花卉识别

模式识别大作业

2021-6-21

清华大学自动化系 85

目录

1. 问题分析

本次任务以花卉识别为题，借用机器学习方法，完成降维可视化、聚类分析、识别分类等任务。

降维可视化是在以 PCA、tSNE 等方法，将花卉数据进行降维，得到降维后的二维图像，这样便可以将其在平面上展示出来。**聚类分析**是利用合适的聚类方法，对花卉数据进行聚类，将聚类之后的结果与标签进行对应，讨论不同聚类方法的结果。使用**非深度学习**和**深度学习**的方法，对已经标注完成但是可能含有误标数据的花卉样本进行**分类**，得到最终的花卉识别准确率。

2. 问题求解

2.1. 数据预处理

2.1.1. 预处理流程

给定的数据集为 jpg 格式的数据，由于读取 jpg 格式的图片比较慢，所以将 jpg 格式存储在 numpy 的多维数组中，将提取出的多维数组存储在磁盘中，并且在下一次读取数据时，先判断是否存在已经打包好的多维数组文件。如果存在，那么直接读取多维数组的数据；如果不存在，那么读取源文件，并将数据存到多维数组中，再存到磁盘里，供下次使用。

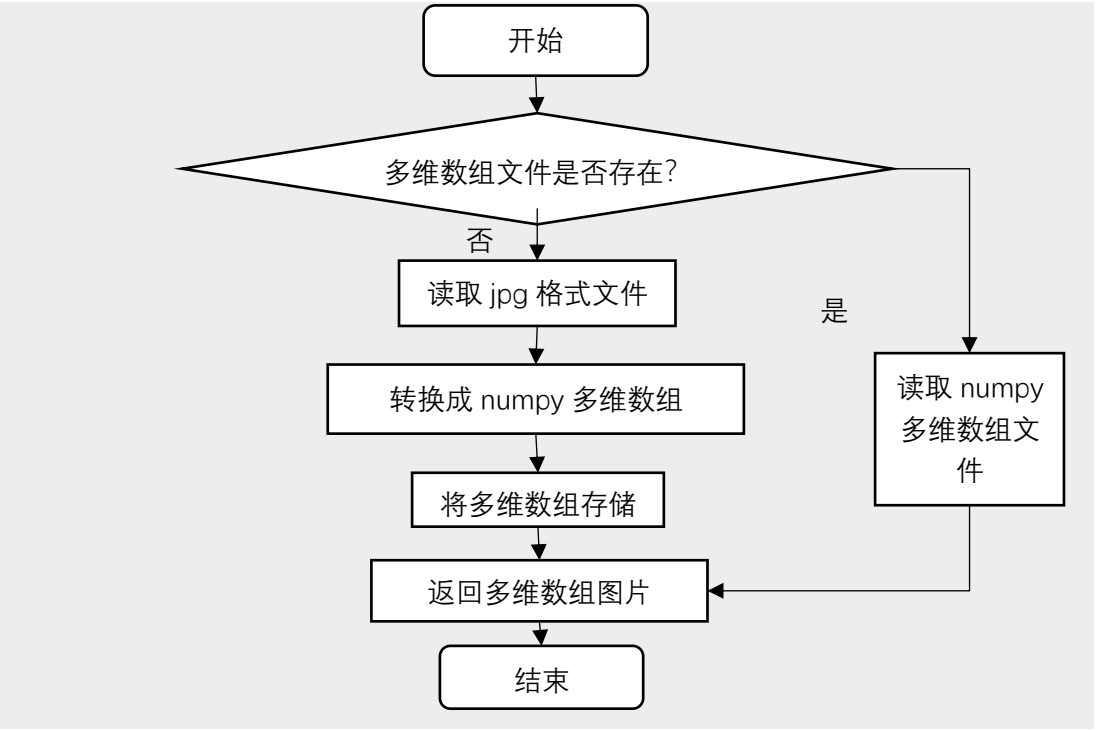


图 1：数据获取流程

2.1.2. 预处理实现

按照上面的处理流程，获取处理后的图片数组文件。为了后续方便，最终得到的图片的格式为 3*80*80 的图片。每一个模块实现过程如下：

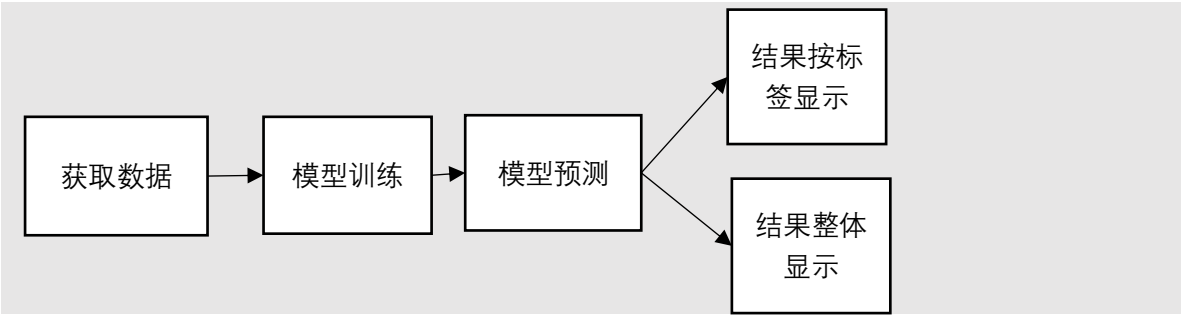
- **读取图片文件：** os.listdir 获取图片所在的文件夹下的所有图片的名称，得到图片的总个数 N。利用 matplotlib 提供的 imread 函数读取 jpg 格式的文件。
