



模式识别课程报告

基于数据驱动的特征提取的道路分割实验研究

班级 _____ 模式识别 (1 班) _____

学号 _____ 23060189 _____

姓名 _____ 朱睿 _____

2023 年 12 月 19 日

目录

1 实验目的	3
1.1 实验目标	3
1.2 实验涉及到的学习内容	3
2 实验具体完成情况	3
2.1 实验总体方案设计	3
2.2 具体技术途径	4
2.2.1 数据准备：构建真值图像，用于后续的算法评估	4
2.2.2 数据预处理：对于初始图像，采用不同滤波方式对图像进行预处理	5
2.2.3 特征提取：采用不同方法进行特征提取，用于后续的计算	5
2.2.4 流形学习方法可视化：采用不同方法对特征提取效果进行可视化	8
2.2.5 分类算法（道路分割）：采用不同的算法进行道路分割，并进行比对	8
2.2.6 数据后处理：采用不同方式对分割后的道路进行进一步优化，提升分割效果	9
3 实验结果与分析	10
3.1 数据准备：真值图像构建	10
3.2 数据预处理：滤波	10
3.3 特征提取：采用 PCA、KPCA、LDA、Dictionary_Learning	11
3.4 流形学习方法：评估特征提取效果	13
3.5 分类算法：道路分割	20
3.6 数据后处理：采用形态学方法优化结果	23
4 流形学习各个参数的探究	24
5 实验心得与体会	24
6 存在的主要问题和建议	25
7 附录	25

1 实验目的

1、实验目标

利用至少两种不同的特征提取算法，对第一次作业中的道路图像中的路面进行特征建模。要求如下

- 利用任何一款流形学习方法（可直接使用，无需自己编程）对两种特征的有效性进行可视化对比分析
- 分别利用自己提取的特征对道路进行识别，并评估流形学习可视化得到的特征有效性评估结果与实际道路识别效果之间的相关性

2、实验涉及到的学习内容

- 基于数据驱动的特征提取方法
- 使用流形学习方法对提取到的特征进行可视化

2 实验具体完成情况

1、实验总体方案设计

实验需要利用聚类算法对道路进行二值化分割，具体分为如下 6 步

- **数据准备**: 构建真值图像，用于后续的算法评估
- **数据预处理**: 对于初始图像，采用中值滤波、均值滤波、高斯滤波等方式对图像进行预处理
- **特征提取**: 采用不同方法进行特征提取，用于后续的计算，例如原始 RGB 通道特征、主成分分析 (PCA)、核主成分分析 (KPCA)、线性判别 (LDA)、字典学习 (Dictionary Learning) 等
- **流形学习方法可视化**: 采用 t-SNE (t-Distributed Stochastic Neighbor Embedding)、等度量特征映射 (Isometric Feature Mapping, Isomap)、局部线性嵌入 (Locally Linear Embedding, LLE) 等方法对特征提取效果进行可视化
- **分类算法 (道路分割)**: 采用不同的算法进行道路分割，并进行比对，例如支持向量机 (SVM)、
- **数据后处理**: 采用开运算、闭运算及二者结合的方式对分割后的道路进行进一步优化，提升分割效果

由于本人对 python 较为感兴趣，且后续科研工具主要为 python 语言，因此报告主要基于 python 完成上述任务。

2、具体技术途径

针对实验方案中的 6 个步骤，基本原理和实现方法如下：

(1) 数据准备：构建真值图像，用于后续的算法评估

SAM-Adapter 近期大模型的涌现给 AI 研究带来显著的发展，META 的工作 Segment Anything (SAM)，就是其中一个为图像分割任务设计的基础大模型。SAM 是一种交互型的图像分割大模型，通过提供的 prompt 如点、框、文本描述等粗略的提示，就可以分割出图像中指定的目标，其 demo 的效果十分惊艳。然而在某些特殊场景的图片上并不会带来如此惊艳的效果，可能是由训练数据的差异性导致。

针对于图像分割特定任务，我们采用 SAM-Adapter，其通过设计一个 Adapter 模块，使得在不微调 SAM 网络的情况下，将领域特定的信息或视觉提示注入到分割网络中，从而提高 SAM 在特定任务上的性能。

评估指标 在实验中，我们构建了 4 个评估指标，分别为准确率 (Accuracy)、精确率 (Precision)、召回率 (Recall)、F1 值，其计算方法如下

我们采用 TP (True Positive, 真阳性)、FP (False Positive, 假阳性)、TN (True Negative, 真阴性) 和 FN (False Negative, 假阴性) 来描述不同结果的样本的数量。

准确率 (Accuracy)：准确率衡量模型预测的整体正确性，即正确预测的样本数与总样本数之比。准确率的计算方法如下：

$$\text{Accuracy} = \frac{TP + TN}{TP + TN}$$

精确率 (Precision)：精确率衡量模型在预测为正类的样本中的正确性，即真阳性样本数与所有预测为正类的样本数之比。精确率的计算方法如下：

$$\text{Precision} = \frac{TP}{TP + FP}$$

精确率用于评估模型的预测准确性，高精确率表示模型将负样本误判为正样本的可能性较低。

召回率 (Recall)：召回率衡量模型对正类样本的预测覆盖率，即真阳性样本数与所有真实正类样本数之比。召回率的计算方法如下：

$$\text{Recall} = \frac{TP}{TP + FN}$$

召回率用于评估模型的查全率，高召回率表示模型正确检测到了大部分正样本。

F1 值: F1 值是精确率和召回率的调和平均值，综合考虑了模型的准确性和覆盖率。F1 值的计算方法如下：

$$F1 = \frac{2 \times (Precision \times Recall)}{Precision + Recall}$$

F1 值的范围在 0 到 1 之间，值越高表示模型在准确性和覆盖率之间取得了更好的平衡。

(2) 数据预处理：对于初始图像，采用不同滤波方式对图像进行预处理

中值滤波 (Median Filtering) 中值滤波是一种非线性滤波方法，用于图像去噪。它的原理是将像素点周围的邻域像素值进行排序，然后用中间值来代替当前像素的值。中值滤波对于去除图像中的椒盐噪声或斑点噪声效果良好，因为它能够有效地消除异常值的影响。

均值滤波 (Mean Filtering) 均值滤波是一种线性滤波方法，用于平滑图像并减少噪声。它的原理是将像素点周围的邻域像素值取平均值，然后用该平均值来代替当前像素的值。均值滤波对于高斯噪声等均值为零的噪声效果较好，但在去除椒盐噪声等具有较高峰值的噪声时效果较差。

高斯滤波 (Gaussian Filtering) 高斯滤波是一种线性滤波方法，用于图像平滑和去噪。它的原理是将像素点周围的邻域像素值按照高斯分布进行加权平均，然后用加权平均值来代替当前像素的值。高斯滤波对于平滑图像并保留边缘信息效果较好，因为它能够考虑到像素之间的空间关系。

双边滤波 (Bilateral Filtering) 双边滤波是一种非线性滤波方法，用于图像平滑和去噪。它的原理是同时考虑像素之间的空间距离和像素值之间的差异，通过加权平均来计算当前像素的值。双边滤波在保留图像边缘信息的同时能够有效地去除噪声，适用于保持图像细节和边缘清晰的应用场景。

均值迁移滤波 (Mean Shift Filtering) 均值迁移滤波是一种基于样本点密度的非线性滤波方法，用于图像平滑和分割。它通过在像素点周围的邻域内计算样本点的均值偏移来对图像进行平滑处理。均值迁移滤波对于保持图像细节和纹理信息的同时能够有效地去除噪声和进行图像分割。

这些滤波方法在图像处理领域中被广泛应用，每种方法都有其适用的场景和特点。选择合适的滤波方法取决于图像的特性、噪声类型以及对图像细节和边缘保留的要求。

(3) 特征提取：采用不同方法进行特征提取，用于后续的计算

原始 RGB 通道特征 原始 RGB 通道特征是通过直接使用图像的原始 RGB 颜色通道作为特征表示的方法。对于一张 RGB 图像，它包含了红色 (R)、绿色 (G) 和蓝色 (B)

三个颜色通道。原始 RGB 通道特征将每个像素点的 RGB 值作为特征向量的元素。假设图像的尺寸为 $M \times N$, 则特征向量的维度为 $M \times N \times 3$ 。

主成分分析 (PCA) 主成分分析是一种常用的降维技术, 通过线性变换将高维数据转换为低维表示。它通过找到数据中最大方差的方向 (主成分) 来实现降维。给定一个包含 N 个样本的数据集 X , 其中每个样本 $x_i \in \mathbb{R}^d$, PCA 的目标是找到投影矩阵 W , 将原始数据映射到新的低维空间, 使得投影后的数据具有最大的方差。

PCA 的数学表达式如下:

- 计算数据的协方差矩阵 C :

$$C = \frac{1}{N} \times \sum (x_i - \mu)(x_i - \mu)^T$$

其中, μ 是数据的均值向量。

- 对协方差矩阵 C 进行特征值分解得到特征值 $\lambda_1, \lambda_2, \dots, \lambda_k$ 和对应的特征向量 v_1, v_2, \dots, v_k 。
- 将特征向量按照对应的特征值降序排列, 选择前 k 个特征向量组成投影矩阵 W 。
- 将原始数据 x 映射到低维空间:

$$y = W^T(x - \mu)$$

核主成分分析 (KPCA) 核主成分分析是对传统 PCA 方法的扩展, 用于处理非线性关系的数据降维。KPCA 通过使用核函数将数据映射到高维特征空间, 然后在高维空间中进行 PCA 分析。这种非线性映射使得 KPCA 能够提取非线性特征。

KPCA 的数学表达式如下:

- 计算核矩阵 K :

$$K_{ij} = k(x_i, x_j)$$

其中, $k(\cdot, \cdot)$ 是核函数, x_i 和 x_j 是数据集中的样本。

- 对核矩阵 K 进行中心化, 得到中心化核矩阵 \tilde{k} 。
- 对中心化核矩阵 \tilde{k} 进行特征值分解, 得到特征值 $\lambda_1, \lambda_2, \dots, \lambda_k$ 和对应的特征向量 $\alpha_1, \alpha_2, \dots, \alpha_k$ 。
- 将原始数据 x 映射到低维空间:

$$y = \sum \alpha_i k(x, x_i)$$

线性判别分析(LDA) 线性判别分析是一种用于特征提取和降维的线性分类方法。LDA 通过寻找投影方向，使得在降维后的空间中，不同类别的样本尽可能分开，同类样本尽可能接近。

