武漢理工大學

数学建模暑期培训论文

第 2 题

基于卷积自编码与 k-medoids 模型的指纹编码与分类

第 10 组

姓名	方向
刘子川	编程
程宇	建模
祁成	写作

2025 年 5 月 30 日

摘要

指纹识别系统的存储量巨大，提高指纹编码率具有重要意义。本文提出利用深度学习方法建立卷积自编码器模型，在非线性学习空间中有损压缩编码，并根据编码距离对指纹进行聚类。

为消除指纹图像噪声和伪特征点影响，设计了预处理策略体系以统一图像尺度、细化纹线骨架。首先对指纹图像进行灰度值线性归一化处理，尺寸放缩后利用**分数阶微分算子**构造掩膜进行图像增强，最后通过方向场确定脊线实现像素二值化。预处理保留了指纹的连接状态、拓扑结构和纹理信息，剔除了次要纹线宽度信息，有利于后续编码。

针对问题一，鉴于小样本条件下难以有效提取指纹特征的问题，设计了一种**卷积自编码网络**的特征提取算法。将预处理后的图像信号用**卷积核**转化到高维特征空间，利用大量无标签指纹样本训练卷积自编码器网络。通过损失函数 **KL 散度**进行**无监督学习**，编码出最佳线性空间特征。训练提取的指纹特征损失率仅为 **0.04%**，具有高还原度。

针对问题二，对卷积自编码器生成的编码向量进行归一化处理，设计加权相似度量化指纹间相似性。计算归一化编码向量的**谷本系数**、**皮尔逊相关系数**和**马氏距离**，同向化处理后加权求和，得到量化相似度。相似度热图可视化显示指纹间的异同。

针对问题三，建立中心聚类模型，以问题二的量化相似度为代价函数，使用 **k-medoids** 算法选取聚类中心完成分类。对于附件中的 16 个指纹样本，基于加权相似度计算每对样本的相似度，选取 k 个样本点作为聚类中心，以样本到中心的平均相似度为代价函数。遍历聚类中心组合，当 $k=4$ 时代价函数最优，分类结果为样本 **10** 独立为一组，其余三组为 $\{2,3,5\}$、$\{1,4,7,11,16\}$ 和 $\{6,8,9,12,13,14,15\}$。

本文的优点为：1. 采用卷积核提取指纹细节，大幅降低数据维度的同时挖掘图像特征。2. 相较传统编码，模型具有较强的个体表征能力，可通过解码器高精度还原图像。

关键词： 卷积自编码器　分数阶微分算子　**k-medoids**　无监督学习

目录

1 问题重述

1.1 问题背景

自动指纹识别系统（Automated Fingerprint Identification System, AFIS）在多个领域有广泛应用。研究主要集中在图像增强、指纹分类和细节匹配三个方面，分为“离线部分”和“在线部分”。如图 1所示，离线部分包括采集指纹、提取细节点、存储到模板库；在线部分包括采集指纹、提取细节点并与模板库匹配，判断是否来自同一手指[1, 3]。指纹分类用于大规模指纹库中减少搜索范围。指纹图像占用空间大，像素信息不适合计算机分析，需用最少字节描述指纹内在结构、形态和特征。AFIS 可扫描犯罪现场指纹，与执法机关的指纹档案比对，列出匹配百分比供专家验证[2][1]。

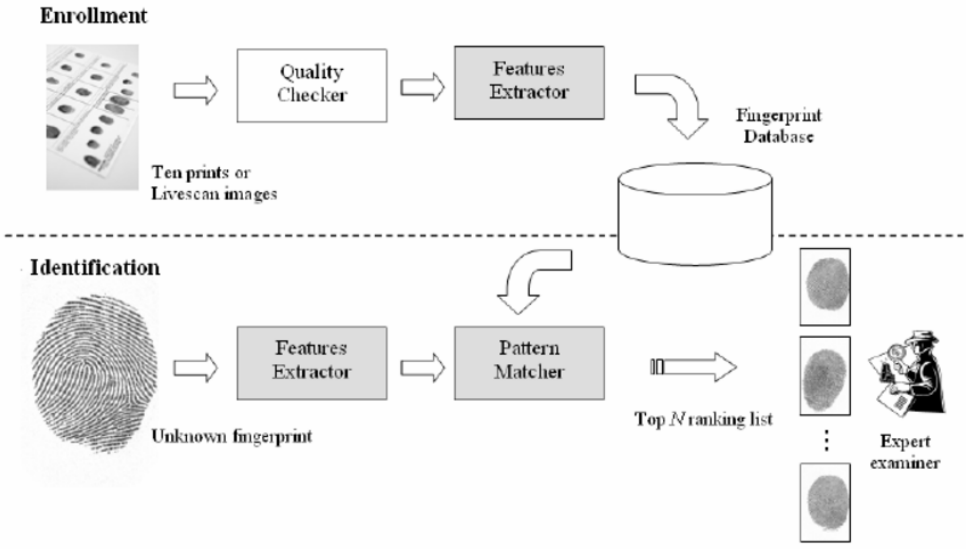


图 **1**　自动指纹识别系统框图

1.2 问题概述

围绕附件中的 16 幅指纹图像，不借助现有指纹软件，提出以下问题：

编码：给出一种用不超过 200 字节（称为“指纹密码”）描述指纹基本特征的表示方法，介绍其数学原理。

匹配：实现编码方法，为每幅指纹生成“指纹密码”。基于这些表示，比较指纹间的异同及相似程度。

应用：对比并归类 16 个指纹，给出依据和结果。

2 模型假设

(1) 附件中所有指纹图像为清晰完整图像，无压缩信息损失。

[1]https://baike.baidu.com/item/AFIS/2851410?fr=aladdin

(2) 指纹图像特征具有代表性，包含正常人指纹的所有特征。

3 符号说明

符号	说明
P_i	第 i 张图像，$i \in \{1, 2, \ldots, 16\}$
$u_{P(x,y)}$	像素矩阵
(x, y)	像素点位置值
$fringe_{x1}, fringe_{x2}$	图片的上下边界
$fringe_{y1}, fringe_{y2}$	图片的左右边界
E	信息熵
$iDir$	脊线方向
γ	编码器的图像编码
b	编码大小
Z	解码器还原的图像矩阵
$epoch$	迭代次数
in/out	出/入通道数
lr	学习率
k_i	卷积核 i
T	谷本系数
P	皮尔逊相关系数
M	马氏距离
$\{c_i\}$	聚类中心
$\max_{1 \leqslant j \leqslant k} S(\gamma_i, c_j)$	样本点 γ_i 与 $\{c_i\}$ 间的最高相似度值

