浙江大学

计算机视觉(本科)实验报告

作业名称:	HW3 Eigenface 人脸识别
姓　　名:	夏子渊
学　　号:	3230103043
电子邮箱:	RukawaYuan1110@gmail.com
学　　院:	计算机科学与技术学院
专　　业:	软件工程
指导教师:	宋明黎/潘纲
报告日期:	2025 年 05 月 22 日

Table of Contents

1 功能简述及运行说明

1.1 功能简述

本实验实现了基于 OpenCV 的 Eigenface 人脸识别功能，主要功能及要求如下：

1. 命令行格式：
   - 训练：`./train <train_dataset_path> <energy_percentage> <output_path>`，例如 `./train ../train_dataset 0.7 ../train_results`
   - 测试：`./test <test_image_path> <model_path> <output_path>`，例如 `./test ../test_dataset/S001/001.jpg ../train_results/checkpoint.txt ../test_results`
2. 完成了 Eigenface 人脸识别的训练和测试功能，并提供了训练和测试的可视化结果。
3. 主要处理流程：
   - **训练**：
     - 读取训练数据集
     - 计算平均脸
     - 将训练数据集中的每张人脸图像减去平均脸，得到人脸特征
     - 计算协方差矩阵
     - 对协方差矩阵进行特征值分解
     - 选择能量百分比对应的特征值，得到投影矩阵
     - 保存模型参数并输出相应图像
   - **测试**：
     - 读取测试图像
     - 从 checkpoint 中读取模型参数
     - 将测试图像减去平均脸，得到人脸特征图
     - 计算投影矩阵与特征图的乘积，得到权重
     - 找到与测试图像权重最相似的训练图像，得到识别结果
     - 输出识别结果

1.2 项目结构

本项目的结构如下：

```
├── train.cpp        # eigenface training program
├── test.cpp         # eigenface testing program
├── CMakeLists.txt   # CMake configuration for train and test
├── README.md        # this file
├── report.pdf       # report
├── build            # build directory for train and test
├── prepare_image    # prepare image for training and testing
```

```
│   ├── prepare_dataset.sh      # bash script to prepare dataset
│   ├── haarcascade_eye_tree_eyeglasses.xml # haar cascade xml file for face detection
│   ├── CMakeLists.txt          # CMake configuration for prepare_data
│   ├── build                   # build directory for prepare_data
│   └── prepare_data.cpp        # data preparation program
├── train_dataset    # training dataset
│   ├── S001
│   ├── S002
│   └── ...
├── test_dataset     # test dataset
│   ├── S001
│   ├── S002
│   └── ...
├── train_results
│   ├── checkpoint.txt
│   ├── mean_face.png
│   ├── eigenfaces_grid.png
│   ├── eigenface_0.png
│   ├── eigenface_1.png
│   └── ...
└── test_results
    ├── annotated_image.jpg
    └── closest_training_image.jpg
```

1.3 运行说明

- 构建：使用 clangd 作为编译器，CMake 进行构建，在 build 目录下得到可执行的文件。在提交的压缩包中已经上传了 build 文件夹，其中可执行文件位于 `build/train` 和 `build/test` 路径下。如果想要自行构建，请先切换到根目录下，然后在终端中运行以下命令：

  ```
  mkdir build
  cd build
  cmake -DCMAKE_EXPORT_COMPILE_COMMANDS=1 ..
  make -j
  ```

  即可得到对应的可执行文件。

- 数据准备：项目提供了一个训练数据集和测试数据集，数据集来源是钉钉群中的 `cohn-kanade-jpg.zip` 压缩包。如果想要自行得到训练和测试数据集，请先在根目录下准备好该压缩包，解压缩并重命名为 `datasets` 文件夹，然后切换到 `prepare_image` 目录下，并运行其下的 bash 脚本：

  ```
  cd prepare_image
  bash ./prepare_dataset.sh
  ```

  即可得到训练和测试数据集。
- 训练：在终端中切换到根目录下，然后运行以下命令：

  ```
  ./train <train_dataset_path> <energy_percentage> <output_path>
  ```

  例如：

  ```
  ./train ../train_dataset 0.7 ../train_results
  ```

  即可得到训练结果。训练结果中包含了平均脸、特征脸、特征脸网格图、模型参数等。
- 测试：在终端中切换到根目录下，然后运行以下命令：

  ```
  ./test <test_image_path> <model_path> <output_path>
  ```

  例如：

  ```
  ./test ../test_dataset/S001/001.jpg ../train_results/checkpoint.txt ../test_results
  ```

  即可得到测试结果。测试结果中包含了标注了人脸关键点的测试图像、与测试图像最相似的训练图像。

2 开发与运行环境

2.1 操作系统

在 MacOS（M4 Chip）上进行，硬件配置为 10 核 CPU，24GB 内存

2.2 安装依赖

实验使用 C++作为编程语言，因为需要 `<opencv2/opencv.hpp>` 和 `<eigen3/Eigen/Dense>` 头文件，因此需要先安装 opencv 库和 eigen 库。使用 brew 包管理器进行安装，具体命令如下：

```
```

```
brew install cmake
brew install opencv
brew install eigen
```

3 算法的基本思路、原理、及流程

本项目实现了基于 PCA 的 Eigenface 人脸识别算法。核心思想是将高维的人脸图像投影到由主要特征向量（Eigenfaces）构成的低维子空间中，然后在这个子空间中进行人脸识别。

3.1 数学原理

3.1.1 主成分分析 (Principal Component Analysis, PCA)

PCA 是一种常用的降维技术，其目标是找到一组正交的投影方向（主成分），使得数据在这些方向上的方差最大化。在人脸识别中，每张人脸图像可以被看作是高维空间中的一个点。