假设有 N 个样本，每个样本 $x_i \in \mathbb{R}^d$ ，对应的类别标签为 $y_i \in 1, 2, \dots, C$ ，其中 C 表示类别的数量。

LDA 的数学表达式如下：

- 计算类别均值向量：

$$\mu_i = \frac{1}{N_i} \times \sum_j x_j$$

其中， N_i 是类别 i 的样本数量。

- 计算类内散度矩阵 S_w 和类间散度矩阵 S_b ：

$$S_w = \sum_i \sum_x (x_j - \mu_i)(x_j - \mu_i)^T$$

$$S_b = \sum_i N_i (\mu_i - \mu)(\mu_i - \mu)^T$$

其中， μ 是所有样本的均值向量。

- 对矩阵 $S_w^{-1} S_b$ 进行特征值分解，得到特征值 $\lambda_1, \lambda_2, \dots, \lambda_k$ 和对应的特征向量 w_1, w_2, \dots, w_k 。
- 将特征向量按照对应的特征值降序排列，选择前 k 个特征向量组成投影矩阵 W 。
- 将原始数据 x 映射到低维空间：

$$y = W^T x$$

字典学习 (Dictionary Learning) 字典学习是一种无监督学习方法，用于从数据中学习一组基础原子（字典），以表示数据。字典学习的目标是通过最小化数据的重构误差，学习出能够有效表示数据的原子集合。给定一个包含 N 个样本的数据集 X ，其中每个样本 $x_i \in \mathbb{R}^d$ ，字典学习的目标是找到一个字典 $D = [d_1, d_2, \dots, d_k]$ ，其中 $d_j \in \mathbb{R}^d$ 为字典的原子，以及一个稀疏编码矩阵 $C \in \mathbb{R}^{k \times N}$ ，使得 $X \approx DC$ ，其中 k 是字典的大小。

字典学习的数学表达式如下：

- 初始化字典 D 和稀疏编码矩阵 C 。
- 迭代优化以下两个步骤直至收敛：

- 固定字典 D ，更新稀疏编码矩阵 C ：

$$\min ||X - DC||_2^2 + \lambda ||C||_1$$

$\min \|X - DC\|^2 + \|\mathbf{C}\|$ 其中, λ 是控制稀疏性的参数。

b. 固定稀疏编码矩阵 C , 更新字典 D :

$$\min \|X - DC\|_2^2 \quad s.t. \|d_j\|_2 \leq 1, \forall j$$

通过迭代优化, 字典学习可以学到数据的稀疏表示, 以及适合于表示数据的原子字典。字典学习在图像去噪、图像压缩、图像恢复等任务中有广泛应用, 能够自动学习到有效的特征表示。

(4) 流形学习方法可视化: 采用不同方法对特征提取效果进行可视化

流形学习 (Manifold Learning) 是一种无监督学习方法, 用于从高维数据中发现其潜在的低维流形结构。在高维空间中, 数据可能呈现复杂的非线性关系, 而流形学习的目标是通过学习数据的流形结构, 将高维数据映射到低维空间, 以便更好地理解和可视化数据。以下是三种常见的流形学习方法的介绍:

t-SNE (t-Distributed Stochastic Neighbor Embedding) t-SNE 是一种流行的降维和可视化方法, 通过保留数据样本之间的相似性关系, 将高维数据映射到二维或三维空间。t-SNE 的核心思想是, 在低维空间中, 通过高斯分布来表示样本之间的相似性, 而在高维空间中, 则通过 t 分布来表示相似性。t-SNE 使用梯度下降的方法最小化高维空间和低维空间之间的 Kullback-Leibler (KL) 散度, 从而实现降维和可视化。

Isomap (Isometric Feature Mapping) Isomap 是一种基于流形学习的降维方法, 它通过保持数据点之间的测地距离来捕捉数据的流形结构。Isomap 中的关键步骤是构建数据点之间的邻接图, 并计算它们之间的最短路径距离。然后, 使用多维缩放 (MDS) 算法将最短路径距离映射到低维空间, 得到最终的降维结果。

LLE (Locally Linear Embedding) LLE 是一种非线性的流形学习方法, 它假设数据在局部区域内近似线性, 并通过保持局部线性关系来进行降维。LLE 的主要步骤包括两个阶段: 首先, 对每个数据点找到其最近的邻居; 然后, 通过最小化数据点与其邻居之间的线性重构误差, 计算低维表示。LLE 的优点是能够捕捉复杂的流形结构, 但对于大规模数据集可能计算复杂度较高。

这些流形学习方法在降维和可视化高维数据时都具有一定的优势和适用性。选择合适的方法取决于数据的特点以及所需的降维效果。

(5) 分类算法 (道路分割): 采用不同的算法进行道路分割, 并进行比对

支持向量机 (Support Vector Machine, SVM) 支持向量机是一种监督学习算法, 可用于分类和回归问题。在道路分割中, SVM 可以用于将图像像素分类为道路和非道

路。SVM 通过在特征空间中找到最优的超平面来实现分类。它通过将数据映射到高维空间，并找到能够最大化类别间间隔的超平面来进行分类。在道路分割中，SVM 可以利用图像的颜色、纹理和边缘等特征来判断像素是否属于道路。

随机森林 (Random Forest) 随机森林是一种集成学习方法，它由多个决策树组成。在道路分割中，随机森林可以用于像素级别的分类。每个决策树通过选择随机的特征子集和随机的样本子集进行训练。在预测阶段，随机森林中的多个决策树投票决定像素的类别。随机森林具有较好的鲁棒性和泛化能力，能够处理复杂的特征和数据不平衡问题。

K 最近邻算法 (K-Nearest Neighbors, KNN) K 最近邻算法是一种基于实例的学习算法，它在道路分割中常用于像素级别的分类。KNN 通过计算距离来确定一个未知样本的类别，即找出其最近的 K 个邻居，并根据邻居的类别进行投票决策。在道路分割中，KNN 可以根据图像像素的颜色、纹理和位置等特征，将其分类为道路或非道路。KNN 算法简单易实现，但对于大规模数据集可能计算复杂度较高。

(6) 数据后处理：采用不同方式对分割后的道路进行进一步优化，提升分割效果

数据后处理，我们采用**形态学处理**的方式。形态学方法是一种基于数学形态学理论的图像处理技术，用于图像的形状分析和特征提取。形态学方法主要应用于二值图像，通过定义结构元素（也称为模板或核）和基本的形态学操作来改变图像的形状和结构。

形态学方法包括一系列的形态学操作，其中最常用的是开运算 (Opening) 和闭运算 (Closing)。

开运算 (Opening) 开运算是先进行腐蚀操作，然后再进行膨胀操作。它主要用于去除二值图像中的小型噪点和细小的边缘连接。开运算能够平滑图像，同时保持主要对象的形状和结构。具体步骤如下：

- 腐蚀：使用结构元素对图像进行腐蚀操作，使边界向内缩小。
- 膨胀：对腐蚀结果进行膨胀操作，恢复对象的大小并平滑边界。

开运算的特点：

- 去除小型噪点和细小边缘连接。
- 平滑对象的形状。
- 保持主要对象的大小和结构。

闭运算 (Closing) 闭运算是先进行膨胀操作，然后再进行腐蚀操作。它主要用于填充图像中的小型空洞和细小的断开区域。闭运算能够平滑图像，同时保持主要对象的形状和结构。具体步骤如下：

- 膨胀：使用结构元素对图像进行膨胀操作，填充空洞和连接断开的区域。
- 腐蚀：对膨胀结果进行腐蚀操作，恢复对象的大小并平滑边界。

闭运算的特点：

- 填充小型空洞和细小断开区域。
- 平滑对象的形状。
- 保持主要对象的大小和结构。

先开运算后闭运算 这种操作顺序的特点是先将较小的噪声和细小结构从图像中去除，然后再对图像进行平滑和连接，以保持主要对象的形状。

先闭运算后开运算 这种操作顺序的特点是先将空洞和断开区域填充和连接，然后再对图像进行平滑和去除细小结构，以保持主要对象的形状。

3 实验结果与分析

1、数据准备：真值图像构建

我们利用成熟的 SAM-Adapter 方法（这是一种基于预训练大模型的图像分割方法，具备很好的 0 样本推理能力），对图像道路进行分割，可以得到道路的 mask 标签，如图1所示

2、数据预处理：滤波

在先前的实验中，我们发现在分割结果中，总是会有一些噪点，经过分析，我们发现：道路通常是较为亮的点，即 RGB 的值均比较高的点，而在非道路中，也可能存在若干点亮度也较高，其与道路样本更加接近，使得可能把一些小的孤立点分割为道路；另一方面，道路中也存在一些颜色较暗的点，其与非道路样本更加接近，使得道路上的一些暗点可能被分割为非道路。

为此，我们计划采用滤波的方式，对原始图像进行预处理。一方面，**非道路的亮点**会与周围的非道路样本更为接近，降低其被分割为道路的可能性；另一方面，**道路的暗点**会与周围的道路样本更加接近，增加其被分割为道路的可能性。

在实验中，采用的滤波方式有**中值滤波**、**均值滤波**、**高斯滤波**、**双边滤波**、**均值迁移滤波**，我们设置的滤波参数如表 1 所示，原先我们计划对于滤波设置不同的参数，对

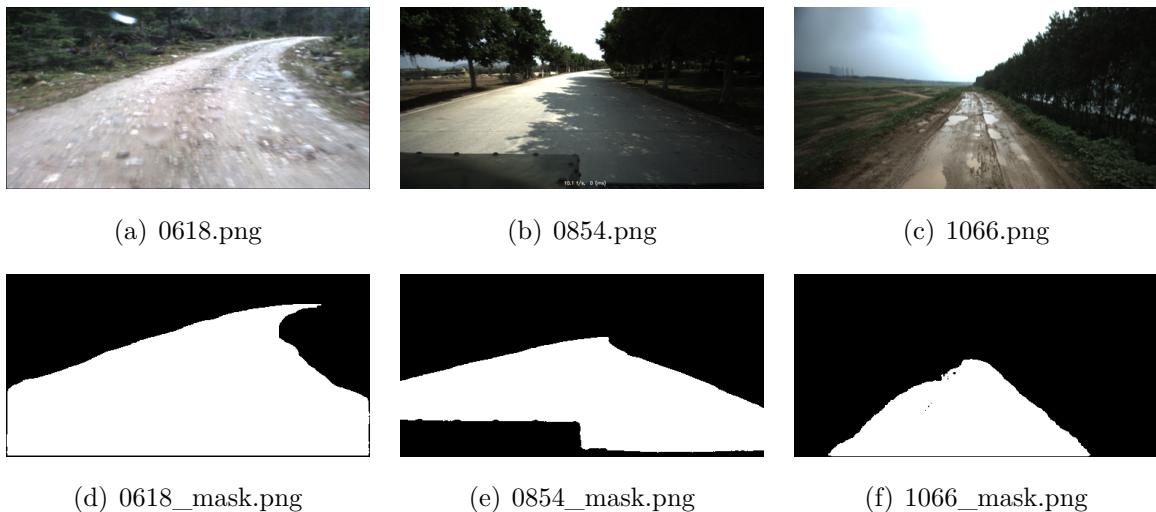


图 1: 道路 mask 标签示意图

不同滤波参数也进行比较，考虑到实验量较大，且本次实验的核心不在于此，因此，我们精心为每一种滤波仅挑选了一组参数进行实验。

表 1: 滤波实验参数设置

方法	参数设置
中值滤波	ksize=5
均值滤波	ksize=(5,5)
高斯滤波	ksize=(5,5)
双边滤波	d=9, sigmaColor=75, sigmaSpace=75
均值迁移滤波	sp=10, sr=50