注：表中未说明的符号以首次出现处为准

4 数据预处理

在搭建模型前，对指纹图像进行预处理以统一规格、去除噪声，流程如图 2所示。

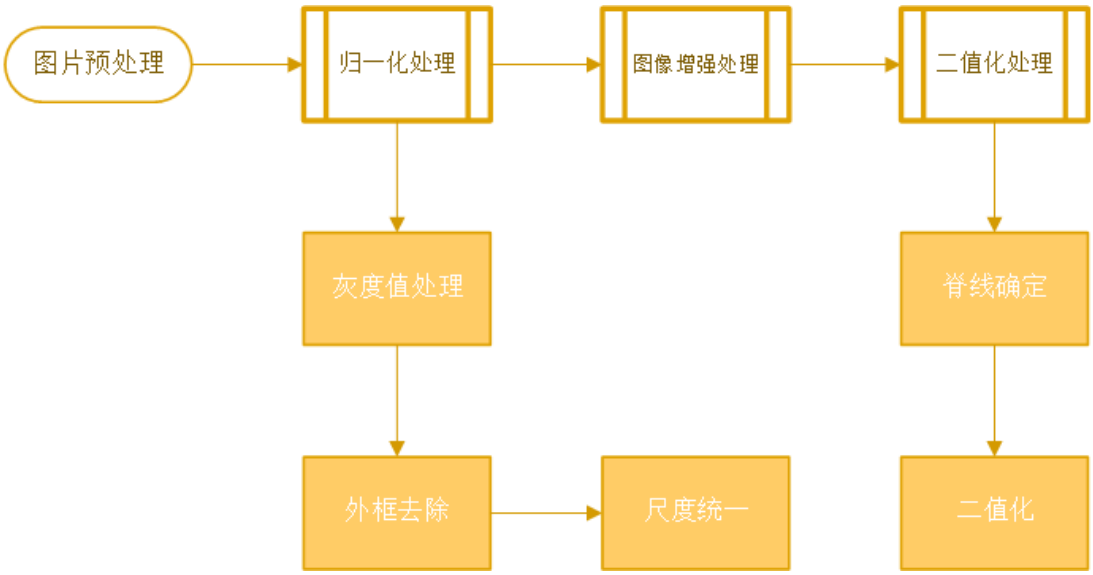


图 **2**　图片数据预处理流程

4.1 归一化处理

4.1.1 灰度值处理

由于指纹图像纹理深浅不一，先进行灰度值归一化。图像 P_i 的像素灰度值 $P_i(x,y) \in [0,255]$，255 为纯白，0 为纯黑。计算最大和最小灰度值：

$$P_{i,\max} = \max_{1\leqslant x\leqslant n, 1\leqslant y\leqslant m} P_i(x,y),$$
$$P_{i,\min} = \min_{1\leqslant x\leqslant n, 1\leqslant y\leqslant m} P_i(x,y),$$

其中 n、m 为图像行数和列数。线性归一化：

$$P_i'(x,y) = \frac{255}{P_{i,\max} - P_{i,\min}}(P_i(x,y) - P_{i,\min}),$$

得到灰度值归一化图像 P_i'，范围为 $[0,255]$。

4.1.2 外框去除与尺度统一

为去除空白外框，设定边界阈值 $threshold_N$。当某行非空白像素数大于 $threshold_N$，定义上下边界：

$$fringe_{x1} = \min\{x \mid \sum_{i=1}^{m}(P_i'(x,i) < 255) > threshold_N\},$$
$$fringe_{x2} = \max\{x \mid \sum_{i=1}^{m}(P_i'(x,i) < 255) > threshold_N\},$$

左右边界：

$$fringe_{y1} = \min\{y \mid \sum_{i=1}^{n}(P_i'(i,y) < 255) > threshold_N\},$$
$$fringe_{y2} = \max\{y \mid \sum_{i=1}^{n}(P_i'(i,y) < 255) > threshold_N\}.$$

去除外框后图像为 $P_i'' = P_i'(fringe_{x1} : fringe_{x2}, fringe_{y1} : fringe_{y2})$。定义标准行数 N 和列数 M，放缩图像：

$$P_i^s(x,y) = P_i''\left(\left\lfloor\frac{(fringe_{x2} - fringe_{x1})(x-1)}{N-1}\right\rfloor + fringe_{x1},\right.$$
$$\left.\left\lfloor\frac{(fringe_{y2} - fringe_{y1})(y-1)}{M-1}\right\rfloor + fringe_{y1}\right),$$
$$x \in [1,N], \quad y \in [1,M].$$

得到归一化图像 $P_i^s(x,y)$。

4.2 图像增强处理

利用 Grünwald-Letnikov 分数阶微分算子[5] 构造自适应函数，采集梯度信息和信息熵确定微分阶数 v。梯度 G 反映像素特征突变：

$$G[P(x,y)] = \max\left\{\left|\frac{\partial P}{\partial x}\right|, \left|\frac{\partial P}{\partial y}\right|\right\}. \tag{1}$$

信息熵 E 描述边缘纹理变化：

$$E = -\sum_{i=1}^{n} p(x_i) \cdot \log_2(p(x_i)), \tag{2}$$

归一化：

$$E' = \frac{E - E_{\min}}{E_{\max} - E_{\min}}. \tag{3}$$

构造关系函数 $v = w_1 \cdot G + w_2 \cdot E' + w_3$，梯度和信息熵越大，$v$ 越大。实验中 $v = 3$。Grünwald-Letnikov 微分近似：

$$\frac{\partial^v P(x,y)}{\partial x_i^v} \doteq \sum_{k=0}^{t-\alpha}(-1)^k\binom{v}{k}P(x_i - k), \tag{4}$$

取 8 个方向，构造 5×5 掩膜，权重为前三项系数，中心点、第一层、第二层权重：

$$w_0 = \frac{8}{4v^2 - 12v + 8}, \quad w_1 = \frac{-v}{4v^2 - 12v + 8}, \quad w_2 = \frac{v}{16(v-2)}.$$

对图像进行掩膜操作：

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} P(x+s, y+t) \cdot w(x,y), \tag{5}$$

其中 $a = b = \frac{n-1}{2}$，$n = 5$。

4.3 二值化处理

4.3.1 场方向估计

指纹图像具有清晰方向场。对像素 $P(x,y)$，取 9×9 像素矩阵 $u_{P(x,y)}$：

$$u_{P(x,y)} = \begin{bmatrix} P(x-4, y-4) & \cdots & P(x+4, y-4) \\ \vdots & \ddots & \vdots \\ P(x+4, y+4) & \cdots & P(x+4, y+4) \end{bmatrix}.$$