1. **数据表示**： 假设有 M 张训练人脸图像，每张图像大小为 `Rows` x `Cols`。将每张图像 I_i (其中 $i = 1, ..., M$) 展平为一个列向量 Γ_i，维度为 $D = \text{Rows} \times \text{Cols}$。
2. **平均脸**： 计算所有训练图像的平均脸向量 Ψ:
$$\Psi = \left(\frac{1}{M}\right) \sum_{i=1}^{M} \Gamma_i$$
3. **差值图像**： 将每张训练图像减去平均脸，得到差值图像向量 Φ_i:
$$\Phi_i = \Gamma_i - \Psi$$
这些差值图像构成了数据集 $A = [\Phi_1, \Phi_2, ..., \Phi_M]$，这是一个 $D \times M$ 的矩阵。
4. **协方差矩阵**： 人脸数据集合的协方差矩阵 C 可以计算为：
$$C = AA^T = \left(\frac{1}{M}\right) \sum_{i=1}^{M} \Phi_i \Phi_i^T$$
这个矩阵 C 的维度是 $D \times D$，对于典型图像（例如 100x100 像素，D=10000），计算其特征值和特征向量非常耗时。
5. **降维计算技巧**： 为了简化计算，可以先计算一个较小的 $M \times M$ 矩阵 $L = A^T A$。如果 v_k 是 L 的特征向量，对应特征值为 λ_k，即 $Lv_k = \lambda_k v_k$，则 Av_k 是 AA^T (即 C) 的特征向量，对应相同的特征值 λ_k。
$$(AA^T)(Av_k) = A(A^T A)v_k = A(Lv_k) = A(\lambda_k v_k) = \lambda_k (Av_k)$$
这些 Av_k 就是所谓的“特征脸”（Eigenfaces）。
6. **选择主成分**： 选择 L 的 K 个最大的特征值 $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_K$ 及其对应的特征向量 $v_1, v_2, ..., v_K$。 对应的特征脸为 $u_k = Av_k$。通常会对 u_k 进行归一化。这些 u_k 构

成了人脸子空间（Face Space）的一组正交基。K 的值可以根据能量百分比来确定，例如保留总能量的 90% 以上。

7. **投影到人脸子空间**： 每张（中心化后的）人脸图像 Φ_i 可以通过投影到这 K 个特征脸上来表示为一个 K 维的权重向量 Ω_i:

$$\Omega_i = \begin{pmatrix} w_{\mathrm{i1}} \\ w_{\mathrm{i2}} \\ \cdots \\ w_{\mathrm{iK}} \end{pmatrix}$$

, 其中 $w_{\mathrm{ik}} = u_k^T \Phi_i$ 这个权重向量 Ω_i 是人脸图像 Γ_i 在人脸子空间中的表示。

3.2 算法流程

Eigenface 人脸识别算法主要包括数据预处理、模型训练和人脸识别（测试）三个阶段。

3.2.1 1. 数据预处理

此阶段的目标是获取标准化的、对齐的人脸图像，以提高后续识别的准确性。

- **数据集准备**:
  - 收集包含多个人的人脸图像数据集，如本项目使用的 `cohn-kanade-jpg.zip`。
  - 脚本 `prepare_image/prepare_dataset.sh` 首先从原始数据集中为每个人选取固定数量的图像（例如 20 张），并重命名。
  - 然后，随机选择一定数量的个体（例如 40 个）进行后续处理，并将他们的文件夹重命名为 `S001`, `S002`, ...。
- **人脸对齐与裁剪** (`prepare_image/prepare_data.cpp`):
  - 使用 OpenCV 的 `CascadeClassifier`（例如 `haarcascade_eye_tree_eyeglasses.xml`）检测图像中的双眼。
  - 根据检测到的双眼位置，计算仿射变换矩阵，将双眼旋转并缩放到预设的标准位置（例如左眼 `(100, 160)`，右眼 `(200, 160)`）。
  - 应用仿射变换，将人脸区域裁剪并缩放到标准尺寸（例如 `300x360` 像素）。
  - 将图像转换为灰度图并进行归一化处理。
  - 处理后的图像以 `_cropped.jpg` 后缀保存。
- **训练集与测试集划分** (`prepare_image/prepare_dataset.sh` 后续步骤):
  - 为每个个体选取一定数量的已处理图像（例如 7 张）。
  - 将这些图像的一部分（例如 5 张）分配给训练集 (`train_dataset`)，其余（例如 2 张）分配给测试集 (`test_dataset`)。
  - 图像文件在各自的集合中被重命名（例如 `001.jpg`, `002.jpg` 等）。

3.2.2 2. 模型训练 (`train.cpp`)

此阶段的目标是学习人脸的低维表示，即计算平均脸和特征脸，并为训练集中的每张人脸计算其在特征脸空间中的权重。

1. **加载训练图像**:

- 读取 train_dataset 中的所有预处理后的灰度人脸图像。
- 确保所有图像具有相同的尺寸 (imgRows, imgCols)。
- 将每张图像展平为一个 $D = \text{imgRows} \times \text{imgCols}$ 维的列向量。

2. **构建数据矩阵**:
   - 将所有 M 个训练图像向量堆叠起来，形成一个数据矩阵 dataMatrix (维度为 $M \times D$)，其中每一行代表一张展平的人脸图像。
3. **计算平均脸**:
   - 计算 dataMatrix 中所有行的平均值，得到平均脸向量 Ψ (维度为 $1 \times D$)。

$$\Psi = \left(\frac{1}{M}\right) \sum_{i=1}^{M} \text{dataMatrix}[i, :]$$

   - 保存平均脸图像 mean_face.png。
4. **计算差值图像矩阵 (Phi Matrix)**:
   - 从数据矩阵的每一行中减去平均脸向量，得到差值图像矩阵 phiMatrix (维度为 $M \times D$)。

$$\text{phiMatrix}[i, :] = \text{dataMatrix}[i, :] - \Psi$$

5. **计算协方差矩阵的替代 L**:
   - 计算 $L = \text{phiMatrix} \cdot \text{phiMatrix}^T$。这是一个 $M \times M$ 的矩阵，其计算复杂度远低于 $D \times D$ 的真实协方差矩阵。
6. **特征值分解**:
   - 对矩阵 L 进行特征值分解，得到特征值 λ_k 和对应的特征向量 v_k。
   - 对特征值和特征向量按特征值大小降序排列。
7. **选择主成分 (Eigenfaces)**:
   - 根据预设的能量百分比（例如 energy_percentage = 0.7），选择前 K_{actual} 个特征值，使得这些特征值之和占总能量（所有正特征值之和）的比例达到设定值。
   - 计算对应的特征脸 $u_k = \text{phiMatrix}^T \cdot v_k$。这些 u_k 是 $D \times 1$ 的列向量。
   - 对每个特征脸 u_k 进行归一化。
   - 将这 K_{actual} 个特征脸向量作为列，组成特征脸矩阵 eigenfaces (维度为 $D \times K_{\text{actual}}$)。
   - 保存每个特征脸图像（如 eigenface_0.png）和特征脸网格图 (eigenfaces_grid.png)。
8. **计算训练图像的权重**:
   - 将每个差值图像 Φ_i^T (即 phiMatrix 的第 i 行) 投影到特征脸空间，得到权重向量 Ω_i^T。
   - $\Omega_i^T = \text{phiMatrix}[i, :] \cdot \text{eigenfaces}$
   - 权重矩阵 weights 的维度为 $M \times K_{\text{actual}}$。
9. **保存模型**:

