFolder(const string &folderPath)
{
    vector<string> extensions = {".jpg", ".jpeg", ".png", ".ppm"};
    vector<Mat> images;
    for (const auto &ext : extensions)
    {
        vector<String> fileNames;
        glob(folderPath + "/*" + ext, fileNames, false);
        for (const auto &fileName : fileNames)
        {
            Mat img = imread(fileName, IMREAD_COLOR);
            if (!img.empty())
            {
                images.push_back(preprocessImage(img));
            }
            else
            {
```

```
                cerr << "Error loading image: " << fileName << endl;
            }
        }
    }
    return images;
}
```

### 4.3 *HOG*特征提取

```
vector<float> computeHOGFeatures(const Mat &image)
{
    // 参数设置
    Size winSize(128, 128);
    Size blockSize(8 * 16, 8 * 16); // 3倍单元格大小
    Size blockStride(16, 16);      // 与单元格大小相同
    Size cellSize(8, 8);
    int numBins = 9;

    HOGDescriptor hog(winSize, blockSize, blockStride, cellSize,
numBins);

    vector<float> descriptors;
    hog.compute(image, descriptors);

    return descriptors;
}
```

### 4.4 数据读取与标签化

```
vector<string> categories = {"A", "C", "Five", "V"};
    string basePath_Train = "/Users/yimimgpan/Desktop/Marcel-
Train/";
    string basePath_Test
="/Users/yimimgpan/Desktop/Hand_Posture_Easy_Stu/";
    vector<vector<Mat>> dataset_Train;
    vector<vector<Mat>> dataset_Test;

    vector<int> labels;
```

```
    // 初始化训练集和测试集
    vector<vector<Mat>> trainDataset(categories.size());
    vector<vector<Mat>> testDataset(categories.size());
    vector<int> trainLabels;
    vector<int> testLabels;

    float trainPercentage = 0.8;

    clock_t start, endP;

    start = clock();

    int label = 0;


    //为训练集标签
    for (const auto &category : categories)
    {
        string folderPath = basePath_Train + category;
        vector<Mat> images = readImagesFromFolder(folderPath);
        dataset_Train.push_back(images);

        // 为每个类别的图像分配标签
        for (size_t i = 0; i < images.size(); ++i)
        {
            labels.push_back(label);
        }
        cout << "Loaded " << images.size() << " images from
category " << category << " with label " << label << endl;

        // 更新标签
        label++;
    }
    //为测试集标签
    label = 0;
    for (const auto &category : categories)
    {
        string folderPath = basePath_Test + category;
        vector<Mat> images = readImagesFromFolder(folderPath);
        dataset_Test.push_back(images);
```

```
        // 为每个类别的图像分配标签
        for (size_t i = 0; i < images.size(); ++i)
        {
            labels.push_back(label);
        }
        cout << "Loaded " << images.size() << " images from
category " << category << " with label " << label << endl;

        // 更新标签
        label++;
    }

    endP = clock();
    cout << "Time: " << (double)(endP - start) / CLOCKS_PER_SEC <<
"s" << endl;

    // 将训练集导入
    label = 0;
    for (const auto &images : dataset_Train)
    {
        // 对类别内的图像进行随机排列
        vector<int> indices(images.size());
        for (auto i = 0; i < images.size(); ++i)
        {
                trainDataset[label].push_back(images[indices[i]]);
                trainLabels.push_back(label);
        }
        label++;
    }

    // 将测试集导入
    label = 0;
    for (const auto &images : dataset_Test)
    {
        // 对类别内的图像进行随机排列
        vector<int> indices(images.size());
        for (auto i = 0; i < images.size(); ++i)
        {
                testDataset[label].push_back(images[indices[i]]);
                testLabels.push_back(label);
        }
        label++;
```

```
    }

    // 计算总训练图像数量
    int numTrainImages = 0;
    for (const auto &images : trainDataset)
    {
        numTrainImages += images.size();
    }
```

4.5 初始化*HOG*特征

```
// 提取第一张图像的 HOG 特征以获取特征向量长度
    vector<float> hogFeatureExample =
computeHOGFeatures(dataset_Train[0][0]);
    int featureLength = hogFeatureExample.size();

    // 初始化 featuresTrain 矩阵以存储所有训练图像的 HOG 特征
    Mat featuresTrain(numTrainImages, featureLength, CV_32F);
```

4.6 提取训练集*HOG*特征

```
// 提取所有训练图像的 HOG 特征
    int imageIndex = 0; // 训练图像索引
    for (const auto &images : trainDataset)
    {
        for (const auto &image : images)
        {
            vector<float> hogFeatures = computeHOGFeatures(image);
            for (size_t i = 0; i < hogFeatures.size(); ++i)
            {
                featuresTrain.at<float>(imageIndex, i) =
hogFeatures[i];
            }
            imageIndex++;
        }
    }

    // 使用 featuresTrain 和 labels 训练分类器并进行预测
```

```
    // 将 labels 转换为 cv::Mat
    Mat labelsMat(numTrainImages, 1, CV_32S);
    for (size_t i = 0; i < trainLabels.size(); ++i)
    {
        labelsMat.at<int>(i, 0) = trainLabels[i];
    }
```

### 4.7 训练*SVM*