超出边界的像素设为 255。按图 3的 8 个方向计算灰度平均值 $Gmean[i]$，分组比较垂直方向差值：

$$Gdiff[j] = |Gmean[j] - Gmean[j+4]|, \quad j = 1, 2, 3, 4.$$

最大差值方向为脊线方向：

$$iDir = \begin{cases} \arg\max_j(Gdiff[j]), & |P(x,y) - \max_j(Gdiff[j])| < |P(x,y) - \max_j(Gdiff[j+4])|, \\ \arg\max_j(Gdiff[j]) + 4, & \text{otherwise}. \end{cases}$$

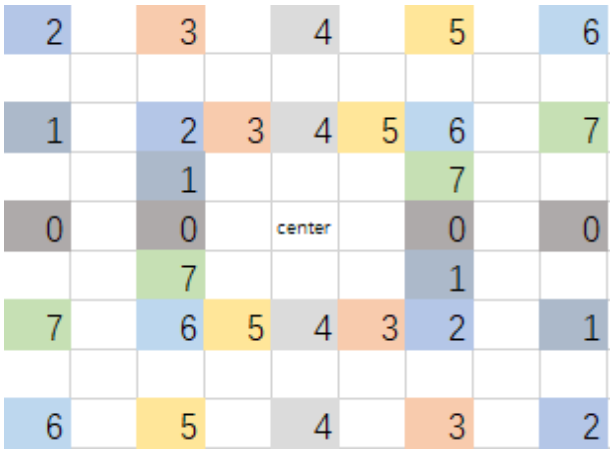


图 **3**　脊线估计方向

4.3.2 二值化

根据方向场，计算脊线方向 $iDir$ 和法线 $iVar = iDir+4$ 的灰度平均值 $Gmean[iDir]$ 和 $Gmean[iVar]$，二值化：

$$P(x,y) = \begin{cases} 255, & Gmean[iDir] \geqslant Gmean[iVar], \\ 0, & \text{otherwise}. \end{cases}$$

255 表示背景和谷线，0 表示脊线。预处理效果如图 4所示：

(a) 指纹原始图像　　(b) 指纹预处理后

图 **4**　图像预处理效果图

5 问题一模型的建立与求解

5.1 问题描述与分析

问题一要求用不超过 200 字节的“指纹密码”描述指纹特征。传统压缩方法（如霍夫曼编码、Golomb 编码、LZW 编码、小波编码[7, 8, 9]）在 200 字节限制下易丢失细节。

卷积自编码器通过无监督学习提取关键信息，降低维度，适合小样本编码。

5.2 搭建卷积自编码器模型

自编码器（AutoEncoder）通过无监督学习生成低维编码，适用于降维和特征检测[10]。本文基于 VGG 模型[4]，构造卷积自编码器（CAE），通过交叉验证优化小样本数据参数。

5.2.1 卷积神经网络

卷积核 矩阵卷积提取图像特征，全卷积定义为：

$$z(u,v)=\sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty}x_{i,j}\cdot k_{u-i,v-j}, \tag{6}$$

有效值卷积：

$$z(u,v)=\sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty}x_{i+u,j+v}\cdot k_{rot,i,j}\cdot\chi(i,j),\quad \chi(i,j)=\begin{cases}1, & 0\leqslant i,j\leqslant n,\\ 0, & \text{otherwise.}\end{cases} \tag{7}$$

其中 X 为 $m\times m$ 图像矩阵，K 为 $n\times n$ 卷积核（通常 3×3），K_{rot} 为 K 转置。卷积示意图如图 5所示：

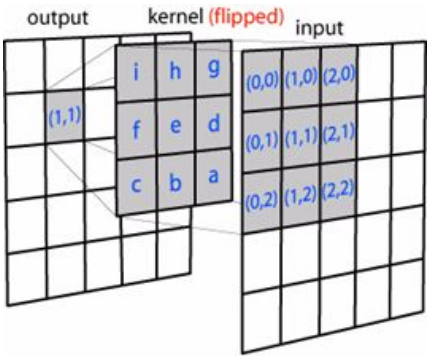


图 **5** 卷积核函数示意图

池化 池化压缩 Feature Map，减少参数，增强鲁棒性。Max-Pooling 取 2×2 邻域最大值，如图 6所示：

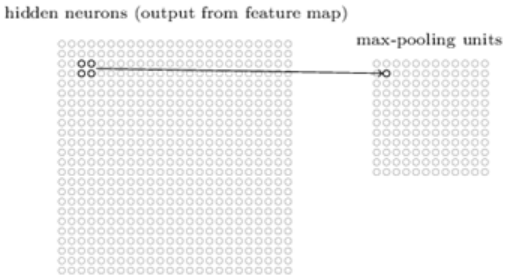


图 **6**　池化层 **Max-Pooling** 示意图

激活　激活层计算：

$$u_k = \sum_{i=1} w_{ki} x_i, \quad y_k = f(u_k - b_k),$$

使用 ReLU 激活函数：

$$f_{\text{ReLU}}(z) = \max(0, z), \quad \frac{d}{dz} f_{\text{ReLU}} = \begin{cases} 1, & z > 0, \\ 0, & z \leqslant 0. \end{cases} \tag{8}$$

输出层使用 softmax：

$$f_{\text{softmax}} = \frac{e^i}{\sum_j e^j}. \tag{9}$$

5.2.2 卷积自编码器

给定样本 $X = \{x_1, x_2, \ldots, x_n\}, X \in \mathbb{R}^{n \times c \times w \times h}$，自编码器编码和解码：

$$\gamma(x) = \text{Encoder}(W_1 x + b), \quad z(x) = \text{Decoder}(W_2 \gamma(x) + c). \tag{10}$$

最小化重构误差：

$$\theta = \arg\min_{\theta} L(X, Z) = \arg\min_{\theta} \frac{1}{2} \sum_{i=1}^{N} \|x^{(i)} - z(x^{(i)})\|^2. \tag{11}$$