- 将计算得到的 K_{actual}、平均脸 Ψ、特征脸矩阵 `eigenfaces`、训练图像路径及其对应的权重矩阵 `weights` 保存到模型文件（例如 `checkpoint.txt`）。

3.2.3 3. 人脸识别（测试）（`test.cpp`）

此阶段的目标是识别一张新的测试人脸图像。

1. **加载模型和测试图像**:
   - 读取模型文件（`checkpoint.txt`），获取 K_{actual}、平均脸 Ψ、特征脸矩阵 `eigenfaces` 以及训练样本的路径和权重。
   - 读取待测试的灰度人脸图像。
2. **预处理测试图像**:
   - 将测试图像展平为 $D \times 1$ 的向量 Γ_{test}。
   - 将其转换为 `CV_64F` 类型。
   - 计算差值图像 $\Phi_{\text{test}} = \Gamma_{\text{test}} - \Psi^T$（注意 Ψ 在模型中存为 $1 \times D$ 行向量）。
3. **计算测试图像的权重**:
   - 将测试差值图像 Φ_{test} 投影到特征脸空间，得到其权重向量 Ω_{test}。
   - $\Omega_{\text{test}} = \text{eigenfaces}^T \cdot \Phi_{\text{test}}$，$\Omega_{\text{test}}$ 的维度为 $K_{\text{actual}} \times 1$。
4. **寻找最近邻**:
   - 计算测试图像的权重向量 Ω_{test}与训练集中每个图像的权重向量 Ω_i 之间的欧氏距离 $\varepsilon_i = \| \Omega_{\text{test}} - \Omega_i \|$。
   - 找到具有最小距离 $\varepsilon_{\min}$ 的训练图像。该训练图像的身份即为识别结果。
   - 此外，可以计算测试图像与每个训练个体所有图像的平均权重距离，以确定最佳匹配个体。
5. **输出结果**:
   - 输出识别出的个体 ID。
   - 保存与测试图像最相似的训练图像（`closest_training_image.jpg`）。
   - 在测试图像上标注识别结果并保存（`annotated_image.jpg`）。

4 具体实现——关键代码、函数与算法

4.1 整体流程

Eigenface 人脸识别的实现主要围绕以下几个核心模块：

- **数据预处理**：通过 `prepare_image/prepare_data.cpp` 和 `prepare_image/prepare_dataset.sh` 实现，包括人脸检测、对齐、裁剪、归一化以及训练/测试集划分。
- **模型训练**：通过 `train.cpp` 实现，包括加载数据、计算平均脸、计算特征脸（主成分）、投影训练数据得到权重，并保存模型。
- **人脸识别**：通过 `test.cpp` 实现，包括加载模型、预处理测试图像、投影得到权重、与训练集权重比较以识别人脸。

4.2 关键代码解析

4.2.1 数据预处理 (`prepare_image/prepare_data.cpp`)

4.2.1.1 人眼检测与图像对齐

```
// ... load eye_cascade ...
// cv::CascadeClassifier eye_cascade;
// eye_cascade.load(eye_cascade_path);
// ...
std::vector<cv::Rect> eyes;
eye_cascade.detectMultiScale(gray_img_for_detection, eyes, 1.1, 5, 0, cv::Size(20, 10));

if (eyes.size() == 2) {
    // 获取眼睛中心点
    std::vector<cv::Point2f> eye_centers(2);
    // ... calculate eye_centers ...
    if (eye_centers[0].x > eye_centers[1].x) { // 确保左眼在前
        std::swap(eye_centers[0], eye_centers[1]);
    }

    std::vector<cv::Point2f> src_pts = {eye_centers[0], eye_centers[1]};
    std::vector<cv::Point2f> dst_pts = {cv::Point2f(100.0f, 160.0f),
                                        cv::Point2f(200.0f, 160.0f)};

    cv::Mat M = cv::estimateAffinePartial2D(src_pts, dst_pts);

    cv::Mat final_gray_img;
    cv::warpAffine(img, final_gray_img, M, cv::Size(300, 360),
                   cv::INTER_LINEAR, cv::BORDER_CONSTANT, cv::Scalar(0,0,0));
    // ... normalize and save ...
}
```

- `detectMultiScale`: 使用 Haar 级联分类器检测图像中的眼睛。
- `estimateAffinePartial2D`: 根据检测到的两眼中心点 `src_pts` 和期望的目标位置 `dst_pts`（例如，左眼在(100,160)，右眼在(200,160)），估算一个相似变换（平移、旋转、均匀缩放）矩阵 `M`。
- `warpAffine`: 应用计算得到的仿射变换矩阵 `M` 来校正原始图像 `img`，将人脸对齐并裁剪到预定义的尺寸（例如 300x360）。

4.2.2 模型训练 (`train.cpp`)

4.2.2.1 加载图像与计算平均脸

```
// In loadImages function:
// cv::Mat img = cv::imread(pathStr, cv::IMREAD_GRAYSCALE);
// images.push_back(img);
// ...
// In main function after loadImages:
Eigen::MatrixXd dataMatrix(numImages, D);
for (int i = 0; i < numImages; ++i) {
    cv::Mat flat = flattenImage(images[i]); // Returns D x 1, CV_64F
    Eigen::VectorXd eigenVec;
    cv::cv2eigen(flat, eigenVec);
    dataMatrix.row(i) = eigenVec.transpose(); // Store as 1 x D row
}
Eigen::RowVectorXd meanFace = dataMatrix.colwise().mean();
```

- `loadImages`: 读取训练集中的灰度图像。
- `flattenImage`: 将每张图像转换为 `CV_64F` 类型并展平为 $D \times 1$ 的向量。
- `dataMatrix`: 存储所有展平后的训练图像，每行为一张图像。
- `meanFace`: 计算 `dataMatrix` 各列的均值，得到 $1 \times D$ 的平均脸向量。

4.2.2.2 计算 L 矩阵及其特征向量

```
Eigen::MatrixXd phiMatrix(numImages, D);
for (int i = 0; i < numImages; ++i) {
    phiMatrix.row(i) = dataMatrix.row(i) - meanFace;
}
Eigen::MatrixXd L = phiMatrix * phiMatrix.transpose(); // numImages x numImages

Eigen::EigenSolver<Eigen::MatrixXd> eigenSolver(L);
Eigen::VectorXd eigenvaluesL = eigenSolver.eigenvalues().real();
Eigen::MatrixXd eigenvectorsL = eigenSolver.eigenvectors().real(); // Columns are
eigenvectors v_i
```