- **转换到 3*80*80 的格式：** 直接读出的 jpg 格式的格式为 80*80*3 的数组，利用 numpy 的 transpose 函数，转换各个坐标轴，即$APicture = APicture.transpose(2,0,1)$得到 3*80*80 的数据。
- **得到所有的图片数据和标签数据：**申请 N*3*80*80 的数组 Pictures 和 N 维向量 labels，每读取一个图片，将当前图片赋值到 Pictures 中，同时将文件名的第一个字母转换成数字，即为对应的标签，将其赋值到 labels 向量中。
- **保存读取的文件：** 利用 np.save 将处理得到的数据保存。
- **读取已经保存后文件：** 利用 np.load 加载已经保存的数据。

2.2. 降维可视化

降维可视化是将高维的图片数据映射到二维和三维空间中，以图像的方式观察各个类别的分布，达到可视化的目的。

2.2.1. 降维流程分析

以上面图 1 中获取数据的流程为基础，进行可视化。由于类别比较多，考虑到可视化的显示效果，可视化结果分别进行了每个标签的显示和所有标签放到一起的显示。可视化的流程如下：



2.2.2. PCA 方法降维

2.2.2.1. PCA 算法原理

PCA 方法是一种通过线性变换，用一组正交向量表示原特征的方法。PCA 通过求取最大的投影方向来使得降维后的数据尽量保持原数据的特征，同时能够达到降维的目的。

2.2.2.2. PCA 实现过程

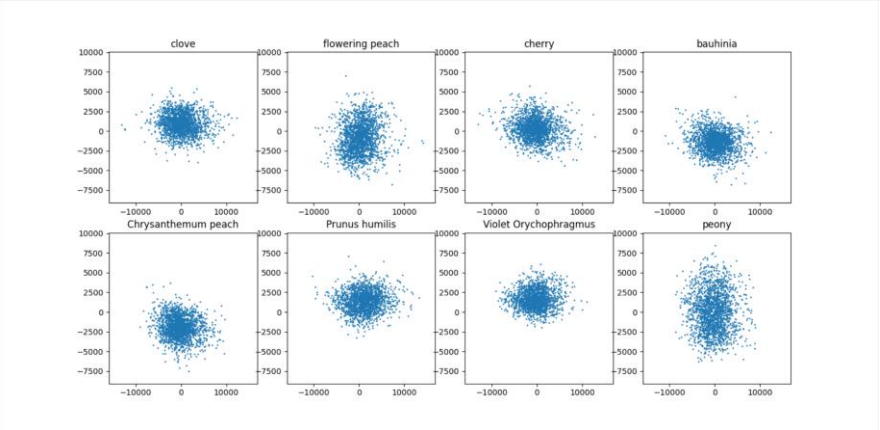
PCA 求取正交向量的方法如下：

1. 计算样本均值，将数据减去均值，得到中心化的数据结果X
2. 计算协方差矩阵$X * X^T$
3. 对协方差矩阵进行特征值分解，取最大的 k 个特征值对应的特征向量，即为最终要求得的正交向量
4. 将原数据利用 k 个特征向量构成的矩阵投影到低维空间中，即得到降维结果。