CAE 编码器结构与 CNN 卷积池化部分相同，解码器对称，结构如图 7所示：

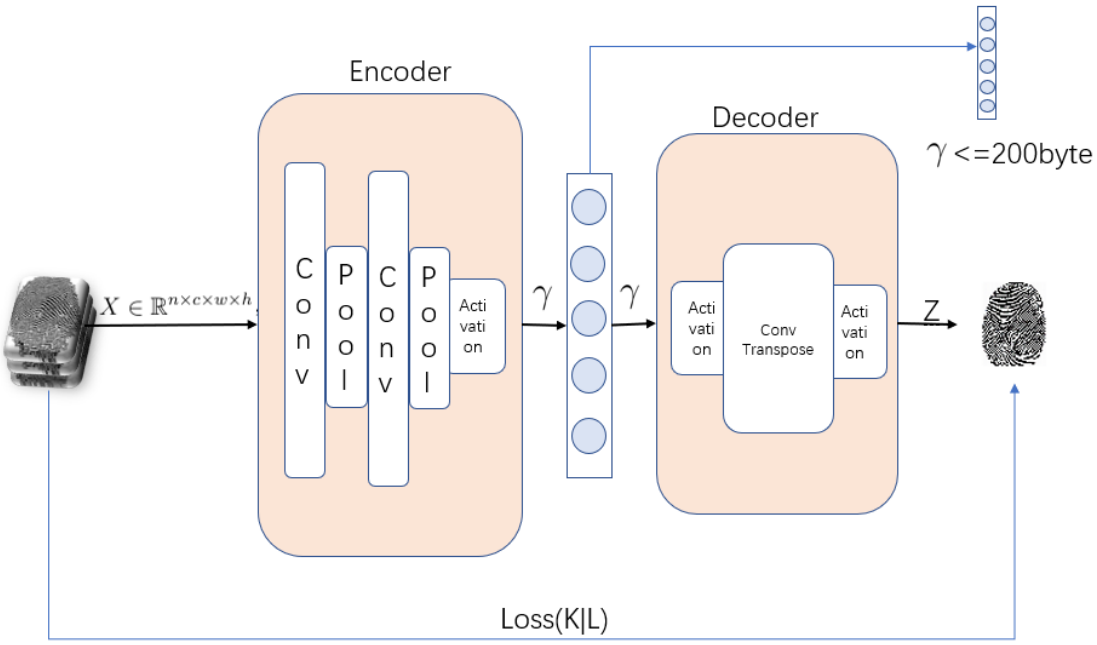


图 7　卷积自编码器 **CAE** 模型

添加稀疏约束，目标函数为：

$$\mathrm{CAE} = L(X, Z) + \gamma \sum_{j=1}^{H_D} \mathrm{KL}(\rho \| \hat{\rho}_j), \tag{12}$$

$$\mathrm{KL}(\rho \| \hat{\rho}_j) = \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j}, \tag{13}$$

其中 γ 为稀疏权重，H_D 为隐藏单元数，ρ 为稀疏参数，$\hat{\rho}_j = \frac{1}{N}\sum_{i=1}^{N} y_j(x^{(i)})$ 为平均激活。使用 Adam 优化器：

$$\begin{cases} m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla_w f(w_t), \\ v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla_w f(w_t))^2, \\ \hat{m}_t = \frac{m_t}{1-\beta_1^t}, \hat{v}_t = \frac{v_t}{1-\beta_2^t}, \\ w_t = w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \varepsilon}}, \end{cases} \tag{14}$$

其中 $\beta_1 = 0.9$，$\beta_2 = 0.999$。

5.3 实验结果及分析

输入 400×300 指纹图像，参数设置如表 1所示：

表 **1**　参数设置表

参数名称	值	参数名称	值
迭代次数 epoch	2000	吞吐量 batch	64
出/入通道数 in/out	3/3	编码大小 size	200
学习率 lr	0.01	偏置 β_1, β_2	0.9, 0.999
卷积核 k_1, k_2, k_3	3,3,4	池化层 padding	1

损失函数变化如图 8所示：

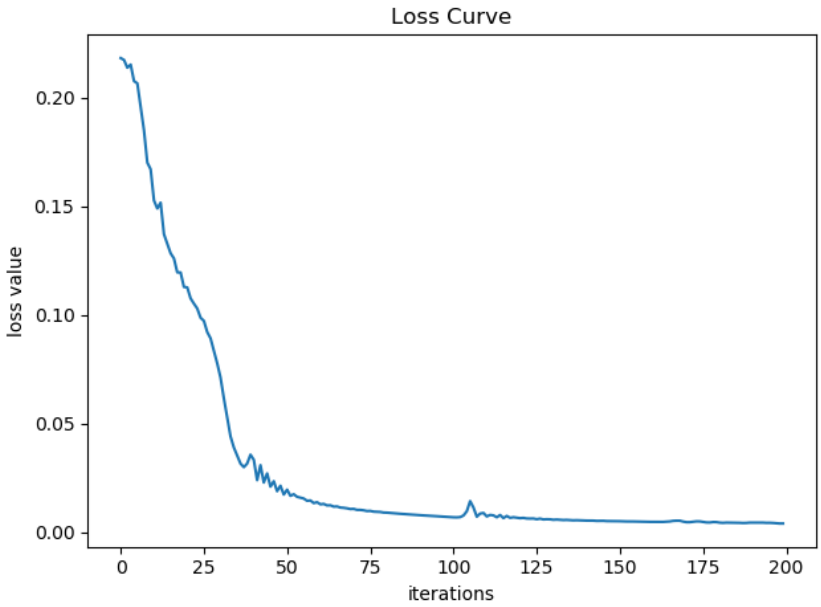
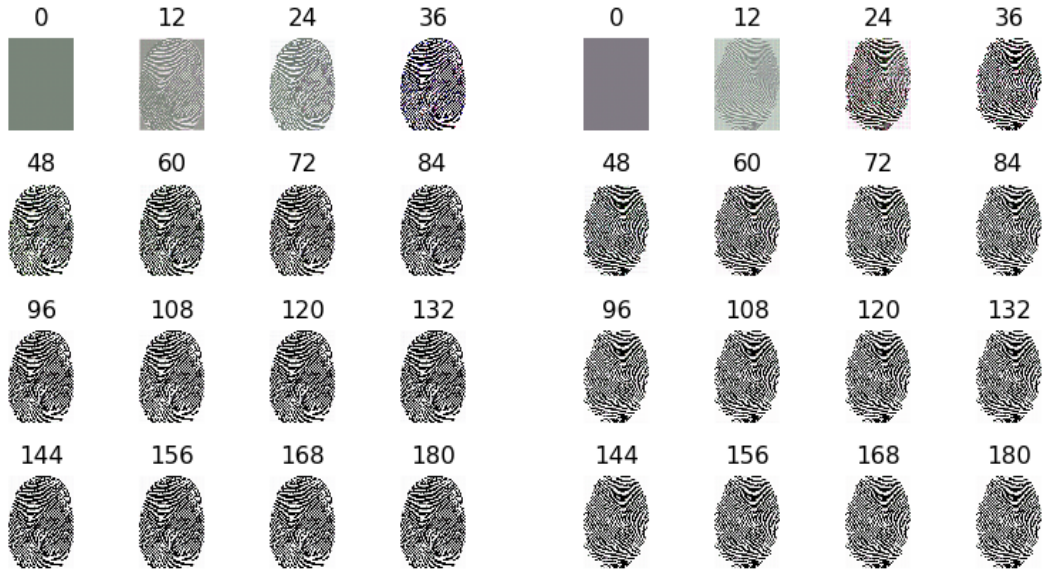