图 2展示了在表 1参数设置下，滤波前后的图像对比，具体滤波形式和图像类型均在图中进行了详尽的描述。

3、特征提取：采用 PCA、KPCA、LDA、Dictionary_Learning

在本次实验中，我们主要采用了主成分分析（PCA）、核主成分分析（KPCA）、线性判别分析（LDA）和字典学习（Dictionary Learning），在实验中，我们设置不同的参数，具体实验设置如表 2所示

LDA 的参数 $n_components$ 只能为 1，这是因为其在运算中 $n_components \leq \min(features, n_classes)$

通过计算，我们可以得出，总共对 $1 + 2 + 2 \times 4 + 1 + 2 = 14$ 组特征提取参数进行实验对比。加之前面有 5 种滤波方式，那么总共有 $5 \times 14 = 70$ 种不同的实验设置。

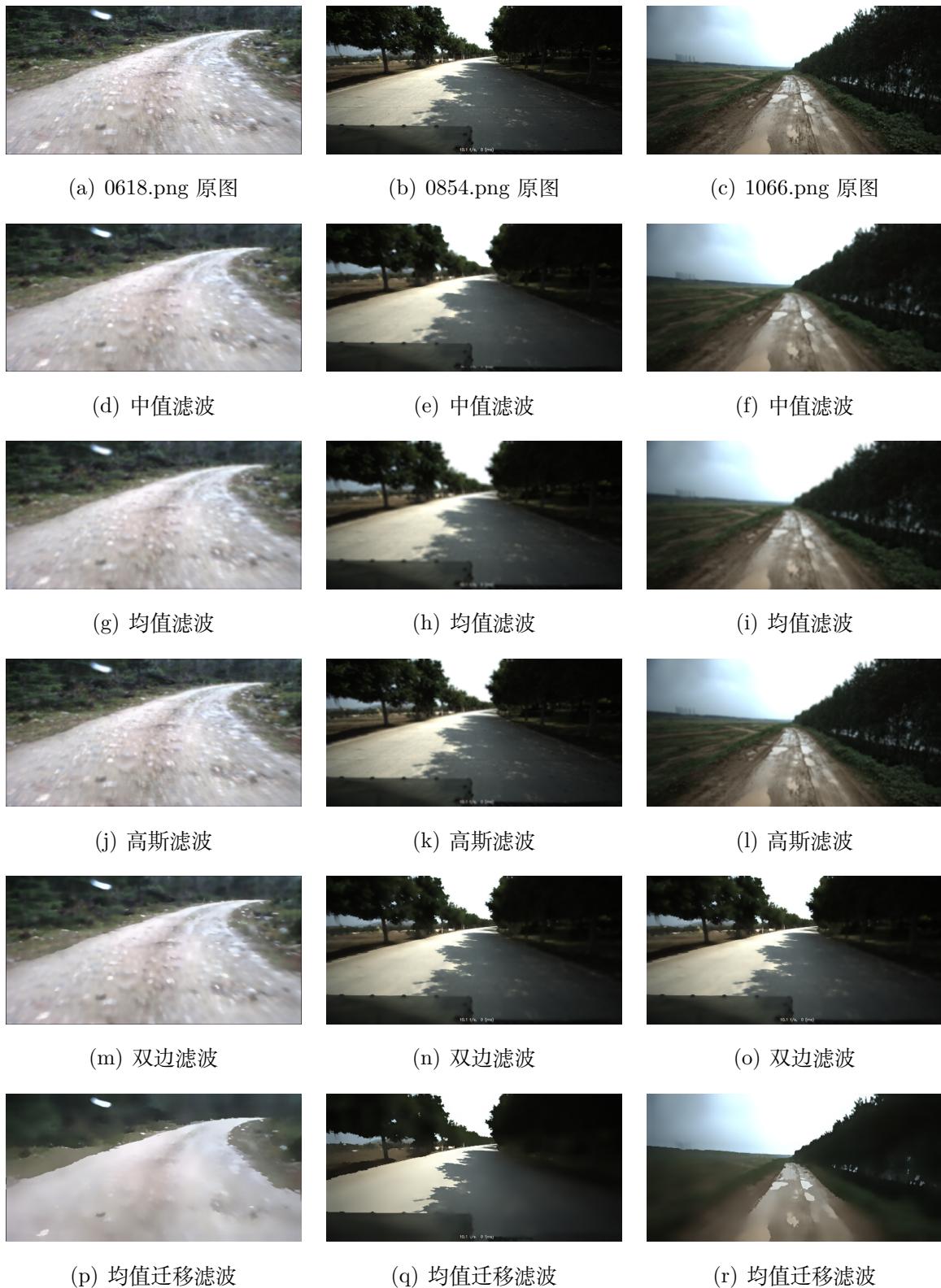


图 2: 原始图像滤波结果

表 2: 特征提取实验参数设置

方法	参数设置
Original RGB	None
PCA	n_components=1,2
KPCA	n_components=1,2 kernel=rbf, poly, sigmoid, cosine
LDA	n_components=1
Dictionary Learning	n_components=1,2

4、流形学习方法：评估特征提取效果

在流形学习中，我们采用 t-SNE, ISOMAP, lle 三种方法进行可视化特征提取效果的可视化展示，由于 sklearn 是在 CPU 条件下运行，运算速度特别慢，因此对于这三种方法，我们只针对训练集数据的特征提取效果进行可视化展示。

流形学习库的调研通过查阅资料，我们发现关于如何高效使用 t-SNE 的网站([How to Use t-SNE Effectively](#))，在这个网站中，可以探究不同样本分布情况下，不同参数对于 t-SNE 的效果展现。

在后续的调研中，我们发现前人在 t-SNE 的加速上做了很多工作，我们在 github 上找到了[tsne-cuda 库](#)，该库对于 t-SNE 进行了显著的加速，加速效果如图 3 所示

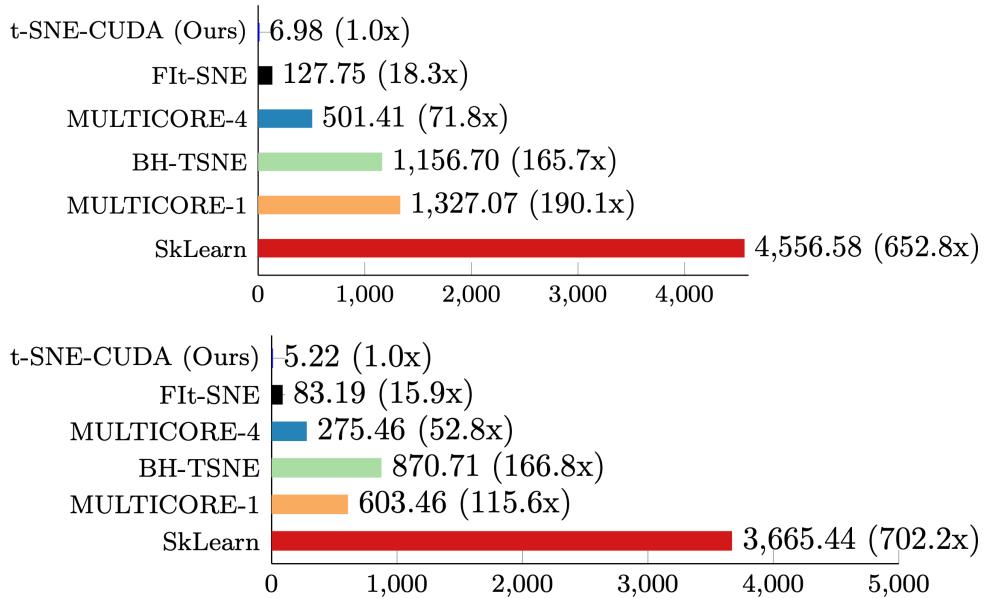


图 3: tsnecuda 在 MINIST 和 CIFAR 数据集上与其他库运算速度的对比

原本以为，可以采用 tsnecuda 进行加速，但是 tsnecuda 存在很多问题，其中之一就是不能用 pca 作为初始化。通过本地的实验对比，我们发现其效果远不如 sklearn。从图 4 中，我们不难发现，sklearn 上的效果明显要优于 tsnecuda。

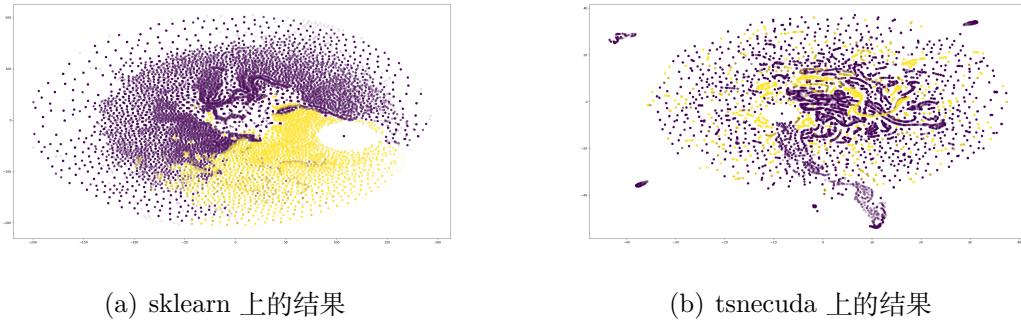


图 4: sklearn 和 tsnecuda 结果对比

从图 4 中，我们发现，sklearn 上能够明显分出紫色和黄色两个类别，但是 tsnecuda 上，黄色和紫色混杂在了一起，区分的效果明显差于前者。

抛弃了 tsnecuda 之后，通过调研，我们又发现了 cuML 库，cuML 是一套实现机器学习算法和数学原语函数的库，与其他 RAPIDS 项目共享兼容的 api。cuML 使数据科学家、研究人员和软件工程师能够在 gpu 上运行传统的表格式 ML 任务，而无需深入 CUDA 编程的细节。对于大型数据集，这些基于 gpu 的实现可以比 CPU 实现快 10-50 倍。