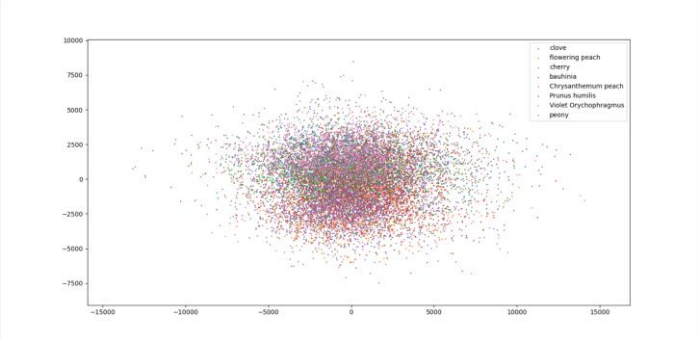
在程序实现中，我们采用 sklearn 提供的模型进行实现。

2.2.2.3. PCA 实现结果

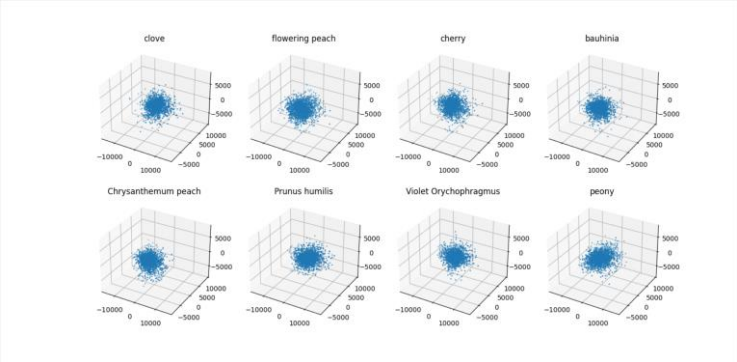
降维结果为了更加明显，将结果先按照标签进行了分开展示，之后又将所有数据的结果放到了一起进行展示，各个花卉数据对应的降维结果分别如下。从下图的结果中可以看到，不同的花卉图片在二维和三维空间上的分布有一定的区别，但是总体而言，各种花卉的结果重合程度还是比较高的。比如菊花桃（Chrysanthemum peach）和麦李（Prunus humilis）能够有一定的区分，但是麦李（Prunus humilis）和诸葛菜（Violet Orychophragmus）的重合程度比较高。



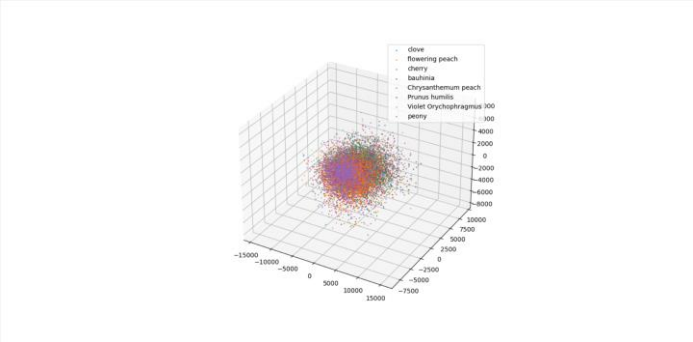
降为二维，PCA 按照标签分别展示



降为二维，PCA 将各个结果集中展示



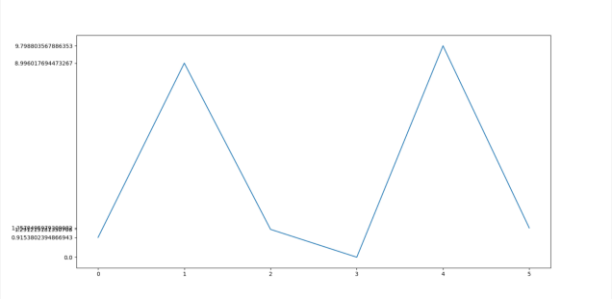
降为三维，PCA 按照标签分别展示



降为三维，PCA 将各个结果集中展示

2.2.2.4. PCA 模型性能

各个阶段的用时如下所示：



从图中给出的结果得到各个阶段的用时

读取数据	0.915s
PCA 降到两维的模型求解	8.996s
根据 PCA 模型求降维后的数据	1.291s
PCA 降到三维的模型求解	9.799s
根据 PCA 模型求降维后的数据	1.358s