图 **8**　**CAE** 迭代过程中损失函数

信息损失仅 0.04%，优于传统编码。解码图像如图 9所示，迭代超过 120 次后清晰度接近原始图像。

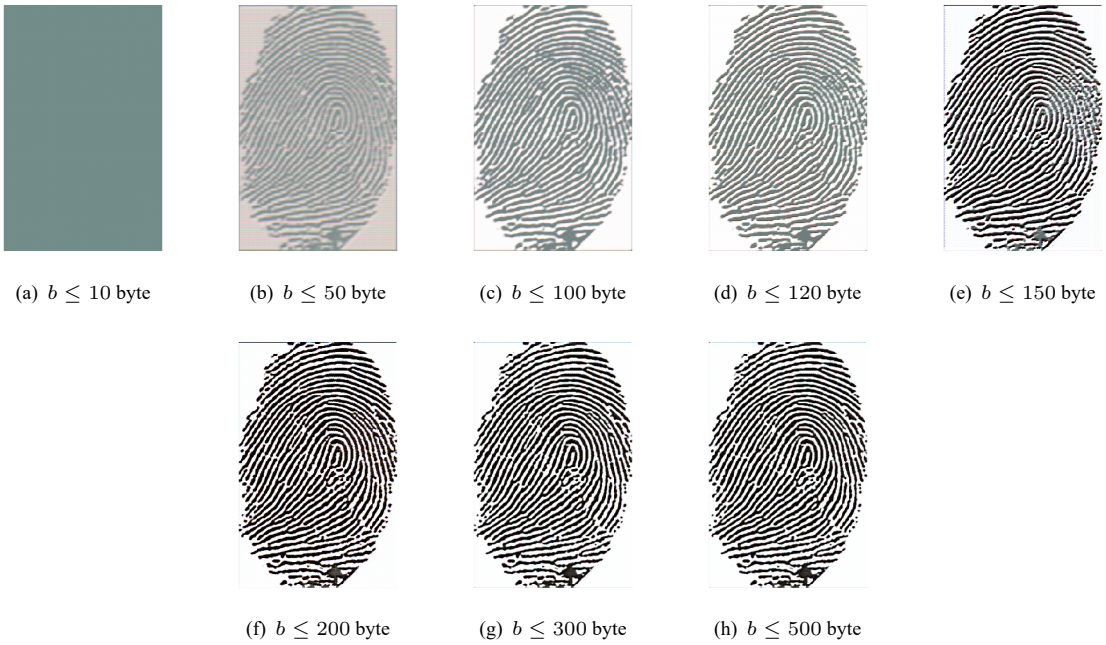


(a) 指纹 1 解码过程　　(b) 指纹 2 解码过程

图 **9**　解码器还原效果图

5.4 灵敏度分析

调整编码大小 b，解码效果如图 10所示：



(a) $b \leq 10$ byte　(b) $b \leq 50$ byte　(c) $b \leq 100$ byte　(d) $b \leq 120$ byte　(e) $b \leq 150$ byte

(f) $b \leq 200$ byte　(g) $b \leq 300$ byte　(h) $b \leq 500$ byte

图 **10**　不同编码字节限制的解码效果

50 字节可还原大致轮廓，100 字节还原微细节但有模糊，150 字节较清晰，200 字节以上可完美还原。

6 问题二模型的建立与求解

6.1 问题分析与相似度计算

问题二要求基于指纹编码分析相似性。CAE 将指纹图像转化为特征向量，相似性分析转化为向量相似度量化。设计加权相似度，归一化向量后计算谷本系数、皮尔逊相关系数和马氏距离，构造关系函数。

- **谷本系数**用于符号或布尔值相似度：

$$T(A,B) = \frac{A \cdot B}{\|A\|^2 + \|B\|^2 - A \cdot B}. \tag{15}$$

  越大表示相似度越高。

- **皮尔逊相关系数**反映线性相关性：

$$P(A,B) = \frac{\text{cov}(A,B)}{\sigma_A \cdot \sigma_B}, \tag{16}$$

  绝对值越大，线性相关性越高。

- **马氏距离**修正欧氏距离：

$$M(A,B) = \sqrt{(A-B)^T \Sigma^{-1} (A-B)}, \tag{17}$$

  越小表示相似度越高，Σ 为协方差矩阵。

相似度函数：

$$S(A,B) = \omega_1 T(A,B) + \omega_2 |P(A,B)| + \omega_3 / M(A,B), \tag{18}$$

其中 $\omega_1 + \omega_2 + \omega_3 = 1$，$S(A,B)$ 越大表示相似度越高。

6.2 实验结果及分析

取 $\omega_1 = 0.2$，$\omega_2 = 0.3$，$\omega_3 = 0.5$，相似度热图如图 11所示：

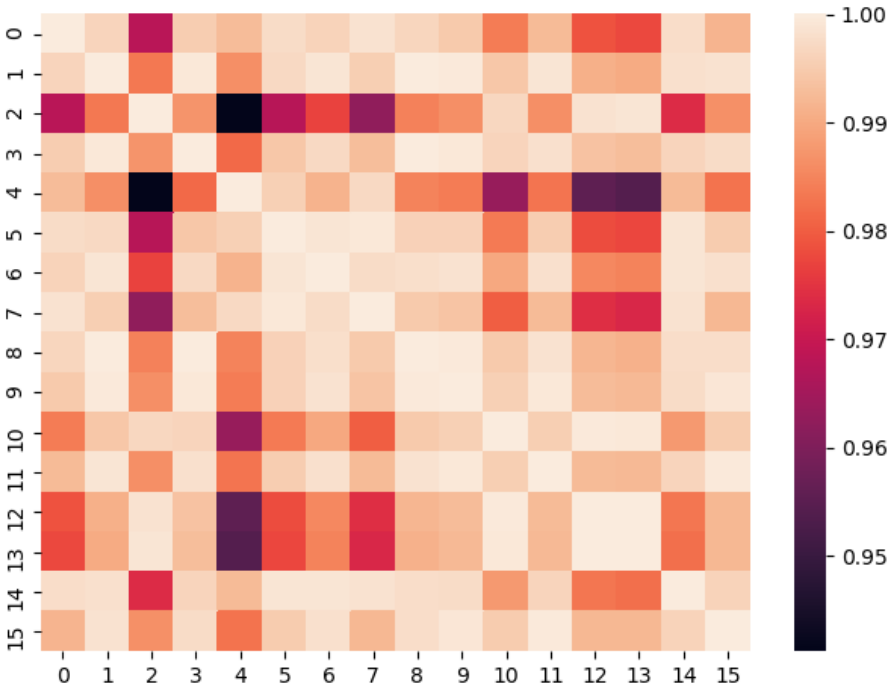
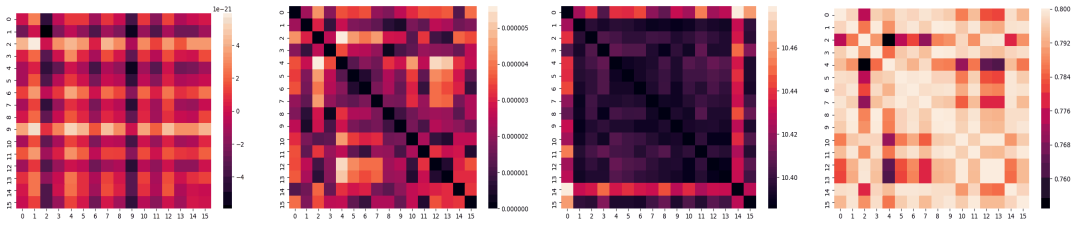