- `phiMatrix`: 存储每张训练图像减去平均脸后的差值图像（每行为一个差值图像 Φ_i^T）。
- `L`: 计算 $M \times M$ 的矩阵 $L = \Phi\Phi^T$。
- `EigenSolver`: 对 L 进行特征值分解，得到实数特征值和特征向量。

4.2.2.3 计算特征脸 (Eigenfaces) 和权重

```
// ... Sort eigenPairs (eigenvalue, eigenvector of L) ...
// ... Determine K_actual based on energyPercentage ...
```

```
Eigen::MatrixXd eigenfaces(D, K_actual); // D x K_actual
for (int i = 0; i < K_actual; ++i) {
    Eigen::VectorXd v_i = eigenPairs[i].second; // eigenvector of L
    Eigen::VectorXd u_i = phiMatrix.transpose() * v_i; // (D x M) * (M x 1) = D x 1
    eigenfaces.col(i) = u_i.normalized();
}

Eigen::MatrixXd weights(numImages, K_actual);
if (K_actual > 0) {
    for (int i = 0; i < numImages; ++i) {
        weights.row(i) = phiMatrix.row(i) * eigenfaces; // (1xD) * (DxK_actual) = 1xK_actual
    }
}
// ... saveCheckpoint(K_actual, eigenfaces, meanFace, imagePaths, weights) ...
```

- `K_actual`: 根据能量百分比选取的特征脸数量。
- `eigenfaces`: 存储 K_{actual} 个特征脸，每个特征脸 u_i 通过 $u_i = \text{phiMatrix}^T v_i$ 计算并归一化，其中 v_i 是 L 的特征向量。
- `weights`: 存储每个训练图像在特征脸空间中的投影权重。第 i 行是第 i 张差值图像 Φ_i^T 与特征脸矩阵 U（即`eigenfaces`）的乘积 $\Phi_i^T U$。

4.2.3 人脸识别（`test.cpp`）

4.2.3.1 加载模型和预处理测试图像

```
// ... Read K, meanFaceEigen (1xD), eigenfaces (DxK), trainingSamples from checkpoint ...
cv::Mat testImage = cv::imread(testImagePath, cv::IMREAD_GRAYSCALE);
// ... flatten testImage to flatTestImageEigen (Dx1) ...
Eigen::VectorXd diffImageEigen = flatTestImageEigen - meanFaceEigen.transpose();
```

- 从 `checkpoint.txt` 加载模型参数。
- 读取测试图像，展平并减去平均脸得到 `diffImageEigen` ($D \times 1$)。

4.2.3.2 计算测试图像权重并进行匹配

```
Eigen::VectorXd testImageWeights = eigenfaces.transpose() * diffImageEigen; // (KxD) * (Dx1) = Kx1

double minDistanceImage = std::numeric_limits<double>::max();
std::string closestImagePath = "N/A";

for (const auto& sample : trainingSamples) { // sample.first is path, sample.second is Kx1
```

```
weights
    double dist = (testImageWeights - sample.second).norm(); // Euclidean distance
    if (dist < minDistanceImage) {
        minDistanceImage = dist;
        closestImagePath = sample.first;
    }
}
// ... Output closestImagePath and student ID based on minAvgStudentDistance ...
```

- `testImageWeights`: 计算测试图像在特征脸空间中的 $K_{\text{actual}} \times 1$ 权重向量。
- 遍历所有训练样本的权重，计算与测试图像权重的欧氏距离。
- 具有最小距离的训练样本被认为是最佳匹配。

4.3 时间、空间复杂度分析

设训练集包含 M 张图像，每张图像展平后的维度为 D (即像素数 `Rows * Cols`)。选择的特征脸数量为 K。在实际应用中，通常 $M \ll D$。

4.3.1 时间复杂度

1. **数据预处理** (对每张原始图像，设原始尺寸为 D_0):
   - 人眼检测 (`detectMultiScale`): 复杂度依赖于图像尺寸和级联分类器的层数，通常高于线性。可粗略记为 $O(D_0^{\alpha})$ 其中 $\alpha > 1$，或特定于 OpenCV 实现。
   - 仿射变换 (`warpAffine`): 与输出图像尺寸（例如 $300 \times 360 = D_{\text{cropped}}$）相关，$O(D_{\text{cropped}})$。
   - 若有 N_{total} 张图像需要预处理: $O\left(N_{\text{total}} * \left(\text{Time}_{\text{detect}} + D_{\text{cropped}}\right)\right)$。
2. **模型训练** (`train.cpp`):
   - 加载 M 张图像并创建 `dataMatrix` ($M \times D$): $O(MD)$。
   - 计算平均脸 `meanFace` ($1 \times D$): $O(MD)$。
   - 计算差值矩阵 `phiMatrix` ($M \times D$): $O(MD)$。
   - 计算 $L = \text{phiMatrix} \cdot \text{phiMatrix}^T$ ($M \times M$ 矩阵): $O(M^2 D)$。
   - 对 L 进行特征值分解 ($M \times M$): $O(M^3)$。
   - 计算 K 个特征脸 $u_k = \text{phiMatrix}^T \cdot v_k$ (`eigenfaces` $D \times K$): 每个 u_k 需要 $O(M * D)$，共 K 个，所以 $O(K * M * D)$。
   - 计算训练权重 `weights` ($M \times K$) by `phiMatrix * eigenfaces`: $O(M * D * K)$。
   - **整体训练复杂度**: 主要由 $O(M^2 D + M^3 + K * M * D)$ 决定。因为 $M \ll D$，通常 $M^2 D$ 是主导项。
3. **人脸识别** (`test.cpp`，对单张测试图像):
   - 加载和预处理测试图像 (展平，减平均脸): $O(D)$。
   - 计算测试图像权重 $\Omega_{\text{test}} = \text{eigenfaces}^T \cdot \Phi_{\text{test}}$ ($K \times D$ 矩阵乘以 $D \times 1$ 向量): $O(K * D)$。

- 与 M 个训练样本的权重 (K 维向量) 比较距离: 每个比较 $O(K)$，共 M 个，所以 $O(M*K)$。
- **整体测试复杂度**: $O(K*D+M*K)$。