```
// 创建 SVM 分类器并设置参数
    Ptr<SVM> svm = SVM::create();
    svm->setType(SVM::C_SVC);
    svm->setC(1);
    svm->setKernel(SVM::RBF); // 使用径向基核函数
    svm->setTermCriteria(TermCriteria(TermCriteria::EPS |
TermCriteria::MAX_ITER, 1000, 1e-6));

    // 训练 SVM 分类器
    svm->train(featuresTrain, ROW_SAMPLE, labelsMat);

    // 保存训练好的 SVM 分类器
    string svmFilename = "svm_classifier.yml";
    svm->save(svmFilename);
    cout << "SVM classifier saved to " << svmFilename << endl;

     // 对训练集进行预测并计算准确率
     int correctPredictions = 0;
     for (int i = 0; i < featuresTrain.rows; ++i)
     {
         Mat currentFeature = featuresTrain.row(i);
         float prediction = svm->predict(currentFeature);
         if (prediction == trainLabels[i])
         {
             correctPredictions++;
         }
     }
     double accuracy = static_cast<double>(correctPredictions) /
featuresTrain.rows;
     cout << "Training set accuracy: " << accuracy * 100 << "%" <<
endl;
```

4.8 计算测试集*HOG*特征并计算准确率

```
// 计算测试集的 HOG 特征
    int numTestImages = testLabels.size();
    Mat featuresTest(numTestImages, featureLength, CV_32F);
    int testImageIndex = 0;
    for (const auto &images : testDataset)
    {
        for (const auto &image : images)
        {
            vector<float> hogFeatures = computeHOGFeatures(image);
            for (size_t i = 0; i < hogFeatures.size(); i++)
            {
                featuresTest.at<float>(testImageIndex, i) =
hogFeatures[i];
            }
            testImageIndex++; }
    }

    // 对测试集进行预测并计算准确率
    int correctTestPredictions = 0;
    for (int i = 0; i < featuresTest.rows; ++i)
    {
        Mat currentFeature = featuresTest.row(i);
        float prediction = svm->predict(currentFeature);
        if (prediction == testLabels[i])
        {
            correctTestPredictions++;
        }
        cout<<"Predicted Label:"<<prediction<<" Actual
        Label:"<<testLabels[i]<<endl;
    }
    double testAccuracy =
static_cast<double>(correctTestPredictions) / featuresTest.rows;
    cout << "Test set accuracy: " << testAccuracy * 100 << "%" <<
endl;
```

## 5 实验结果显示

训练结果：

```
Loaded 1329 images from category A with label 0
Loaded 572 images from category C with label 1
Loaded 654 images from category Five with label 2
Loaded 435 images from category V with label 3
```

图 5.1 训练集导入过程

```
Training set accuracy: 100%
```

图 5.2 训练集训练结果

```
Loaded 50 images from category A with label 0
Loaded 50 images from category C with label 1
Loaded 49 images from category Five with label 2
Loaded 50 images from category V with label 3
```

图 5.3 测试集导入过程

```
Test set accuracy: 100%
```

图 5.4 测试集训练结果

```
Time: 60.8472s
SVM classifier saved to svm_classifier.yml
```

图 5.5 运算时间与模型文件

预测结果：


图 5.6 预测结果图像

```
Predicted Label: A
Predicted Label: A
Predicted Label: A
Predicted Label: A
Predicted Label: A
Predicted Label: A
Predicted Label: A
Predicted Label: A
Predicted Label: A
Predicted Label: A
Predicted Label: A
Predicted Label: V
Predicted Label: V
Predicted Label: V
Predicted Label: V
Predicted Label: V
Predicted Label: V
Predicted Label: V
Predicted Label: V
Predicted Label: V
```

图 5.7 预测结果

## 6 实验总结与分析

本实验旨在探索基于计算机视觉技术的手势识别方法，使用 HOG 特征提取与 SVM 分类器相结合的方法进行手势识别任务。

HOG 特征的优势在于其对目标形状和纹理的描述能力，可以有效地捕捉手势图像的特征信息。SVM 分类器则在处理非线性分类问题上具有良好的性能，并且可以通过选择适当的核函数和正则化参数来优化模型。

在实验中，通过训练集的特征提取和模型训练，我们得到了一个手势识别模型，并在测试集上进行了预测和准确率评估。

实验结果表明，该方法在手势识别任务上取得了一定的准确率，但实际的准确率受多种因素影响，如图像质量、手势复杂性、数据量等。

进一步改进和优化该方法可以提高手势识别的准确性和鲁棒性，例如引入更多的图像增强技术、优化 HOG 参数、使用更复杂的分类器或引入深度学习方法等，这点我们将在后续课程报告中涉及。

此外，对于实际应用中的手势识别，还需考虑实时性要求、姿势变化、光照条件等因素的影响，并进行系统的优化和改进。

综上所述，通过本实验我们了解了基于计算机视觉技术的手势识别方法，探索了 HOG 特征提取与 SVM 分类器相结合的方法，并对其在手势识别任务上的性能进行了评估。实验结果表明该方法在手势识别上具有一定的潜力，但仍有改进的空间。进一步研究和改进手势识别方法可以推动其在实际应用中的发展，并提升其在不同场景下的准确性和可靠性。