使用 sklearn 库完成 t-SNE, ISOMAP, lle 基于前面所讲述的，我们先利用 sklearn 库，采用 t-SNE, ISOMAP, lle 三种方法进行可视化特征提取效果的可视化展示。我们现在总共具有三个维度：滤波方式、特征提取方式、流形学习方式，不易直接展示。首先我们先看滤波方式和流形学习之间的关系。

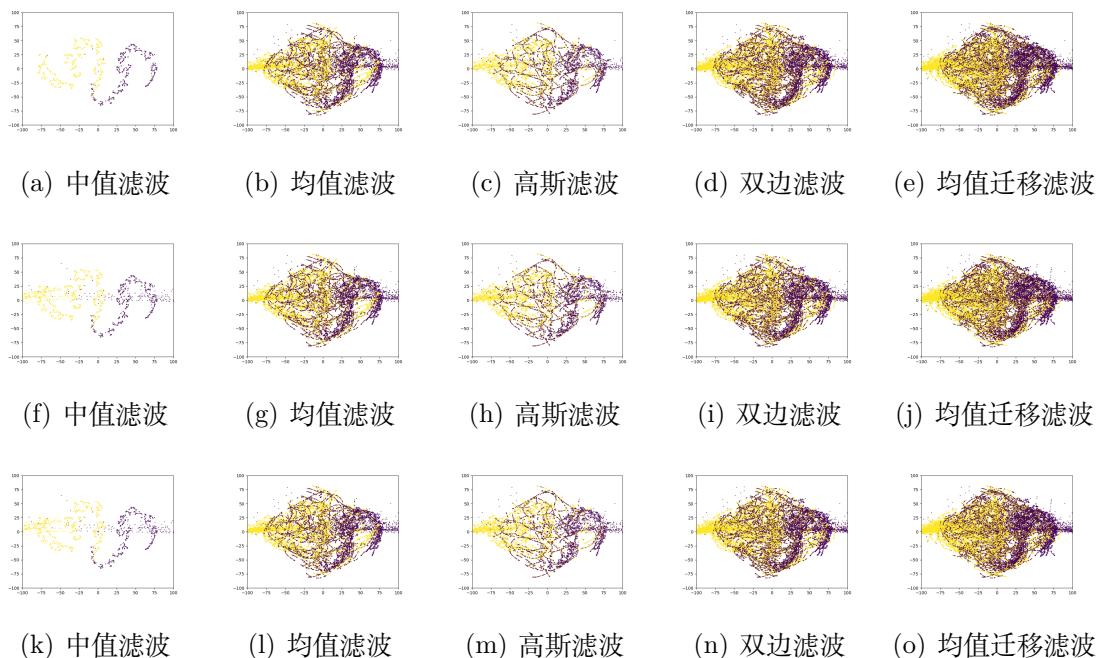


图 5: 0618.png 平滤波方式与流形学习之间的关系

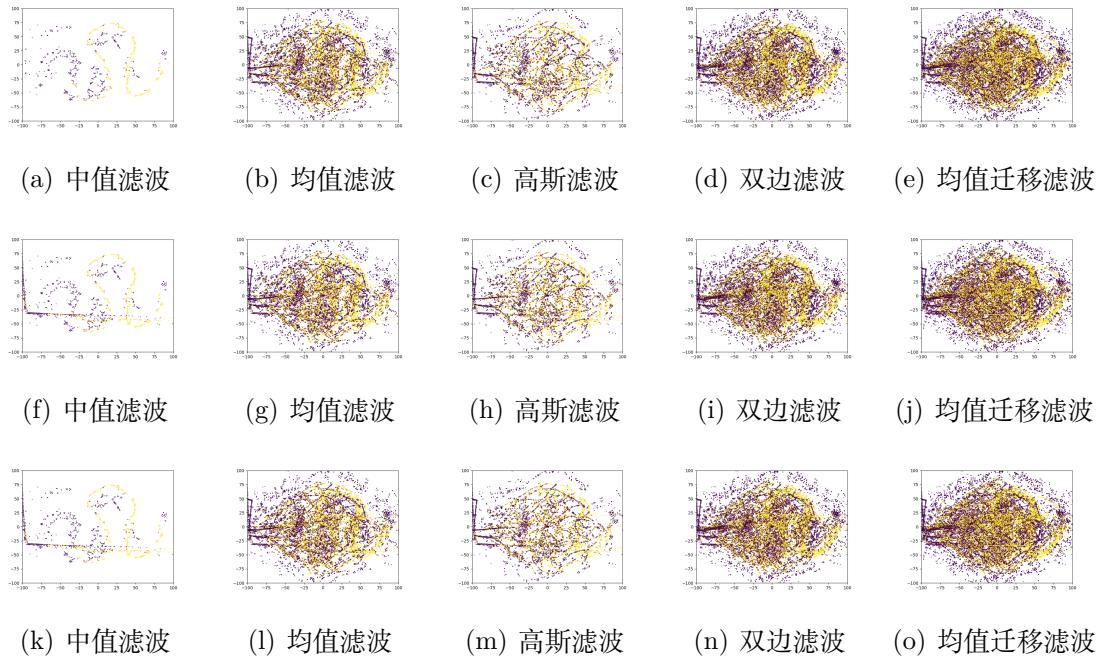


图 6: 0854.png 滤波方式与流形学习之间的关系

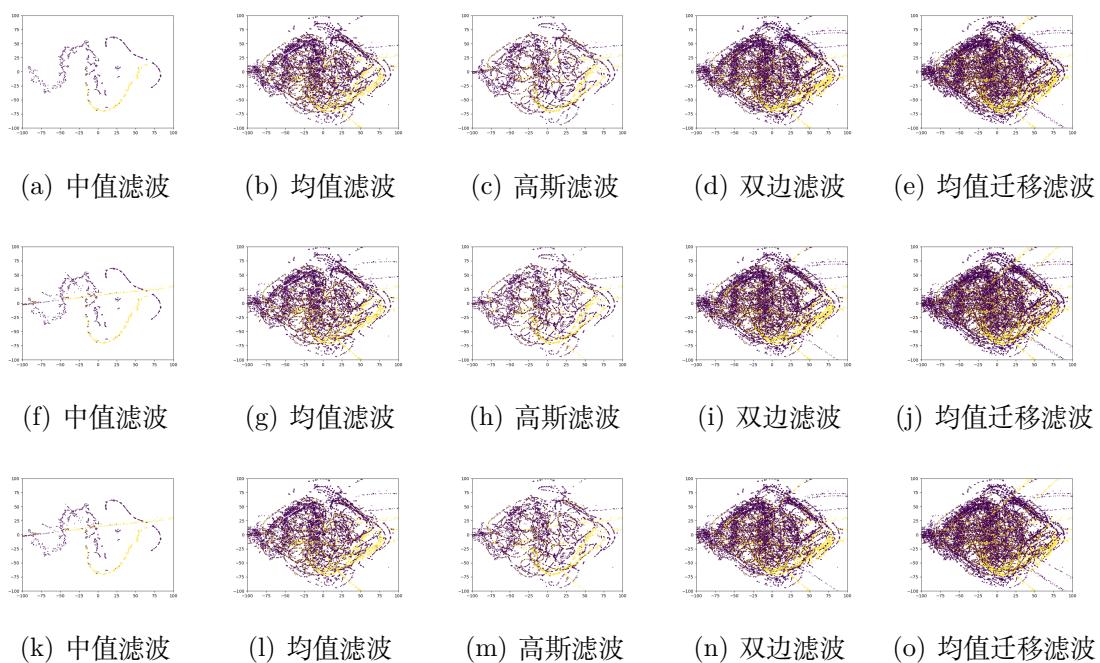


图 7: 1066.png 滤波方式与流形学习之间的关系

图 5-7展示了 5 种滤波方式与流形学习之间的关系,从上至下分别是 t-SNE, ISOMAP, lle 三种方法, 我们发现滤波方式对于流形学习的结果, 其实也就是特征提取的结果, 有一定的影响, 中值滤波的效果要优于其他的方式, 但是也有可能是流形学习没有做到很好的分类效果。横向来看, t-SNE, ISOMAP, lle 三种方法差别不大, 后两者的分类效果要稍微差于 t-SNE, 但是总体来讲, 差别不大。