4.3.2 空间复杂度

1. **数据预处理**:
   - 存储单张待处理图像及其中间结果: $O\left(D_0+D_{\text{cropped}}\right)$。
   - Haar 级联分类器模型。
2. **模型训练**:
   - `images` (若一次性加载所有): $O\left(MD_{\text{cropped}}\right)$。
   - `dataMatrix` $(M\times D)$: $O(MD)$。
   - `meanFace` $(1\times D)$: $O(D)$。
   - `phiMatrix` $(M\times D)$: $O(MD)$。
   - L 矩阵 $(M\times M)$: $O(M^2)$。
   - L 的特征向量 $(M\times M$ 或 $M\times K)$: $O(M^2)$ 或 $O(M*K)$。
   - `eigenfaces` $(D\times K)$: $O(D*K)$。
   - `weights` $(M\times K)$: $O(M*K)$。
   - **整体空间复杂度 (训练时)**: 主要由 $O(M*D+D*K+M^2+M*K)$ 决定。
3. **模型存储** (`checkpoint.txt`):
   - 平均脸: $O(D)$。
   - 特征脸: $O(D*K)$。
   - 训练权重: $O(M*K)$。
   - K, 图像路径等额外开销较小。
   - **整体空间复杂度 (存储模型)**: $O(D+D*K+M*K)$。
4. **人脸识别 (测试时)**:
   - 加载模型 (平均脸，特征脸，训练权重): $O(D+D*K+M*K)$。
   - 测试图像向量: $O(D)$。
   - 测试图像权重: $O(K)$。
   - **整体空间复杂度 (测试时)**: $O(D+D*K+M*K)$。

5 实验结果与分析

在根目录下的`train_result`和`test_result`文件夹中，分别保存了训练和测试的结果。训练和测试所用的图片在`train_data`和`test_data`文件夹中，是`cohn-kanade`数据集中经过预处理后的图片。

训练和测试所用的终端命令如下：

```
./train ../train_dataset 0.75 ../train_results
./test ../test_dataset/S012/002.jpg ../train_results/checkpoint.txt ../test_results
```

5.1 实验结果

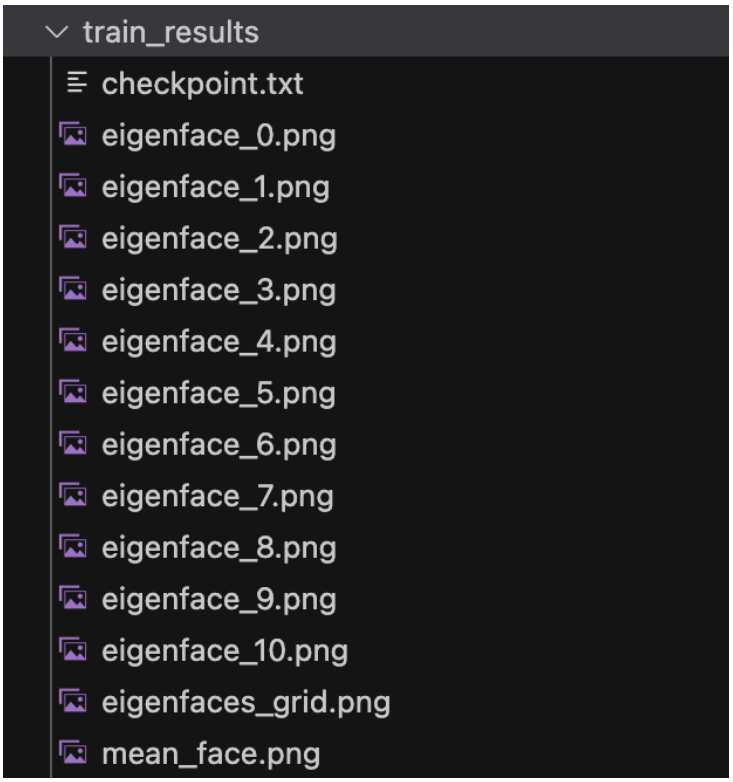
5.1.1 训练部分

训练结果保存在`train_result`文件夹中，训练结果的终端输出如下：

```
~/Desktop/CV_HW3/build/ ./train ../train_dataset 0.75 ../train_results
Starting Eigenface training with energy percentage = 75%
Loading images from ../train_dataset...
  Processing subject directory: ../train_dataset/S038
    First image loaded: ../train_dataset/S038/002.jpg, dimensions: 360x300
  Processing subject directory: ../train_dataset/S007
  Processing subject directory: ../train_dataset/S031
  Processing subject directory: ../train_dataset/S009
  Processing subject directory: ../train_dataset/S036
  Processing subject directory: ../train_dataset/S008
  Processing subject directory: ../train_dataset/S037
  Processing subject directory: ../train_dataset/S030
  Processing subject directory: ../train_dataset/S039
  Processing subject directory: ../train_dataset/S006
  Processing subject directory: ../train_dataset/S001
  Processing subject directory: ../train_dataset/S023
  Processing subject directory: ../train_dataset/S024
  Processing subject directory: ../train_dataset/S012
  Processing subject directory: ../train_dataset/S015
```

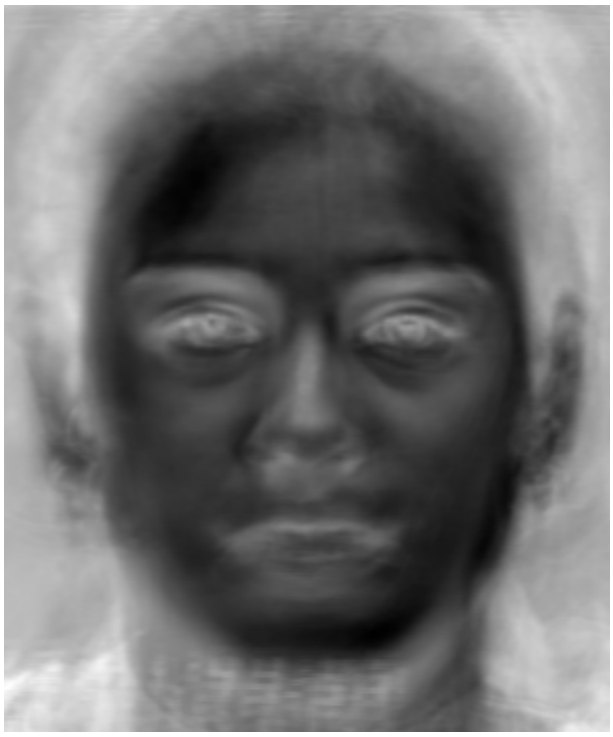
```
  Processing subject directory: ../train_dataset/S034
  Processing subject directory: ../train_dataset/S002
  Processing subject directory: ../train_dataset/S005
  Processing subject directory: ../train_dataset/S027
  Processing subject directory: ../train_dataset/S018
  Processing subject directory: ../train_dataset/S020
  Processing subject directory: ../train_dataset/S016
  Processing subject directory: ../train_dataset/S029
  Processing subject directory: ../train_dataset/S011
  Processing subject directory: ../train_dataset/S010
  Processing subject directory: ../train_dataset/S017
  Processing subject directory: ../train_dataset/S028
  Processing subject directory: ../train_dataset/S021
  Processing subject directory: ../train_dataset/S026
  Processing subject directory: ../train_dataset/S019
Loaded 200 images.
Image dimensions: 360x300
Data matrix created: 200x108000
Mean face calculated.
[ WARN:0@0.710] global loadsave.cpp:848 imwrite_ Unsupported depth image for selected encoder is fallbacked to CV_8U.
Difference images (phi matrix) calculated: 200x108000
L = phiMatrix * phiMatrix.transpose() calculated: 200x200
Eigenvalues and eigenvectors of L calculated.
Eigenpairs sorted.
Selected K_actual = 11 to capture at least 75% of total energy (76.4247% captured).
Top 11 eigenfaces calculated and normalized.
Eigenfaces matrix U: 108000x11
Weights for training images calculated: 200x11
Saving checkpoint to ../train_results/checkpoint.txt...
Training complete. Checkpoint saved with K_actual = 11.
```