图 **11**　距离矩阵相似度热图

颜色越浅表示相似度越高，指纹对 $\{4,7\}$、$\{1,11\}$、$\{6,13\}$ 相似度高，$\{4,13\}$、$\{4,2\}$ 属于不同类型。

6.3 灵敏度分析

不同权重组合的热图如图 12所示：



(a) $\omega_i = 0.1, 0.8, 0.1$　(b) $\omega_i = 0.2, 0.5, 0.3$　(c) $\omega_i = 0.8, 0.1, 0.1$　(d) $\omega_i = 0.2, 0.3, 0.5$

图 **12**　不同权重分配的相似度热图

图 (d) 浅色区域占比适中，深色区域集中，区分度最佳，权重 $\omega_1 = 0.2, \omega_2 = 0.3, \omega_3 = 0.5$ 最优。

7 问题三模型的建立与求解

7.1 问题描述与分析

问题三要求基于问题二的相似度对 16 个指纹进行分类，属于样本分类问题。加权相似度 S 为非欧氏距离[10]，不适用 k-means 算法。本文使用 k-medoids 算法，以相似度平均值为代价函数。

7.2 k-medoids 模型的建立与求解

从 16 个样本 $\{\gamma_i\}$ 中选 k 个作为聚类中心 $\{c_i\}$，代价函数：

$$J(c_1, c_2, \ldots, c_k) = \frac{1}{16}\sum_{i=1}^{16} \max_{1\leqslant j\leqslant k} S(\gamma_i, c_j), \tag{19}$$

优化目标：

$$\max \frac{1}{16}\sum_{i=1}^{16} \max_{1\leqslant j\leqslant k} S(\gamma_i, c_j), \tag{20}$$

$$s.t.\ \{c_j \mid j \in [1,k]\} \subseteq \{\gamma_i \mid i \in [1,16]\}. \tag{21}$$

样本规模小，采用遍历搜索最优聚类中心，伪代码如下：

Algorithm 1: k-medoids 算法

```
Input: 聚类中心数: k
Output: 聚类中心: {c_i}
Initialize J_max ← 0
for π_1 = 1 to 16 − k + 1 do
    for π_2 = π_1 + 1 to 16 − k + 2 do
        ⋮
        for π_k = π_{k−1} + 1 to 16 do
            if J(γ_{π_1}, γ_{π_2}, ..., γ_{π_k}) > J_max then
                J_max = J(γ_{π_1}, γ_{π_2}, ..., γ_{π_k})
                {c_i} = {γ_{π_1}, γ_{π_2}, ..., γ_{π_k}}
            end
        end
        ⋮
    end
end
return {c_i}
```

7.3 实验结果及分析

聚类树状图如图 13所示：

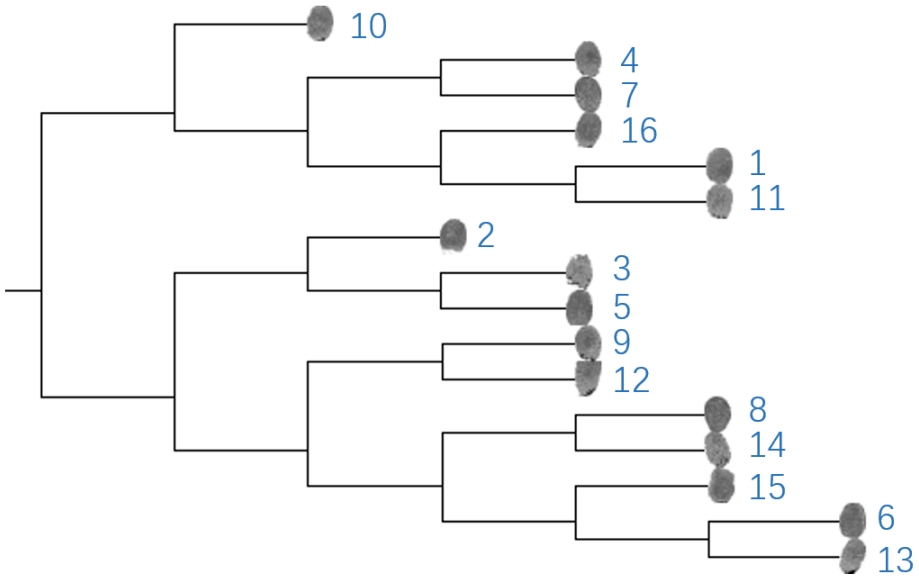


图 **13**　指纹聚类树状图

当 $k=2$，分为 $\{1,4,7,10,11,16\}$ 和 $\{2,3,5,6,8,9,12,13,14,15\}$；当 $k=4$，样本 10 独立，其余为 $\{2,3,5\}$、$\{1,4,7,11,16\}$ 和 $\{6,8,9,12,13,14,15\}$。$k=4$ 时代价函数最优，分类如图 14所示，命名为螺形纹、环形纹、弓形纹、箕形纹。

图 **14**　指纹图像的识别与分类

8 模型的评价

8.1 模型的优点

(1) 卷积核通过无监督学习提取指纹细节，压缩数据，保留关键信息。

(2) CAE 在小样本条件下超越传统编码，可高精度还原图像。

8.2 模型的缺点

小样本数据易导致过拟合，需更多样本训练。

8.3 模型总结与展望

CAE 可优化 Encoder-Decoder 结构。Lucas Theis 等人[16] 提出由 ComCNN 和 RecCNN 组成的压缩网络，ComCNN 提取紧凑表示，RecCNN 进行超分辨率重建。有损自编码器可结合概率模型 Q 分配熵编码比特数[17, 18]，在高斯尺度混合中优化低比特率失真。

参考文献

[1] Davies S G. Touching Big Brother: How biometric technology will fuse flesh and machine[J]. Information Technology & People, 2014, 7(4): 38-47.