从用时中可以看到，对 PCA 的模型进行求解的用时比较长，而求解降维后的数据用时比较短；由于事先采用了预处理的方式读取数据，读取数据用时比较短；同时可以看出，PCA 的用时会随着降维后的维数升高而增加，这是由于维数比较高时，需要求得更多的特征向量，同时需要更具更多的特征向量求出更多的数据，所以时间会比较长。

2.2.3. t-SNE 方法求解

2.2.3.1.t-SNE 算法原理

t-SNE 方法时利用概率分布来度量样本之间的距离，将高维空间中的欧式距离转化成条件概率密度来表示样本间的相似程度。优化的目标是高维空间中距离相近的点在低维空间中仍然相近。

2.2.3.2.t-SNE 实现过程

t-SNE 方法的实现过程如下：

1. 随机产生初始解，得到在低维空间中的映射样本Y
2. 通过初始解Y得到初始的概率分布Q，进而求得目标函数$C = KL(P||Q) = \sum_i \sum_j p_{ij} \log\left(\frac{p_{ij}}{q_{ij}}\right)$关于重构样本的梯度

$$\frac{dC}{dy_i} = 4\sum_j (p_{ij} - q_{ij})(y_i - y_j)\left(1 + \left||y_i - y_j|\right|^2\right)^{-1}$$

3. 通过梯度下降和动量法对结果进行更新

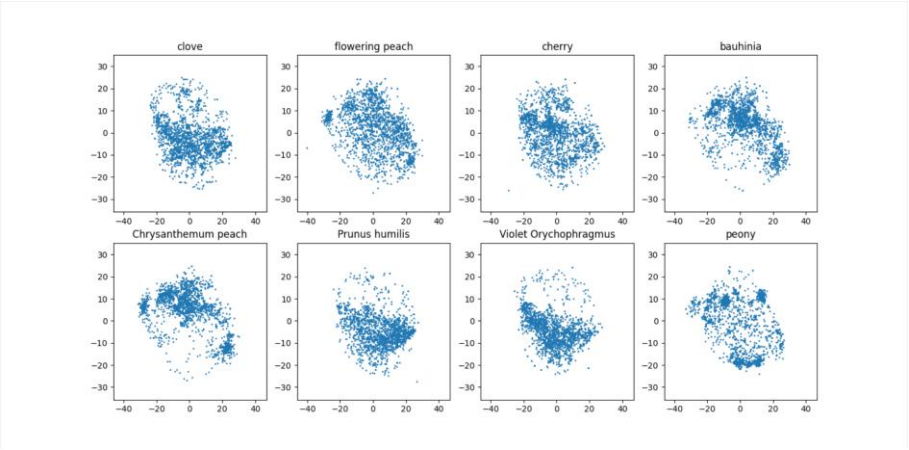
$$Y^t = Y^{t-1} + \eta\frac{dC}{dY} + \alpha(t)(Y^{t-1} - Y^{t-2})$$

4. 算法在达到指定的迭代步数时，停止迭代

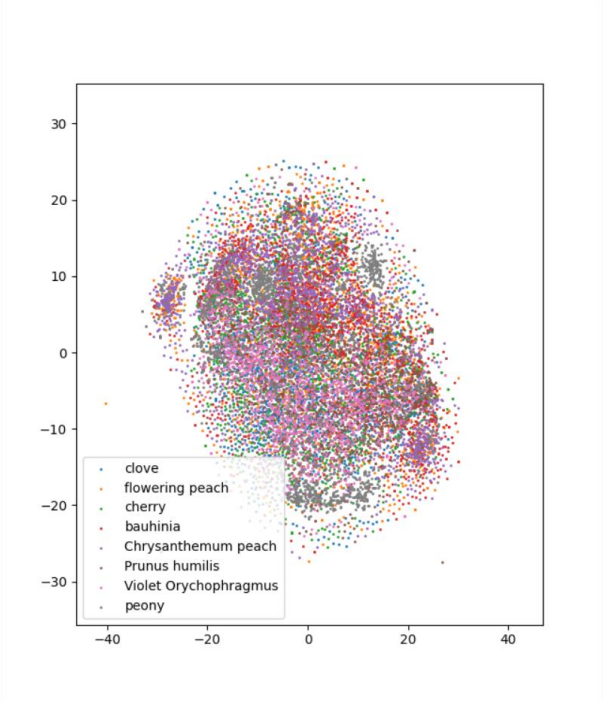
在程序实现中，我们采用 sklearn 提供的模型进行实现。

2.2.3.3.t-SNE 实现结果

降维结果为了更加明显，将结果先按照标签进行了分开展示，之后又将所有数据的结果放到了一起进行展示，各个花卉数据对应的降维结果分别如下。从下图的结果中可以看到，不同的花卉图片在二维和三维空间上的分布有一定的区别，比如丁香（clove）和菊花桃（Chrysanthemum peach）分的比较清楚，一个主要占据了上方，一个主要占据了下方。整体而言，效果优于 PCA 算法。