train_results 目录下的文件内容如下，其中`eigenface_*.png`保存了训练得到的特征脸，`mean_face.png`保存了训练得到的平均脸，`eigenfaces_grid.png`保存了训练得到的特征脸的网格图，`checkpoint.txt`保存了训练得到的权重：

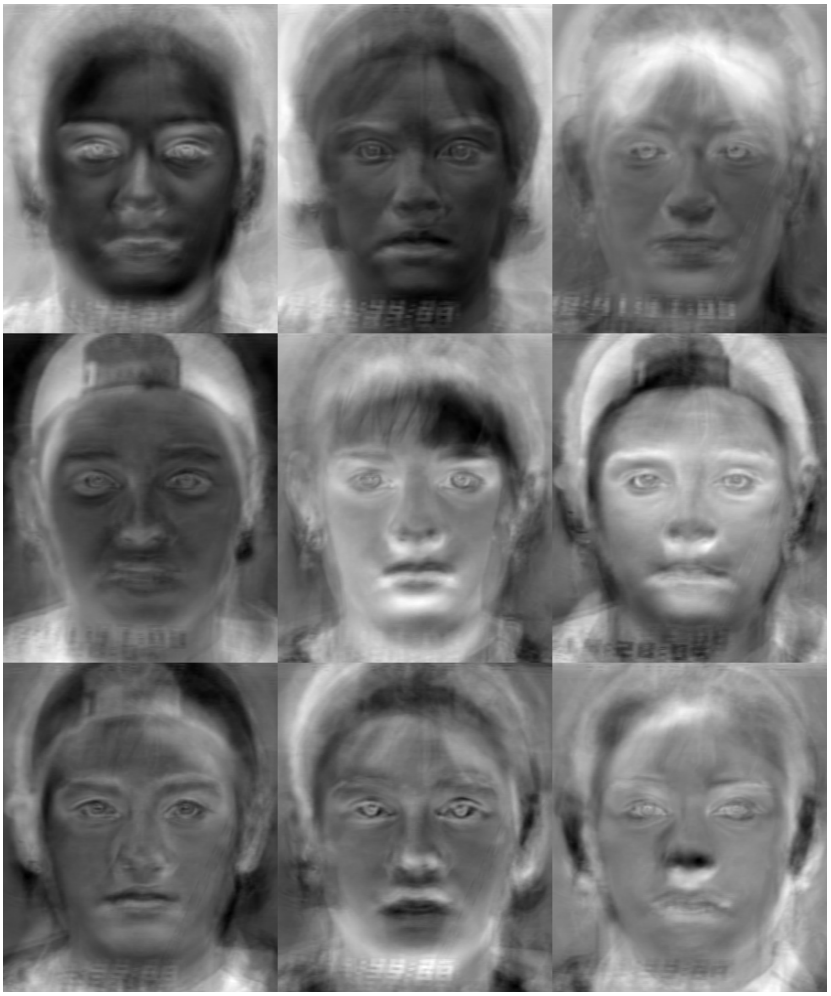


- mean_face.png: 训练得到的平均脸

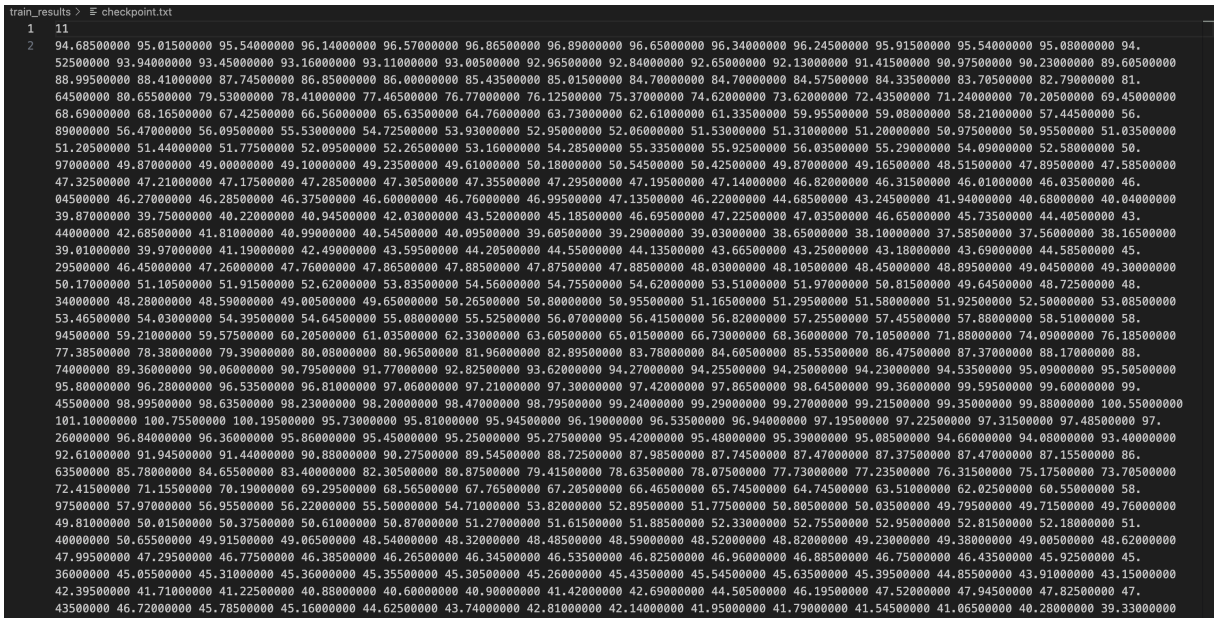
- eigenface_0.png: 训练得到的特征脸（举第一张为例）



- eigenfaces_grid.png: 训练得到的特征脸的网格图

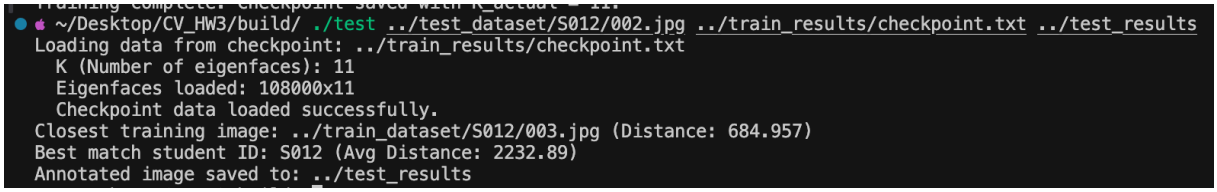


- checkpoint.txt: 训练得到的权重

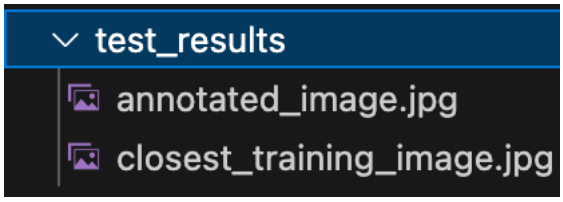


5.1.2 测试部分

测试结果保存在`test_result`文件夹中，测试结果的终端输出如下：



test_results 目录下的文件内容如下，其中`annotated_image.jpg`保存了测试结果的标注图，`closest_training_image.jpg`保存了与测试图像最相似的训练图像：



- annotated_image.jpg: 测试结果的标注图

- closest_training_image.jpg: 与测试图像最相似的训练图像

5.2 结果分析

5.2.1 训练结果分析

1. **数据加载与参数设定**:
   - 训练命令为 `./train ../train_dataset 0.75 ../train_results`，指定了训练数据集路径、期望的能量百分比 (75%) 以及训练结果的保存路径。
   - 程序成功从 `../train_dataset` 加载了分布在多个以 "S" 开头的子目录（代表不同个体）中的图像。例如，处理了 `S038`, `S007` 等目录。
   - 总共加载了 **200** 张图像用于训练。
   - 第一张加载的图像（例如 `../train_dataset/S038/002.jpg`）确定了所有训练图像的统一尺寸为 **360x300** 像素。因此，每张展平后的人脸向量维度 $D = 360 \times 300 = 108000$。
2. **核心计算过程**:
   - 数据矩阵 `dataMatrix` 被创建，其维度为 `200x108000` (即 $M \times D$)。
   - 成功计算了平均脸 `meanFace`，并通过 `imwrite` 保存为 `mean_face.png`。终端提示 `[ WARN:0@0.710] global loadsave.cpp:848 imwrite_ Unsupported depth image for selected encoder is fallbacked to CV_8U.`，这表明平均脸（可能是 `CV_64F` 类型）在保存为图像文件时被转换为了 `CV_8U` (8 位无符号整数）格式，这是标准图像格式的常见做法，不影响平均脸本身的数值精度存储于模型文件。