下面我们再比较特征提取与流形学习之间的关系，这里我们选取中值滤波方式。

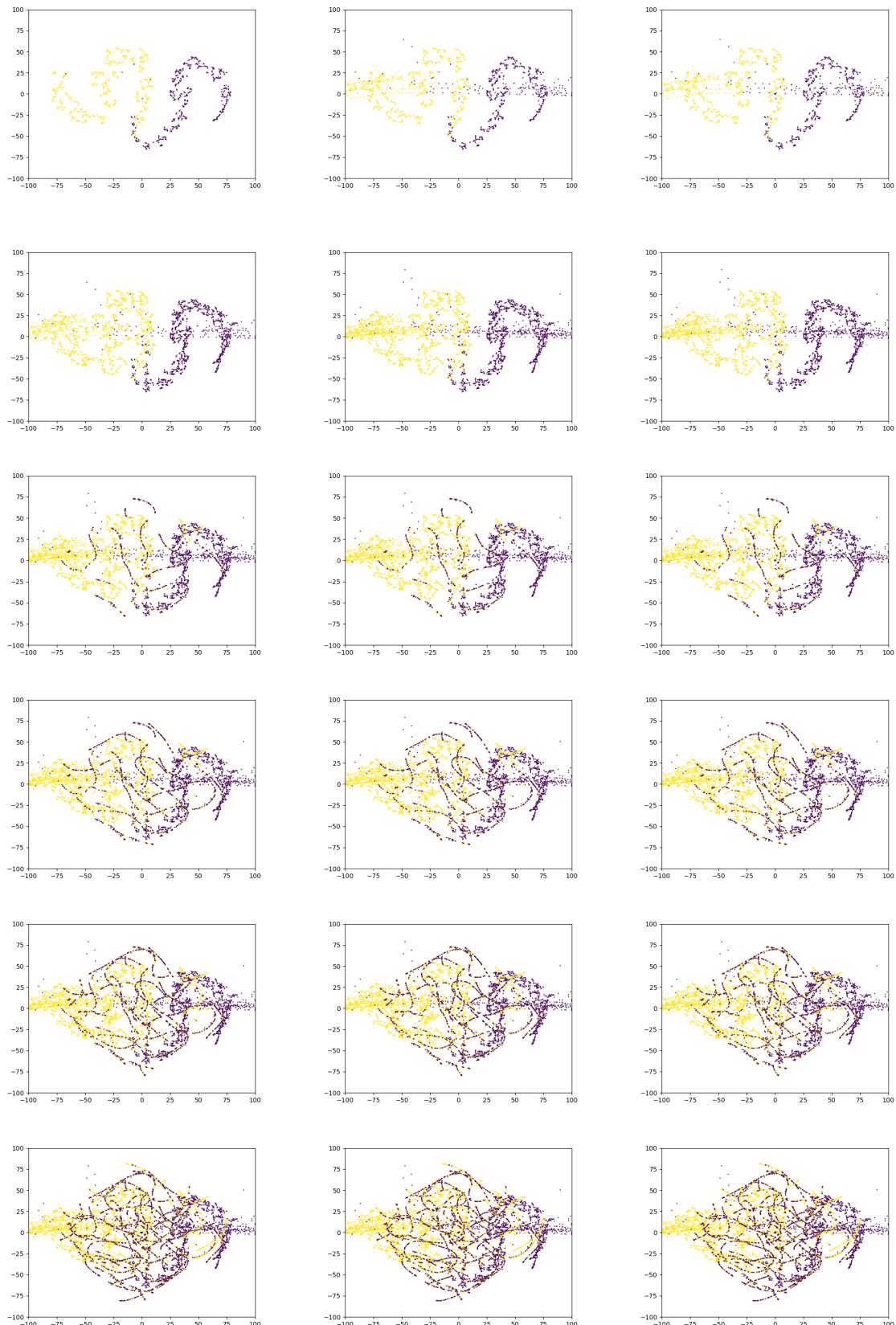


图 8: 0618.png 特征提取与流形学习之间的关系

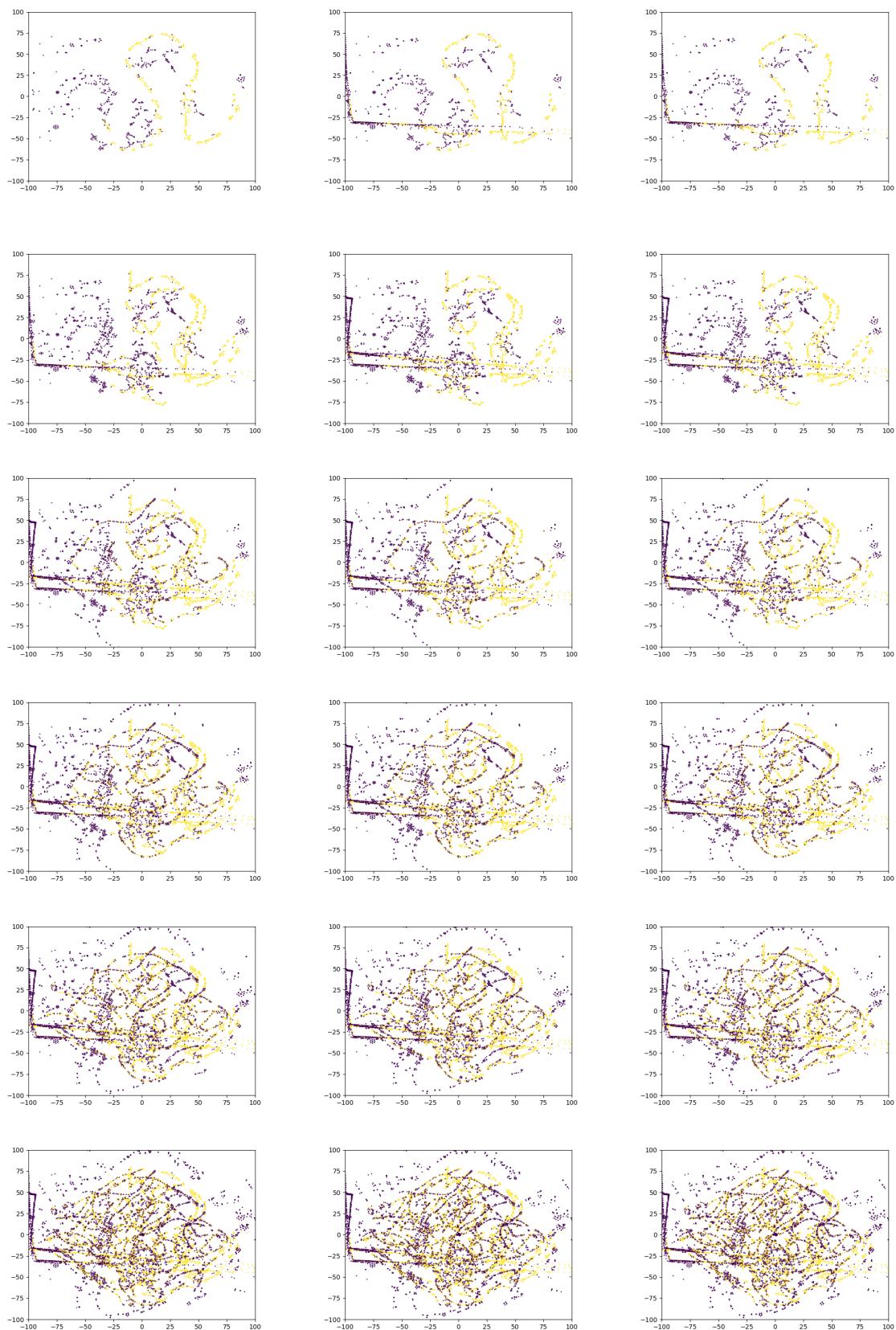


图 9: 0854.png 特征提取与流形学习之间的关系

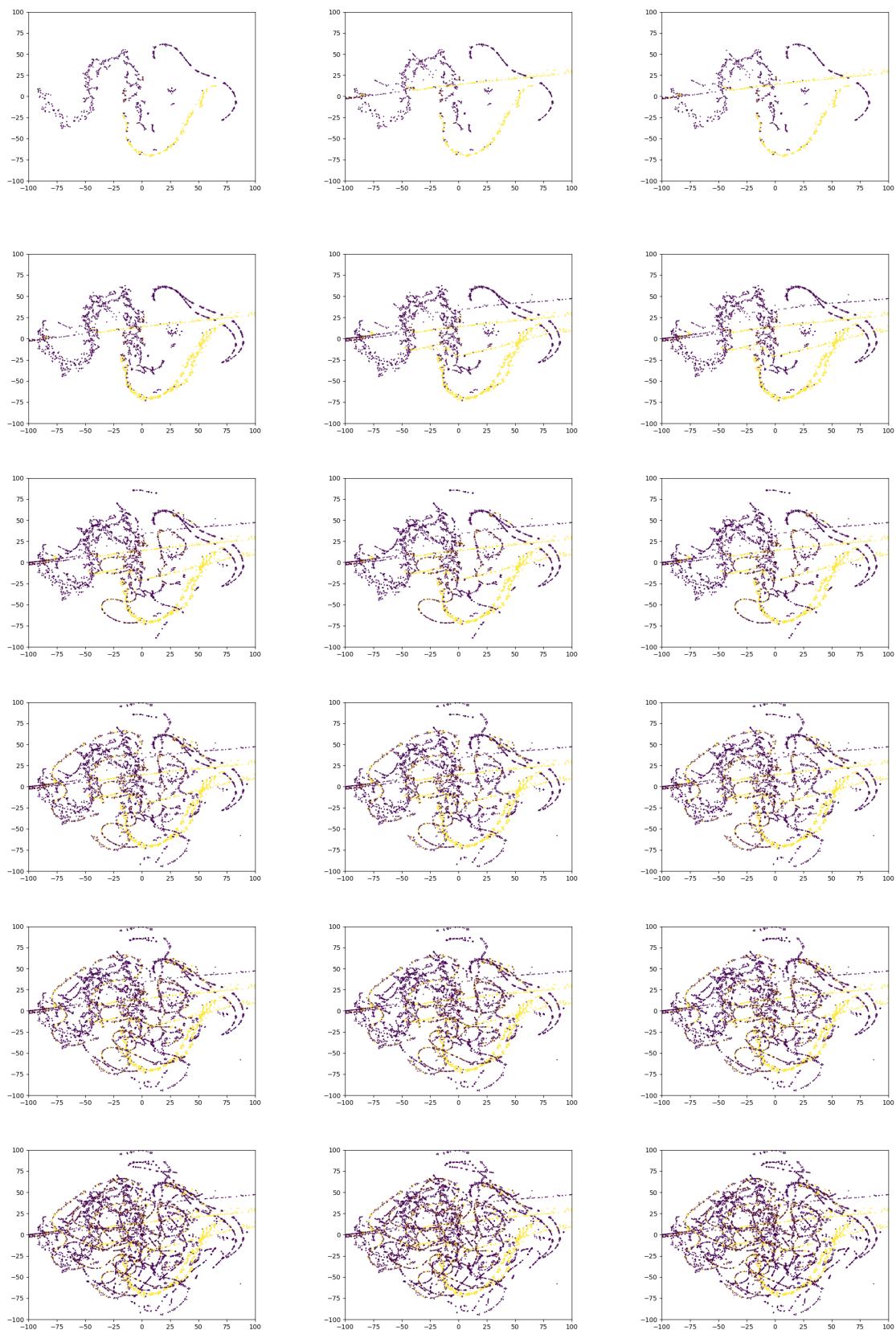


图 10: 1066.png 特征提取与流形学习之间的关系

图 8-10展示了不同图片，在不同特征提取条件下的流形学习图，对于每张图而言，从上至下分别为原始特征、PCA 特征、KPCA (rbf) 特征、KPCA (poly) 特征、KPCA (sigmoid) 特征、字典学习特征，从左至右，则为 t-SNE, ISOMAP, lle 流形学习方法。

图 8展示了 0618.png 图像的特征提取与流形学习之间的关系，从中我们不难发现，PCA 特征的效果要明显优于其他 5 种特征，t-SNE 做流形学习的结果图中噪声更少，效果更佳。对于图 9和图 10有相似的结果。