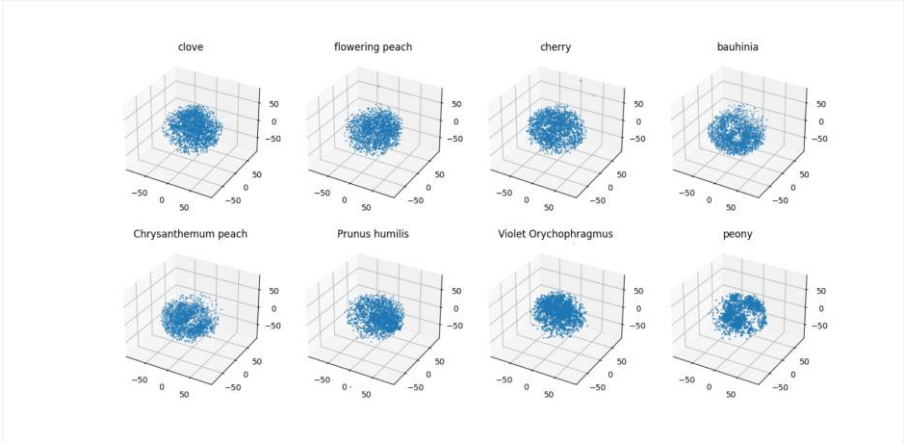


降为二维，t-SNE 按照标签分别展示

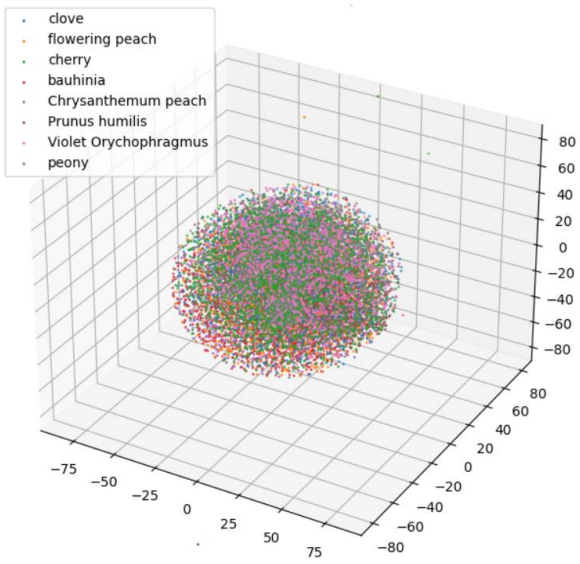


降为二维，t-SNE 将各个结果集中展示

=



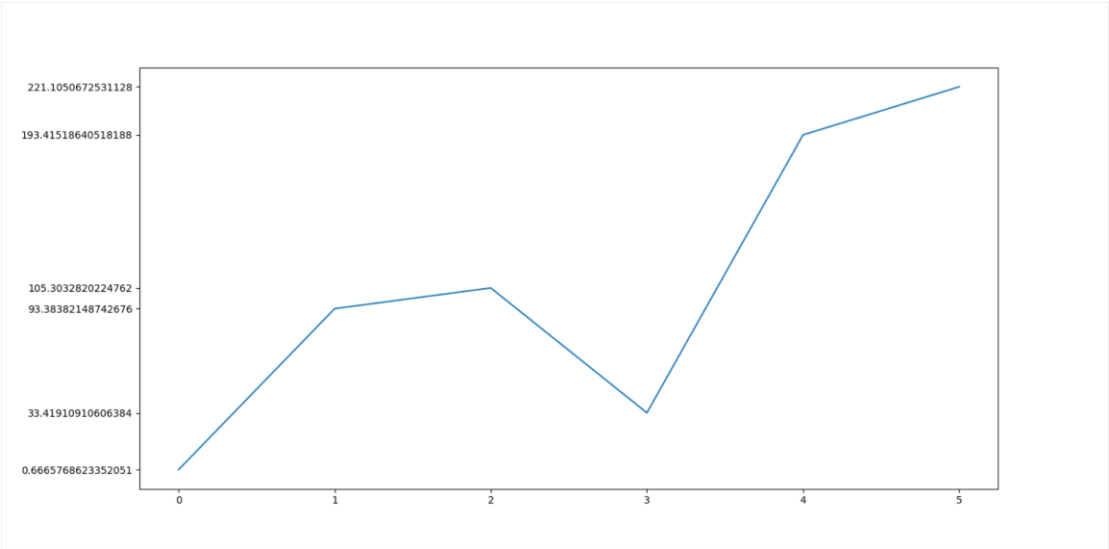
降为三维，t-SNE 按照标签分别展示



降为三维，t-SNE 将各个结果集中展示

2.2.3.4.t-SNE 模型性能

各个阶段的用时如下所示：



与上面 PCA 不同的是，这里采用了 fit 和 fit_transform，所以根据模型求解降维数据的时间需要减去 fit 的时间，从图中给出的结果得到各个阶段的用时

读取数据	0.667s
t-SNE 降到两维的模型求解	93.384s
根据 t-SNE 模型求降维后的数据	105.303-93.384=11.919s
t-SNE 降到三维的模型求解	193.415s
根据 t-SNE 模型求降维后的数据	221.105-193.415=27.690s

从用时中可以看到，对 t-SNE 的模型进行求解的用时比较长,；由于事先采用了预处理的方式读取数据，读取数据用时比较短；与 PCA 相比，整体用时要长得多。

2.2.4. LDA 方法求解

2.2.4.1. LDA 算法原理

LDA 方法是一种有监督的降维可视化方法，是通过最小化类内离散度、增大类间离散度来达到降维和分类的目的。

2.2.4.2. LDA 实现过程

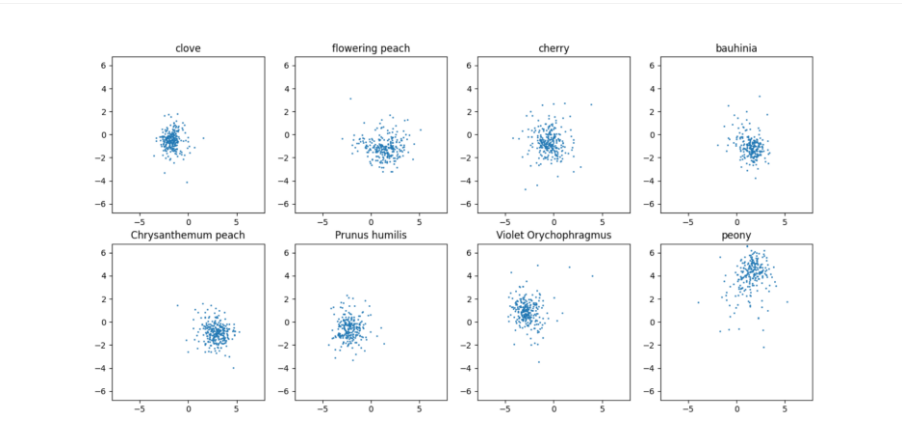
LDA 方法的实现过程如下：

1. 计算总类内离散度矩阵S_w
2. 计算总类间离散度矩阵S_b
3. 计算矩阵$S_w^{-1}S_b$
4. 计算矩阵$S_w^{-1}S_b$的特征值和特征向量，取出从大到小的前 k 个特征值对应的特征向量组成投影矩阵，W
5. 则降维后的结果为$z_i = W^T x_i$