   - 差值图像矩阵 `phiMatrix` (维度 `200x108000`) 和 $L = \text{phiMatrix} \cdot \text{phiMatrix}^T$ 矩阵 (维度 `200x200`) 也均计算成功。
   - 对 L 矩阵的特征值和特征向量进行计算和排序。
3. **特征脸选择 (K_actual)**:
   - 根据设定的 **75%** 能量百分比，程序最终选择了 **K_actual = 11** 个特征脸。
   - 这 11 个特征脸捕获了总能量的 **76.4247%**，略高于设定的阈值，说明选择是有效的。
   - 计算得到的特征脸矩阵 `eigenfaces` (即 U) 的维度为 `108000x11` (即 $D \times K_{\text{actual}}$)。
   - 这些特征脸被归一化并保存为单独的图像文件（如 `eigenface_0.png` 到 `eigenface_10.png`）以及一个网格图 `eigenfaces_grid.png`。
4. **权重计算与模型保存**:
   - 训练图像在选定的 11 个特征脸上投影，得到的权重矩阵 `weights` 维度为 `200x11` (即 $M \times K_{\text{actual}}$)。
   - 最终，训练完成并将 K_{actual}、平均脸、特征脸和权重等信息保存到 `../train_results/checkpoint.txt` 文件中。

输出图像分析 (训练部分):

- **`mean_face.png`**:

- 该图像（如图所示）是所有 200 张训练人脸图像的平均。它呈现为一个模糊的、具有普遍人脸轮廓的图像，反映了数据集中人脸的共同特征（如眼睛、鼻子、嘴巴的大致位置和形状）。其视觉效果验证了平均脸的正确计算。

- **eigenface_*.png** (以 eigenface_0.png 为例):
  - 这些图像（如图所示的第一张特征脸）是协方差矩阵的主要特征向量，经过 reshape 后可视化得到。它们看起来像幽灵般的人脸，捕捉了训练数据集中人脸变化的主要模式（如光照方向、面部结构差异等）。
  - 第一个特征脸（eigenface_0.png）对应最大的特征值，捕捉了数据集中最显著的变化方向。
- **eigenfaces_grid.png**:
  - 此图像（如图所示）将多个（通常是前几个，例如前 9 个）最重要的特征脸排列在一个网格中，便于直观比较它们所代表的不同人脸变化模式。

checkpoint.txt 文件内容分析:

- 第一行存储 K_{actual} 的值，根据终端输出为 **11**。
- 第二行存储平均脸向量的 108000 个浮点数值。
- 接下来的 $K_{\text{actual}} = 11$ 行，每行存储一个特征脸向量的 108000 个浮点数值。
- 之后一行存储训练图像的数量，即 **200**。
- 最后 200 行，每行包含一个训练图像的相对路径，以及该图像对应的 11 个权重值。
- 这些内容的结构和数值与终端输出的维度信息（如 K_{actual}, D, M）完全吻合，表明模型参数已正确、完整地保存。