[2] Moses K R, Higgins P, McCabe M, et al. Automated fingerprint identification system (AFIS)[J]. Scientific Working Group on Friction Ridge Analysis Study and Technology and National institute of Justice (eds.) SWGFAST-The fingerprint sourcebook, 2011: 1-33.

[3] Dror I E, Wertheim K, Fraser-Mackenzie P, et al. The impact of human–technology cooperation and distributed cognition in forensic science: biasing effects of AFIS contextual information on human experts[J]. Journal of forensic sciences, 2012, 57(2): 343-352.

[4] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint arXiv:1409.1556, 2014.

[5] Scherer R, Kalla S L, Tang Y, et al. The Grünwald–Letnikov method for fractional differential equations[J]. Computers & Mathematics with Applications, 2011, 62(3): 902-917.

[6] 邓正宏, 丁有军. 基于动态方向场的指纹图像增强算法 [J]. 微电子学与计算机, 2005, 22(2): 70-72.

[7] 蓝波, 林小竹, 籍俊伟. 一种改进的 LZW 算法在图像编码中的应用 [J]. 计算机工程与科学, 2006, 28(6): 55-57.

[8] 张旭东. 图像编码基础和小波压缩技术: 原理, 算法和标准 [M]. 清华大学出版社有限公司, 2004.

[9] 赵利平, 林涛, 周开伦. 屏幕图像压缩中串复制位移参数的高效编码算法 [J]. 计算机学报, 2017, 40(5): 1218-1228.

[10] 黄健航, 雷迎科. 基于边际 Fisher 深度自编码器的电台指纹特征提取 [J]. 模式识别与人工智能, 2017 (2017 年 11): 1030-1038.

[11] Gao L, Chen P Y, Yu S. Demonstration of convolution kernel operation on resistive cross-point array[J]. IEEE Electron Device Letters, 2016, 37(7): 870-873.

[12] Guo X, Liu X, Zhu E, et al. Deep clustering with convolutional autoencoders[C]//International conference on neural information processing. Springer, Cham, 2017: 373-382.

[13] 王万良, 杨小涵, 赵燕伟, 等. 采用卷积自编码器网络的图像增强算法 [J]. 浙江大学学报 (工学版), 2019, 53(9): 1728-1740.

[14] Arora P, Varshney S. Analysis of k-means and k-medoids algorithm for big data[J]. Procedia Computer Science, 2016, 78: 507-512.

[15] Yu D, Liu G, Guo M, et al. An improved K-medoids algorithm based on step increasing and optimizing medoids[J]. Expert Systems with Applications, 2018, 92: 464-473.

[16] Theis L, Shi W, Cunningham A, et al. Lossy image compression with compressive autoencoders[J]. arXiv preprint arXiv:1703.00395, 2017.

[17] Huszar F, Theis L, Shi W, et al. Lossy Image Compression with Compressive Autoencoders[J]. 2020.

[18] Mukherjee R, Chandran S. Lossy image compression using SVD coding, compressive autoencoders, and prediction error-vector quantization[C]//2017 4th International Conference on Opto-Electronics and Applied Optics (Optronix). IEEE, 2017: 1-5.

附录 A 第一问代码实现及可视化

卷积自编码器–Python 源代码

```
import torch
import torch.nn as nn
from PIL import Image
import matplotlib.pyplot as plt
import math
import numpy as np

def read_single_to_tensor(image_path, device):
    """
    read single image and convert to tensor
    Args:
        image_path: image'path with name
    Returns:
        image tensor on gpu with shape (1, 3, w, h)
    """
    # 1: read and convert to tensor on gpu
    image = Image.open(image_path).convert('RGB')
    image = torch.from_numpy(np.array(image)).to(device)
    image = image
    # image = image.expand(image.shape[0],image.shape[1],3)
    # 2:transpose from (w, h, 3) into (3, w, h)
    image = image.permute(2, 1, 0)

    # 3:add dim from (3, w, h) into (1, 3, w, h)
    image = image.unsqueeze(0).float()
    image = image/255
    print(image.shape)
    return image


class AE(nn.Module):
    """ Auto Encoder """
    def __init__(self, in_channels, out_channels, encode_dim):
        super().__init__()
        self.in_channels = in_channels
        self.out_channels = out_channels
        self.curr_dim = encode_dim
        self.en_net = nn.Sequential(*self.build_encoder())
        self.de_net = nn.Sequential(*self.build_decoder())

    def build_encoder(self):
        encoder = []
```

```
        encoder.append(nn.Conv2d(in_channels=self.in_channels, out_channels=self.curr_dim,
                                 kernel_size=3, stride=1, padding=1))
        for i in range(2):
            encoder.append(nn.Conv2d(in_channels=self.curr_dim, out_channels=self.curr_dim*2,
                                     stride=2, padding=1, kernel_size=3))
            encoder.append(nn.ReLU())
            self.curr_dim = self.curr_dim * 2
        return encoder

    def build_decoder(self):
        decoder = []
        for i in range(2):
            decoder.append(nn.ConvTranspose2d(in_channels=self.curr_dim,
                out_channels=int(self.curr_dim/2),
                                              kernel_size=4, stride=2, padding=1))
            decoder.append(nn.ReLU())
            self.curr_dim = int(self.curr_dim/2)
        decoder.append(nn.Conv2d(in_channels=self.curr_dim, out_channels=self.out_channels,
                                 kernel_size=3, padding=1, stride=1))
        decoder.append(nn.Sigmoid())
        return decoder

    def forward(self, x):
        x = self.en_net(x)
        out = self.de_net(x)
        return out, x

# 1: options
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
total_iterations = 200
image_save_frequency = int(total_iterations/16)

# 2:data prepare
image_tensor = read_single_to_tensor(r"C:\Users\77526\Desktop\lzc\资料\DEMO\data\result10.png",
    device=device)
# i_w, i_h = image_tensor.shape[2], image_tensor.shape[3]
result_images = []
result_iterations = []
record_loss = []
record_iterations = []

# 3:network define
ae_net = AE(in_channels=3, out_channels=3, encode_dim=8).to(device)
print(ae_net)
# 4:optimizer define
optimizer = torch.optim.Adam(ae_net.parameters(), lr=0.01, betas=(0.9, 0.999))
```

```
# 5:loss define
mse_loss = nn.MSELoss()

# 6: plot function
def plot_list_image(image_list, title_list):
    """
    plot a list of image with title
    Args:
        image_list: a list of image with numpy type
        title_list: a list of title with str type

    Returns:
        True after plot
    """
    plt.figure(figsize=(4, 4))
    w_h = int(math.sqrt(len(image_list)))
    for i in range(w_h*w_h):
        plt.subplot(w_h, w_h, i+1)
        plt.axis("off")
        plt.imshow(image_list[i])
        plt.title(title_list[i])
    plt.show()
    return True


if __name__ == "__main__":
    for iteration in range(total_iterations):
        recon_image, encode = ae_net(image_tensor)
        loss = mse_loss(recon_image, image_tensor)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        print("iter: {}, loss: {}".format(iteration, loss.item()))
        record_loss.append(loss.item())
        record_iterations.append(iteration)
        recon_image = recon_image.squeeze(1)
        # print(recon_image.shape)
        if iteration % image_save_frequency == 0:
            result_images.append(recon_image.detach().cpu().numpy().squeeze().transpose(2,1, 0))
            result_iterations.append(str(iteration))
    plot_list_image(result_images, result_iterations)
    plt.plot(record_iterations, record_loss)
    plt.xlabel("iterations")
    plt.ylabel("loss value")
    plt.title("Loss Curve")
    plt.show()
```