使用 cuML 库完成 t-SNE 前面提到 cuML 库可以显著加速 t-SNE 的计算速率，在上一部分的 sklearn 仅仅针对了训练集进行展示，这里我们利用加速的 t-SNE 可以很快计算出整个图片所有样本的流形学习图。由于效果比较相似，这里我们仅展示 0618.png 的结果。

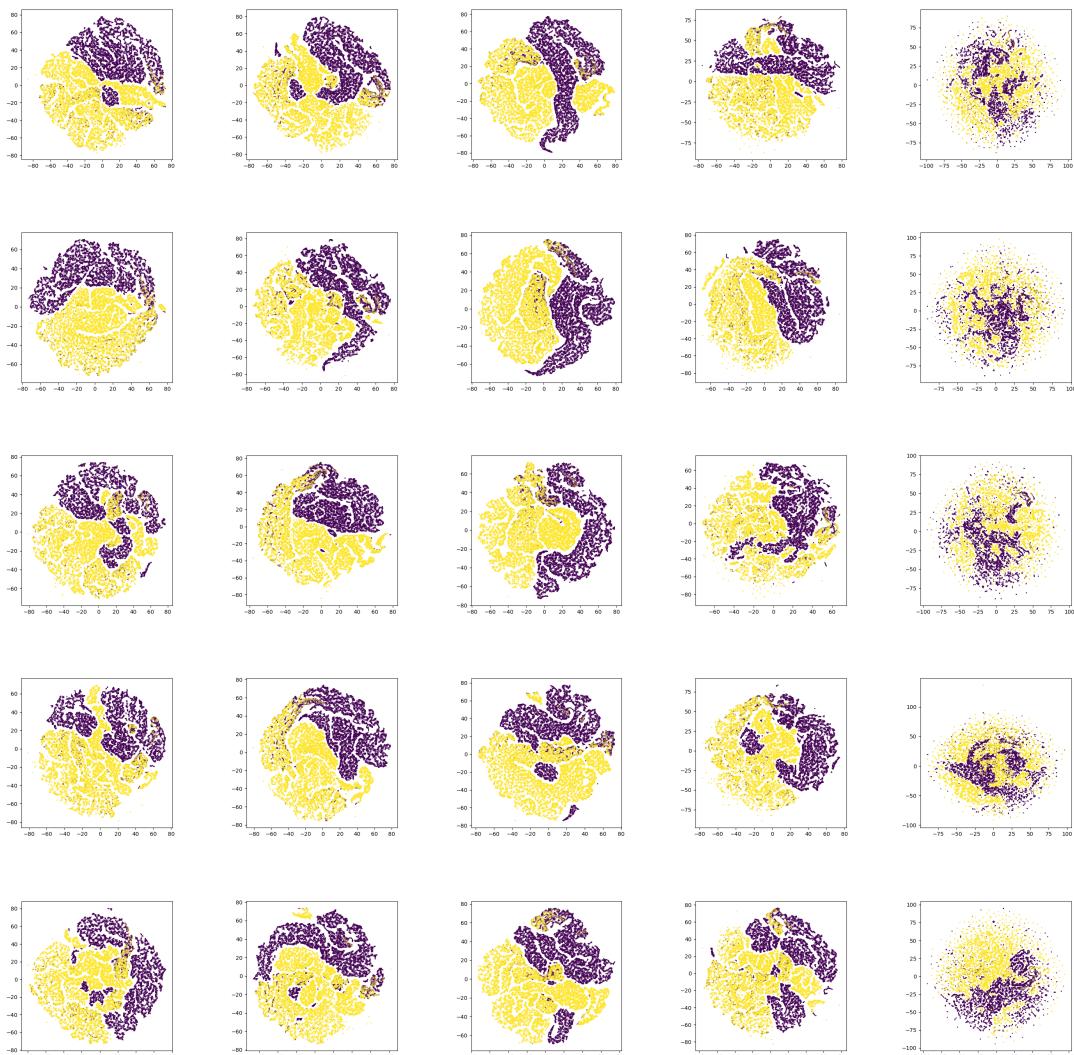


图 11: 利用 cuML 计算 0618.png 特征提取后的 t-SNE

图 11展示了利用 cuML 库使用 t-SNE 对 0618.png 图片进行流形学习的结果，从左至右，为不同的滤波方式：中值滤波、均值滤波、高斯滤波、双边滤波、均值迁移滤波，从上至下，分别为原始特征，PCA 特征、KPCA (rbf) 特征、KPCA (poly) 特征、字典学习特征。从中我们不难发现，cuML 做 t-SNE 的效果还是比较好的，在全样本纳入学习的情况下，能够很好的将道路和非道路特征分开。

5、分类算法：道路分割

前面我们提到，道路分割算法，我们采用支持向量机（Support Vector Machine, SVM），随机森林（Random Forest），K 最近邻算法（K-Nearest Neighbors, KNN）进行道路分割的实验。实验的维度主要有如下三维：滤波方式、特征提取方式、分类方式。

表 3: 0618.png 的准确率结果

特征提取 →	RGB			PCA			dictionary learning		
分类器 →	SVM	RF	KNN	SVM	RF	KNN	SVM	RF	KNN
中值滤波	97.43	98.06	97.47	97.20	94.66	97.19	97.13	97.46	96.37
均值滤波	97.08	97.58	97.15	96.82	93.83	96.64	96.52	97.04	95.96
高斯滤波	97.01	97.53	96.93	96.73	94.39	96.49	96.53	96.87	96.05
双边滤波	97.66	98.26	97.51	97.32	94.04	97.42	96.72	97.45	96.35
均值迁移滤波	97.92	98.09	98.00	97.92	94.64	98.10	97.94	97.87	97.70

特征提取 →	KPCA(rbf)			KPCA(poly)			KPCA(cosine)		
分类器 →	SVM	RF	KNN	SVM	RF	KNN	SVM	RF	KNN
中值滤波	64.24	57.87	54.95	97.23	95.90	97.44	94.31	78.64	94.03
均值滤波	56.58	57.87	56.69	96.77	95.45	96.95	94.32	82.22	93.69
高斯滤波	57.96	57.87	57.22	96.69	95.65	96.57	95.00	83.63	94.33
双边滤波	68.04	57.87	57.00	97.34	95.64	97.38	95.98	83.25	95.66
均值迁移滤波	68.85	57.87	64.26	97.93	96.16	98.17	97.92	88.57	97.84

表 4: 0618.png 的精确率结果

特征提取 →		RGB			PCA			dictionary learning		
分类器 →		SVM	RF	KNN	SVM	RF	KNN	SVM	RF	KNN
中值滤波		96.84	97.49	97.09	96.69	91.58	96.84	96.81	96.91	97.00
均值滤波		96.29	96.93	96.83	96.23	90.39	96.91	95.68	96.53	95.84
高斯滤波		96.62	97.18	96.78	96.67	91.20	96.65	96.57	96.39	96.03
双边滤波		96.97	97.68	97.08	96.80	90.66	97.11	96.42	96.85	96.57
均值迁移滤波		97.03	97.13	97.48	97.03	91.57	97.41	97.12	96.92	97.20

特征提取 →		KPCA(rbf)			KPCA(poly)			KPCA(cosine)		
分类器 →		SVM	RF	KNN	SVM	RF	KNN	SVM	RF	KNN
中值滤波		83.67	57.87	57.73	96.77	93.47	97.19	92.37	73.04	93.94
均值滤波		63.09	57.87	58.30	96.11	92.84	96.90	92.22	76.50	93.39
高斯滤波		63.57	57.87	58.74	96.60	93.11	96.82	92.89	77.95	93.93
双边滤波		82.76	57.87	59.07	96.76	93.05	97.02	94.07	77.55	95.11
均值迁移滤波		78.55	57.87	67.01	97.06	93.90	97.46	96.76	83.52	97.20

表 5: 0618.png 的召回率结果

特征提取 →		RGB			PCA			dictionary learning		
分类器 →		SVM	RF	KNN	SVM	RF	KNN	SVM	RF	KNN
中值滤波		98.78	99.21	98.58	98.53	99.96	98.34	98.29	98.76	96.72
均值滤波		98.76	98.96	98.28	98.35	99.97	97.30	98.44	98.42	97.24
高斯滤波		98.27	98.59	97.95	97.72	99.95	97.30	97.47	98.27	97.20
双边滤波		99.05	99.36	98.66	98.63	100.00	98.47	97.97	98.81	97.14
均值迁移滤波		99.46	99.65	99.11	99.46	99.94	99.35	99.39	99.48	98.88

特征提取 →		KPCA(rbf)			KPCA(poly)			KPCA(cosine)		
分类器 →		SVM	RF	KNN	SVM	RF	KNN	SVM	RF	KNN
中值滤波		47.47	100.00	82.71	98.50	99.90	98.42	98.28	100.00	95.86
均值滤波		60.15	100.00	88.38	98.40	99.84	97.86	98.48	100.00	95.87
高斯滤波		64.07	100.00	87.61	97.73	99.87	97.27	98.93	100.00	96.42
双边滤波		56.56	100.00	83.73	98.72	99.93	98.50	99.31	100.00	97.52
均值迁移滤波		63.52	100.00	75.34	99.43	99.85	99.43	99.75	99.99	99.12

表 6: 0618.png 的 F1 值结果

特征提取 →	RGB			PCA			dictionary learning		
分类器 →	SVM	RF	KNN	SVM	RF	KNN	SVM	RF	KNN
中值滤波	97.80	98.34	97.83	97.60	95.59	97.59	97.54	97.83	96.86
均值滤波	97.51	97.93	97.55	97.28	94.94	97.11	97.04	97.46	96.54
高斯滤波	97.44	97.88	97.36	97.19	95.38	96.97	97.02	97.32	96.61
双边滤波	98.00	98.51	97.86	97.71	95.10	97.79	97.19	97.82	96.86
均值迁移滤波	98.23	98.37	98.29	98.23	95.57	98.37	98.24	98.19	98.03

特征提取 →	KPCA(rbf)			KPCA(poly)			KPCA(cosine)		
分类器 →	SVM	RF	KNN	SVM	RF	KNN	SVM	RF	KNN
中值滤波	60.57	73.32	68.00	97.63	96.58	97.80	95.23	84.42	94.89
均值滤波	61.59	73.32	70.26	97.24	96.21	97.37	95.25	86.68	94.62
高斯滤波	63.82	73.32	70.33	97.16	96.37	97.04	95.81	87.61	95.16
双边滤波	67.20	73.32	69.27	97.73	96.37	97.75	96.62	87.36	96.30
均值迁移滤波	70.24	73.32	70.93	98.23	96.78	98.44	98.23	91.01	98.15