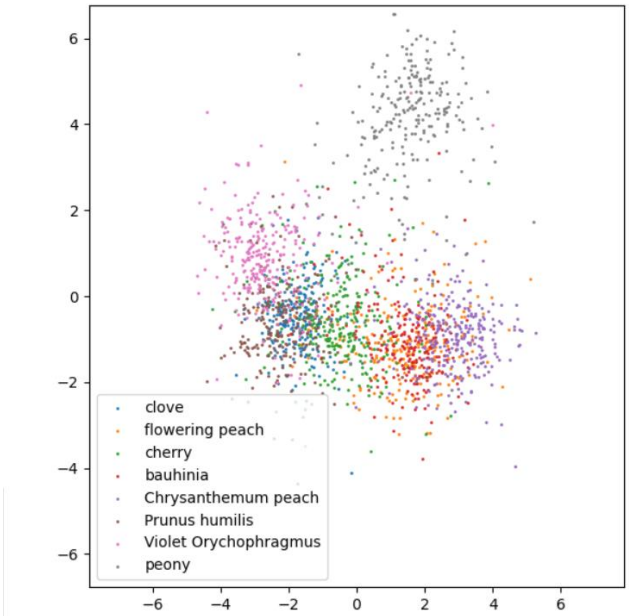
在程序实现中，我们采用 sklearn 提供的模型进行实现。

2.2.4.3. LDA 实现结果

降维结果为了更加明显，将结果先按照标签进行了分开展示，之后又将所有数据的结果放到了一起进行展示，各个花卉数据对应的降维结果分别如下。从下图的结果中可以看到，降维的效果很好，各个类别之间能够看出明显的不同，效果要比上面的 PCA 和 t-SNE 方法有明显改善。