附录 B 第二、三问代码实现及可视化

k-medoids 模型–Python 源代码

```
from itertools import combinations
from PIL import Image
import numpy as np
from numpy import *
from pylab import *
from PIL import Image, ImageDraw
import os
import seaborn as sns

class ClusterNode(object):
    def __init__(self, vec, left, right, distance=0.0, count=1):
        self.left = left
        self.right = right
        self.vec = vec
        self.distance = distance
        self.count = count # 只用于加权平均

    def extract_clusters(self, dist):
        if self.distance < dist:
            return [self]
        return self.left.extract_clusters(dist) + self.right.extract_clusters(dist)

    def get_cluster_elements(self):
        return self.left.get_cluster_elements() + self.right.get_cluster_elements()

    def get_height(self):
        return self.left.get_height() + self.right.get_height()

    def get_depth(self):
        return max(self.left.get_depth(), self.right.get_depth()) + self.distance

    def draw(self, draw, x, y, s, imlist, im):
        """ 用图像缩略图递归地画出叶节点 """
        h1 = int(self.left.get_height() * 20 / 2)
        h2 = int(self.right.get_height() * 20 / 2)
        top = y - (h1 + h2)
        bottom = y + (h1 + h2)
        # 子节点垂直线
        draw.line((x, top + h1, x, bottom - h2), fill=(0, 0, 0))
        # 水平线
        ll = self.distance * s
        draw.line((x, top + h1, x + ll, top + h1), fill=(0, 0, 0))
```

```
        draw.line((x, bottom - h2, x + ll, bottom - h2), fill=(0, 0, 0))
        # 递归地画左边和右边的子节点
        self.left.draw(draw, x + ll, top + h1, s, imlist, im)
        self.right.draw(draw, x + ll, bottom - h2, s, imlist, im)


class ClusterLeafNode(object):

    def __init__(self, vec, id):
        self.vec = vec
        self.id = id

    def extract_clusters(self, dist):
        return [self]

    def get_cluster_elements(self):
        return [self.id]

    def draw(self, draw, x, y, s, imlist, im):
        nodeim = Image.open(imlist[self.id])
        nodeim.thumbnail([20, 20])
        print(self.id)
        ns = nodeim.size
        im.paste(nodeim, [int(x), int(y - ns[1] // 2), int(x + ns[0]), int(y + ns[1] - ns[1] //
            2)])

    def get_height(self):
        return 1

    def get_depth(self):
        return 0


def multipl(a,b):
    sumofab=0.0
    for i in range(len(a)):
        temp=a[i]*b[i]
        sumofab+=temp
    return sumofab


def L2dist(v1, v2):
    return sqrt(sum((v1 - v2) ** 2))


def L1dist(v1, v2):
    return sum(v1 - v2)
```

```
def corrcoefdist(x, y):
    sum1, sum2, n=sum(x), sum(y), len(x)
    sumofxy=multipl(x,y)
    sumofx2 = sum([pow(i,2) for i in x])
    sumofy2 = sum([pow(j,2) for j in y])
    num=sumofxy-(float(sum1)*float(sum2)/n)
    den=sqrt((sumofx2-float(sum1**2)/n)*(sumofy2-float(sum2**2)/n))
    return num/den


def mixdist(v1, v2):
    return corrcoefdist(v1,v2)+L2dist(v1,v2)


def hcluster(features, distfcn=corrcoefdist):
    """ 用层次聚类对行特征进行聚类 """
    # 用于保存计算出的距离
    distances = {}
    # 每行初始化为一个簇
    node = [ClusterLeafNode(array(f), id=i) for i, f in enumerate(features)]
    ddd = {}
    for i,ni in enumerate(node):
        for j,nj in enumerate(node):
            ddd[i,j] = distfcn(ni.vec, nj.vec)

    while len(node) > 1:
        closest = float('Inf')
        # 遍历每对，寻找最小距离
        for ni, nj in combinations(node, 2):
            if (ni, nj) not in distances:
                distances[ni, nj] = distfcn(ni.vec, nj.vec)
            d = distances[ni, nj]
            if d < closest:
                closest = d
                lowestpair = (ni, nj)
        ni, nj = lowestpair
        #  对两个簇求平均
        new_vec = (ni.vec + nj.vec) / 2.0
        #  创建新的节点
        new_node = ClusterNode(new_vec, left=ni, right=nj, distance=closest)
        node.remove(ni)
        node.remove(nj)
        node.append(new_node)
    return node[0], ddd
```

```
def draw_dendrogram(node, imlist, filename='clusters.jpg'):
    rows = node.get_height() * 20
    cols = 600
    # 距离缩放因子，以便适应图像宽度
    s = float(cols - 150) / node.get_depth()
    # 创建图像，并绘制对象
    im = Image.new('RGB', (cols, rows), (255, 255, 255))
    draw = ImageDraw.Draw(im)
    # 初始化树开始的线条
    draw.line((0, rows / 2, 20, rows / 2), fill=(0, 0, 0))
    # 递归地画出节点
    node.draw(draw, 20, (rows / 2), s, imlist, im)
    im.save(filename)
    im.show()


if __name__ == "__main__":
    path = r'C:\Users\77526\Desktop\lzc\资料\DEMO\data'
    imlist = [os.path.join(path, f) for f in os.listdir(path) if f.endswith('.png')]

    # 提取特征向量，每个颜色通道量化成 8 个小区间
    features = zeros([len(imlist), 512])
    for i, f in enumerate(imlist):
        im = array(Image.open(f))
        # 多维直方图
        h, edges = histogramdd(im.reshape(-1, 3), 8, normed=True, range=[(0, 255), (0, 255), (0,
            255)])
        features[i] = h.flatten()
    tree, distances = hcluster(features)
    dis = zeros((16,16))
    for key in distances.keys():
        i,j = key
        dis[i,j] = distances[key]
    ax = sns.heatmap(dis)
    plt.show()

    draw_dendrogram(tree, imlist, filename='sunset.png')
```