表 3-6展示了 0618.png 图像在 5 种滤波、6 种特征提取、3 种分类器条件下的结果，仔细观察，我们发现 SVM、RF 和 KNN 三种分类方法各有优劣。在 RGB 特征、字典学习特征、KPCA (poly) 特征下，三者的分类效果相近，没有很大差别。RF 在 KPCA (rbf) 下特征下，效果要优于 SVM 和 KNN，但是在 PCA 特征和 KPCA (cosine) 特征下，RF 要差于 SVM、KNN。0854.png 和 1066.png 的分类结果表格详见附录。

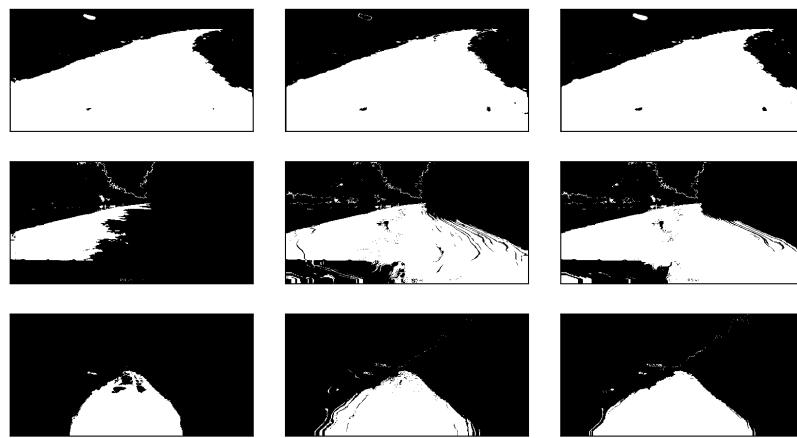


图 12: 利用 svm、rf、knn 对图像进行道路分割的结果

图 12展示了利用不同分类器，对图像进行道路分割的结果，从左至右，分别为 SVM，Random Forest，KNN 三种分类器，从中不难发现，Random Forest 的分类效果最好，其在 0618.png 图像中，能够将左上角的白点进行很好的剔除。SVM 的效果较差，尤其

是在 0854.png 图像中，只能分类出一半的图像。KNN 对于边缘有较好的效果，相较于前两种分类器边缘更加平滑。

6、数据后处理：采用形态学方法优化结果

在形态学方法进行处理的过程中，我们统一选用 kernel 为 $(10, 10)$ 大小的全为 1 的矩阵。

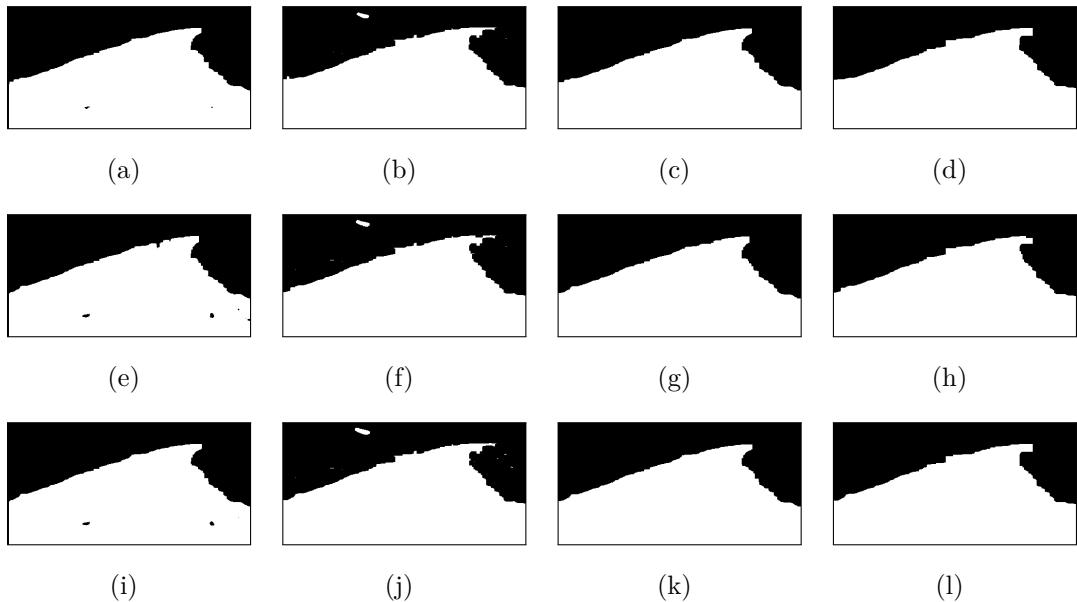


图 13: 0618.png 图像经过形态学方法优化的结果

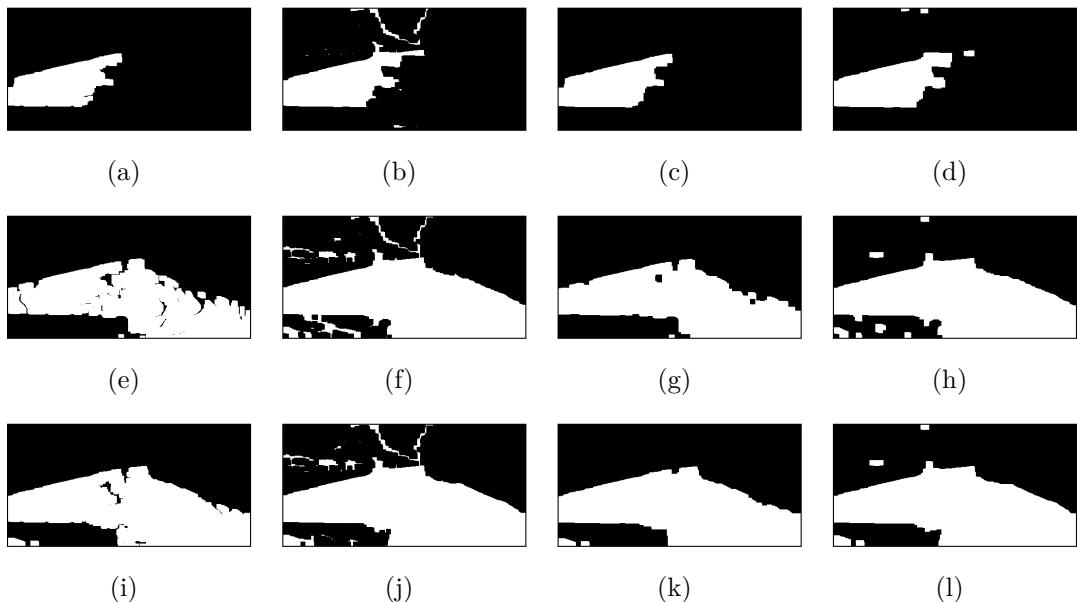


图 14: 0854.png 图像经过形态学方法优化的结果

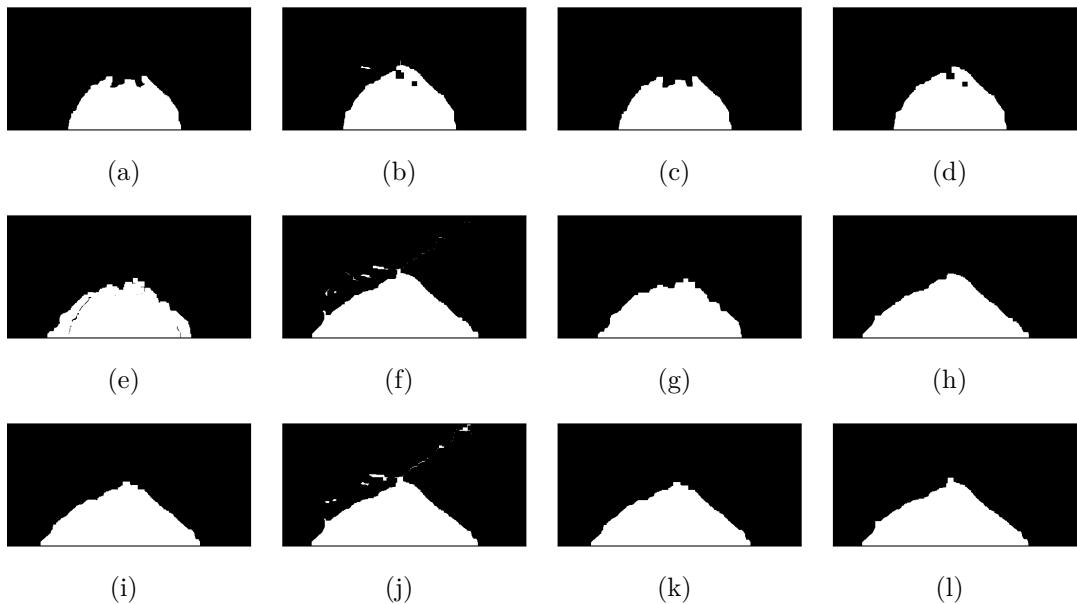


图 15: 1066.png 图像经过形态学方法优化的结果

图 13-15展示了采用形态学处理的方式，对结果进行调优，其中第一行至第三行分别表示 svm、rf、knn 的结果，从第一列至第四列分别表示开运算、闭运算、先开运算后闭运算、先闭运算后开运算。

对于 0618.png 图像的结果图 13而言，整体效果较好，使用了闭运算的结果都能够将左上角的白点给去除，使用了开运算的结果都能够补全整个道路，不至于出现破洞。

对于 0854.png 图像的结果图 14而言，采用了 knn 算法的先开运算后闭运算的结果较好，即子图14(k)，将道路进行了很好的分割。

对于 1066.png 图像的结果图 15而言，svm 的结果有点优化过渡了，使得道路小于原先的道路。而 rf 和 knn 的结果较好，能够将道路进行很好的分割。

4 流形学习各个参数的探究

前面我们谈到([How to Use t-SNE Effectively](#))网站中，可以对不同样本分布、不同 t-SNE 参数设置进行探究。

我们仅仅对简单的参数和分布进行了初步探索，由于时间有限，没有进行更加详尽的探究。

5 实验心得与体会

本次实验主要完成了两个方面的内容，利用 KMeans 和 DBSCAN 对道路进行分割。

在使用 KMeans 进行分割时，我们设置了不同的聚类数目，从 2 到 9，聚类的效果在 3 和 4 的时候达到比较好的状态。

在使用 DBSCAN 进行分割时，我们对不同的 eps (0.1, 0.5, 1.0)，不同的 min_samples (1, 2, ..., 9) 进行实验，发现当 eps 为 0.1 和 0.5 时，在 min_samples 为 5 左右的效果比较好，而当 eps 为 1.0 时，整体分类效果较差，几乎不能看见道路。