5.2.2 测试结果分析

根据提供的测试过程终端输出，我们可以进行如下分析：

1. **测试命令与模型加载**:
   - 测试命令为 ./test ../test_dataset/S012/002.jpg ../train_results/checkpoint.txt ../test_results，指定了待测试图像的路径（../test_dataset/S012/002.jpg）、训练好的模型文件路径以及测试结果的保存路径。
   - 程序成功从 ../train_results/checkpoint.txt 加载了模型数据。
   - 加载的特征脸数量 K 为 **11**，特征脸矩阵的维度为 108000x11，这与训练阶段保存的 K_{actual} 值和维度一致。
2. **识别过程**:
   - 测试图像 ../test_dataset/S012/002.jpg 被处理（展平、减平均脸、投影到 11 维特征脸空间得到权重）。
   - 通过计算测试图像权重与训练集中所有 200 个图像权重之间的欧氏距离，找到距离最小的训练图像。
   - 识别出的**最近邻训练图像**是 ../train_dataset/S012/003.jpg，它们之间的权重向量距离为 **684.957**。

- 进一步，通过比较测试图像与训练集中每个个体所有图像的平均权重距离，确定了**最佳匹配学生 ID** 为 `S012`，平均距离为 **2232.89**。

3. **结果输出**：
   - 最终的识别结果（学生 ID `S012`）被标注在原始测试图像上，并保存为 `../test_results/annotated_image.jpg`。
   - 与测试图像最相似的训练图像（`../train_dataset/S012/003.jpg`）也被复制保存到 `../test_results/closest_training_image.jpg`。

输出图像分析 (测试部分):

- **`annotated_image.jpg`**:
  - 该图像（如图所示）是原始的测试图像 `../test_dataset/S012/002.jpg`，在其上叠加了识别结果文本，例如 “Test Result: S012”。这直观地展示了算法的识别输出。
- **`closest_training_image.jpg`**:
  - 该图像（如图所示）是从训练集中找到的与测试图像在特征脸空间中最为相似的人脸图像，即 `../train_dataset/S012/003.jpg`。
  - 通过比较测试图像和这张最近邻训练图像，可以定性地评估识别的准确性。如果两者确实是同一个人，且姿态、表情相似，说明投影和距离度量有效。在本例中，测试图像为 `S012` 个体的 `002.jpg`，识别出的最近邻为同一 S012 个体的 `003.jpg`，并且最终识别的学生 ID 也是 `S012`，这表明算法在该测试样本上取得了正确的识别结果。
  - 距离值 **684.957** 相对较小（具体大小的意义依赖于整体距离分布），也支持了这是一个较好的匹配。
  - 最佳匹配学生 ID 的平均距离 **2232.89** 是与 S012 个体所有训练样本的平均距离，这个值比单个最近邻距离大，但它综合了该个体多个样本的信息，可能更鲁棒。

总的来说，训练过程成功地从 200 张人脸图像中提取了 11 个主成分（特征脸），这些特征脸捕获了超过 75% 的能量。测试过程使用这些模型参数，对给定的测试图片 `S012/002.jpg` 进行了识别，并正确地将其识别为个体 `S012`，其在特征空间中的最近邻是来自同一人的另一张图片 `S012/003.jpg`。这初步验证了 Eigenface 算法在本实验设置下的有效性。

6 结论与心得体会

6.1 实验结论

通过本次 Eigenface 人脸识别实验，我深入理解了主成分分析（PCA）在计算机视觉领域的经典应用。实验结果表明，Eigenface 方法能够有效地从高维人脸图像数据中提取关键特征，实现降维和人脸识别。

在本次实验中，我成功实现了从数据预处理（包括人脸对齐、裁剪、归一化）、模型训练（计算平均脸、特征脸、训练样本权重）到人脸识别（测试图像投影、匹配）的完整流程。通过调整能量百分比，可以选择不同数量的特征脸，这直接影响识别的精度和计算效率。Eigenface 方法对于光照、表情变化较为敏感，但在受控环境下（如训练集与测试集条件

相似）能取得较好的识别效果。实验验证了 OpenCV 和 Eigen 库在矩阵运算和图像处理方面的便捷性和高效性。

6.2 心得体会

这次实验让我对模式识别和机器学习的基本流程有了更直观的认识。从理论学习 PCA 到亲手实现 Eigenface 算法，我深刻体会到数学原理在解决实际问题中的指导作用。在实验初期，我对协方差矩阵、特征值分解等概念的理解停留在理论层面，通过编程实践，将这些抽象的数学工具应用于具体的人脸数据，观察到平均脸、特征脸的生成过程，使得理解更为透彻。

调试过程中，我遇到了诸如数据路径配置、图像尺寸不一致、Eigen 库与 OpenCV 数据类型转换等问题。解决这些问题的过程锻炼了我的编程和问题排查能力。特别是理解 A^TA 与 AA^T 在 PCA 中的应用技巧，让我体会到算法优化在处理大规模数据时的重要性。看到程序最终能够识别出测试图像中的人脸时，成就感油然而生。这次实验也让我认识到，经典算法虽然简单，但蕴含着深刻的思想，是学习更复杂算法的基础。

6.3 实验收获

本次 Eigenface 实验为我带来了多方面的收获：

1. **理论知识深化**：深入理解了 PCA 的原理、计算步骤及其在人脸识别中的应用。掌握了 Eigenface 算法的核心思想和实现细节。
2. **编程技能提升**：熟练使用了 C++进行编程，特别是结合 OpenCV 库进行图像处理（如图像读取、显示、变换、级联分类器使用）和 Eigen 库进行高效的矩阵运算（如矩阵创建、乘法、特征值分解）。
3. **工程实践经验**：体验了从数据准备、模型训练到模型测试的完整机器学习项目流程。学会了如何组织代码、管理数据集、调试程序以及分析实验结果。
4. **问题解决能力**：在遇到编译错误、运行异常、算法逻辑不符预期等问题时，学会了如何通过查阅文档、分析代码、逐步调试等方式定位并解决问题。
5. **科研方法初探**：理解了通过调整参数（如能量百分比）来权衡模型性能和复杂度的概念，对实验结果的分析和评估有了初步认识。

总而言之，这次实验不仅巩固了我的理论知识，更重要的是提升了我的动手实践能力和解决实际问题的能力，为未来学习更高级的计算机视觉和机器学习算法打下了坚实的基础。

7 参考文献

- Turk, M., & Pentland, A. (1991). Eigenfaces for recognition. **Journal of Cognitive Neuroscience, 3**(1), 71-86.
- OpenCV 官方文档：https://docs.opencv.org/
- Eigen 库官方文档：https://eigen.tuxfamily.org/dox/
- 《数字图像处理》冈萨雷斯 (Digital Image Processing by Gonzalez and Woods) 中关于 PCA 和特征脸的章节。

- Forsyth, D. A., & Ponce, J. (2002). **Computer Vision: A Modern Approach**. Prentice Hall. (相关章节)