6 存在的主要问题和建议

- 使用 DBSCAN 求解时，运算速度明显慢于 KMeans。
 - 在 KMeans 实验中，可以采用选取若干道路的样本点，计算其中心，然后与聚类中心进行距离的判别，设置阈值，实现对于道路判别；由于我们道路拍摄视角较为固定，可以设置一个权重图，其为一个等腰三角形，位于图像正中间，尝试采用加权的 KMeans 聚类方法进行道路分割并实现对类别的判别。
 - 在 DBSCAN 实验中，使用了不同的 eps 和 min_samples 进行实验，但是没有很好的对结果进行解释。

7 附录

附录 A：图表

附录 B：核心代码展示

```
class ImageProcess:

    def __init__(self, image):
        self.image = image
        self.img = None

    def median_blur(self, ksize=5):
        self.img = cv2.medianBlur(self.image, ksize)

    def gaussian_blur(self, ksize=(5, 5)):
        self.img = cv2.GaussianBlur(self.image, ksize, 0)

    def average_blur(self, ksize=(5, 5)):
        self.img = cv2.blur(self.image, ksize)

    def bilateral_filter(self, d=9, sigmaColor=75, sigmaSpace=75):
        self.img = cv2.bilateralFilter(self.image, d, sigmaColor,
```

```

sigmaSpace)

def mean_shift_filter(self, sp=10, sr=50):
    self.img = cv2.pyrMeanShiftFiltering(self.image, sp, sr)

def opening(self, filename, ksize=(10, 10)):
    kernel = np.ones(ksize, np.uint8)
    self.img = cv2.morphologyEx(self.image, cv2.MORPH_OPEN,
        kernel)
    cv2.imwrite(filename, self.img)

def closing(self, filename, ksize=(10, 10)):
    kernel = np.ones(ksize, np.uint8)
    self.img = cv2.morphologyEx(self.image, cv2.MORPH_CLOSE,
        kernel)
    cv2.imwrite(filename, self.img)

def opening_closing(self, filename, ksize=(10, 10)):
    kernel = np.ones(ksize, np.uint8)
    self.img = cv2.morphologyEx(self.image, cv2.MORPH_OPEN,
        kernel)
    kernel = np.ones(ksize, np.uint8)
    self.img = cv2.morphologyEx(self.img, cv2.MORPH_CLOSE,
        kernel)
    cv2.imwrite(filename, self.img)

def closing_opening(self, filename, ksize=(10, 10)):
    kernel = np.ones(ksize, np.uint8)
    self.img = cv2.morphologyEx(self.image, cv2.MORPH_CLOSE,
        kernel)
    kernel = np.ones(ksize, np.uint8)
    self.img = cv2.morphologyEx(self.img, cv2.MORPH_OPEN,
        kernel)
    cv2.imwrite(filename, self.img)

```

```

class FeatureExtractor:
    def __init__(self, train_data, train_label, data):

```

```

        self.td = train_data
        self.d = data
        self.train_data = None
        self.data = None
        self.train_label = train_label

    def original_feature(self):
        self.train_data = self.td
        self.data = self.d

    def pca_feature(self, n_components=2):
        pca = PCA(n_components=n_components)
        pca.fit(self.td)
        self.train_data = pca.transform(self.td)
        self.data = pca.transform(self.d)

    def kpca_feature(self, n_components=1, kernel='rbf'):
        kpca = KernelPCA(n_components=n_components, kernel=kernel)
        kpca.fit(self.td)
        self.train_data = kpca.transform(self.td)
        self.data = kpca.transform(self.d)

    def lda_feature(self, n_components=1):
        lda = LinearDiscriminantAnalysis(n_components=n_components)
        lda.fit(self.td, self.train_label)
        self.train_data = lda.transform(self.td)
        self.data = lda.transform(self.d)

    def dictionary_learning_feature(self, n_components=1):
        dl = MiniBatchDictionaryLearning(n_components=n_components)
        dl.fit(self.td)
        self.train_data = dl.transform(self.td)
        self.data = dl.transform(self.d)

```

```

class ManifoldPlot:
    def __init__(self, train_data, train_label, data):
        self.train_data = train_data

```

```

        self.train_label = train_label
        self.data = data

    def tsne_plot(self, filename, n_components=2, perplexity=10):
        tsne = TSNE(n_components=n_components,
                     perplexity=perplexity)
        tsne_train_data = tsne.fit_transform(self.train_data)
        plt.scatter(tsne_train_data[:, 0], tsne_train_data[:, 1],
                    c=self.train_label, alpha=0.8, s=0.8)
        plt.xlim(-100, 100)
        plt.ylim(-100, 100)
        plt.savefig(filename)

    def isomap_plot(self, filename, n_components=2, n_neighbors=5):
        isomap = Isomap(n_components=n_components,
                        n_neighbors=n_neighbors)
        isomap_train_data = isomap.fit_transform(self.train_data)
        plt.scatter(isomap_train_data[:, 0], isomap_train_data[:, 1],
                    c=self.train_label, alpha=0.8, s=0.8)
        plt.xlim(-100, 100)
        plt.ylim(-100, 100)
        plt.savefig(filename)
        # plt.show()

    def lle_plot(self, filename, n_components=2, n_neighbors=5):
        lle = LocallyLinearEmbedding(n_components=n_components,
                                     n_neighbors=n_neighbors, eigen_solver='dense')
        lle_train_data = lle.fit_transform(self.train_data)
        plt.scatter(lle_train_data[:, 0], lle_train_data[:, 1],
                    c=self.train_label, alpha=0.8, s=0.8)
        plt.xlim(-100, 100)
        plt.ylim(-100, 100)
        plt.savefig(filename)

```

```

def auto_sample(data_dir, img_name, output_dir,
                sample_interval_list):
    for sample_interval in [sample_interval_list]:

```

```

print(f'sample_interval: {sample_interval}')
img_label = np.load(os.path.join(data_dir, img_name[:-4] +
                                  "_label.npy"))
if not os.path.exists(output_dir):
    os.mkdir(output_dir)

output_dir = os.path.join(output_dir, img_name[:-4] +
                           "_{}".format(sample_interval))
img = cv2.imread(os.path.join(data_dir, img_name))
if os.path.exists(output_dir):
    shutil.rmtree(output_dir)
os.mkdir(output_dir)

shutil.copy(os.path.join(data_dir, img_name), output_dir +
            "/" + img_name)
img_h, img_w, img_c = img.shape
sample_img = img.copy()

positive_coordinates = []
negative_coordinates = []
for i in range(img_h):
    for j in range(img_w):
        if i % sample_interval == 0 and j % sample_interval
           == 0:
            if img_label[i, j] == 1:
                positive_coordinates.append((i, j))
                cv2.circle(sample_img, (j, i), 1, (0, 0,
                                                   255), thickness=5)
            else:
                negative_coordinates.append((i, j))
                cv2.circle(sample_img, (j, i), 1, (255, 0,
                                                   0), thickness=5)

cv2.destroyAllWindows()
print("positive_sample", positive_coordinates)
print("negative_sample", negative_coordinates)
sample_indices = []
train_label = []
for coor in positive_coordinates:

```

```

        sample_indices.append(coor[0]*img_w + coor[1])
        train_label.append(1)
    for coor in negative_coordinates:
        sample_indices.append(coor[0]*img_w + coor[1])
        # train_label.append(-1)
        train_label.append(0)
    cv2.imwrite(
        output_dir + "/" + img_name[:-4] + "_sample" +
        img_name[-4:], sample_img)
    data = np.array(img.copy(), dtype=int).reshape(img_h *
        img_w, img_c)
    # label = np.array(label_img, dtype=int).reshape(img_h *
        img_w)
    train_data = data[sample_indices]
    train_label = np.array(train_label)
    return train_data, train_label

#
采样函数，与auto_sample函数采样部分功能相同，但是不需要读取mask，不需要保存图
def sample(data_dir, img_name, output_dir_filter, filter_img_name,
sample_interval):
    img_label = np.load(os.path.join(data_dir, img_name[:-4] +
        "_label.npy"))
    label = img_label.reshape(-1)
    img = cv2.imread(os.path.join(output_dir_filter,
        filter_img_name))
    img_h, img_w, img_c = img.shape
    sample_img = img.copy()

    positive_coordinates = []
    negative_coordinates = []
    for i in range(img_h):
        for j in range(img_w):
            if i % sample_interval == 0 and j % sample_interval == 0:
                if img_label[i, j] == 1:
                    positive_coordinates.append((i, j))
                else:

```

```

                negative_coordinates.append((i, j))

print("positive_sample", positive_coordinates)
print("negative_sample", negative_coordinates)
sample_indices = []
train_label = []
for coor in positive_coordinates:
    sample_indices.append(coor[0]*img_w + coor[1])
    train_label.append(1)
for coor in negative_coordinates:
    sample_indices.append(coor[0]*img_w + coor[1])
    train_label.append(0)
data = np.array(img.copy(), dtype=int).reshape(img_h * img_w,
                                              img_c)
# label = np.array(label_img, dtype=int).reshape(img_h * img_w)
train_data = data[sample_indices]
train_label = np.array(train_label)
return train_data, train_label, data, label

def evaluator(img, img_mask):
    #
        img_mask为图像真值， img为预测的二值图像，计算准确率，精确率，召回率，F1值
    img = img.astype(np.uint8)
    img_mask = img_mask.astype(np.uint8)
    img_h, img_w = img.shape
    img = img.reshape(-1)
    img_mask = img_mask.reshape(-1)
    TP = 0
    TN = 0
    FP = 0
    FN = 0
    for i in range(img_h*img_w):
        if img_mask[i] == 255:
            if img[i] == 255:
                TP += 1
            else:
                FN += 1
        else:
            if img[i] == 255:

```

```
    FP += 1
else:
    TN += 1
#
如果 TP+FP=0 , 说明预测的图像中没有白色像素 , 此时精确率为1 , 召回率为0
if TP+FP == 0:
    precision = 1
    recall = 0
else:
    precision = TP/(TP+FP)
    recall = TP/(TP+FN)
F1 = 2*precision*recall/(precision+recall)
accuracy = (TP+TN)/(TP+TN+FP+FN)
return {"accuracy": accuracy, "precision": precision, "recall": recall, "F1": F1}
```