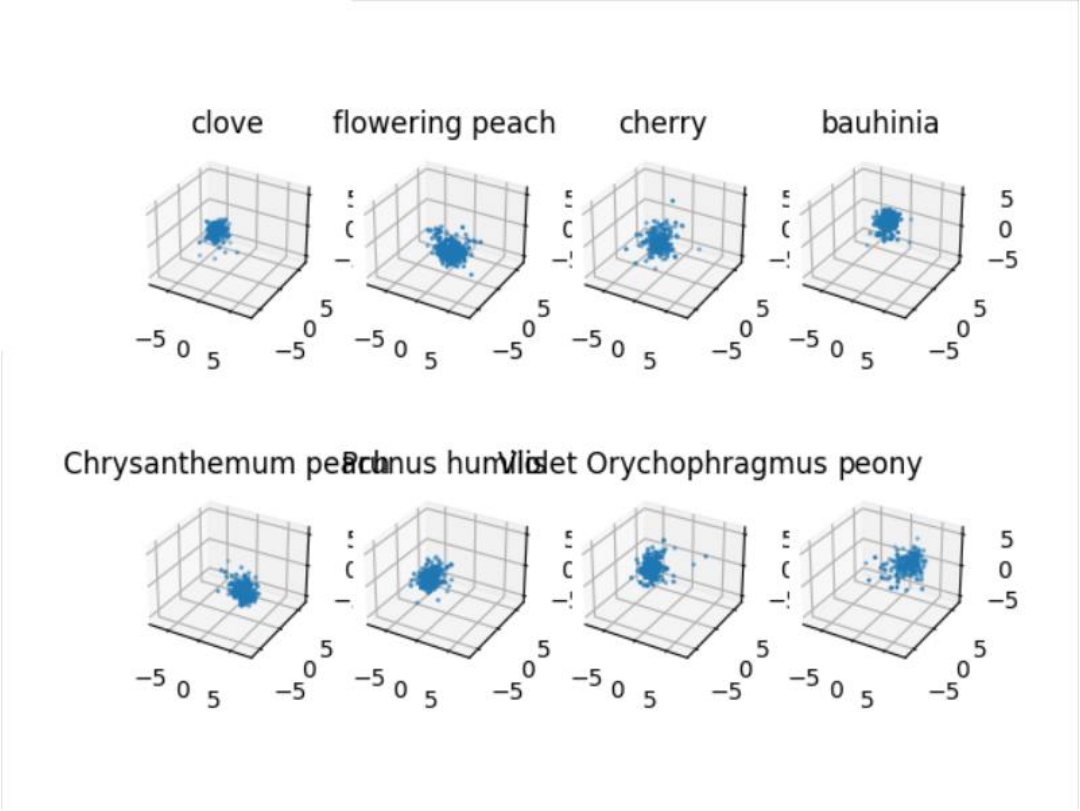


降为二维，LDA 按照标签分别展示

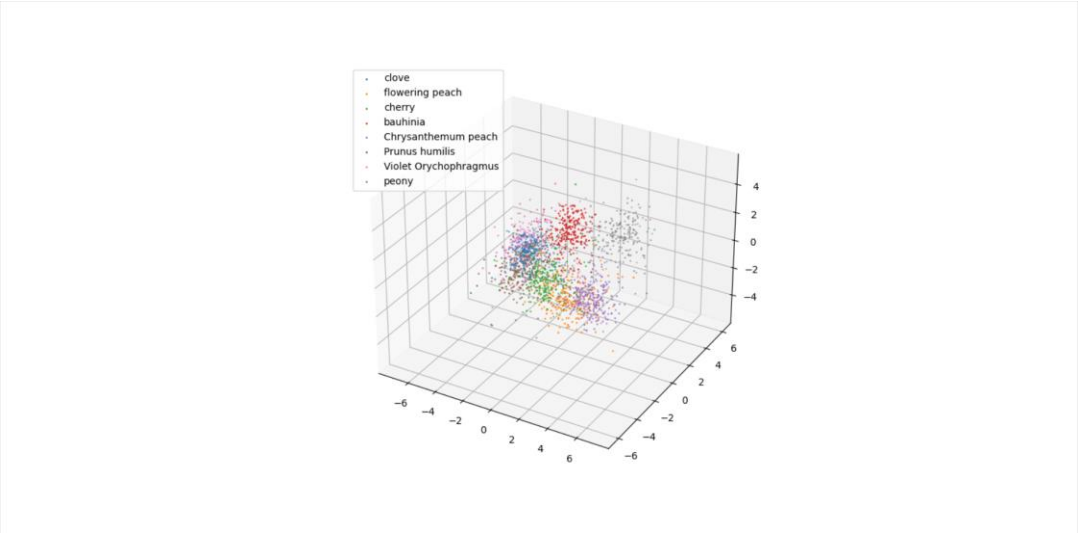


降为二维，LDA 将各个结果集中展示

=



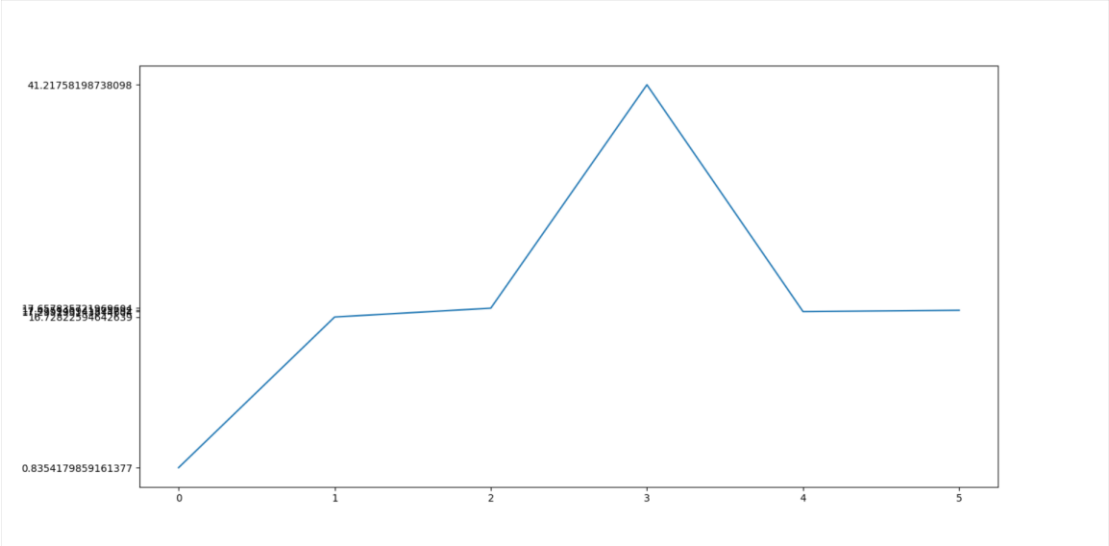
降为三维，LDA 按照标签分别展示



降为三维，LDA 将各个结果集中展示

2.2.4.4. LDA 模型性能

经过测试，当将全部的数据放入模型进行训练时，在 15 分钟内没有运行出结果，因此我在所有数据中随机选取了 2000 个点，得到的运行时间如下图



从图中给出的结果得到各个阶段的用时

读取数据	0.835s
LDA 降到两维的模型求解	16.728s
根据 LDA 模型求降维后的数据	17.243-16.728=0.515s
LDA 降到三维的模型求解	17.239s
根据 LDA 模型求降维后的数据	17.657-17.239=0.418s

从用时中可以看到，对 LDA 的模型进行求解用时比较长，；由于事先采用了预处理的方式读取数据，读取数据用时比较短；与 PCA 和 t-SNE 相比，用时要长得多，在 15 分钟内并没有运行出结果，因此采用了减少数据量得方法。经过研究，LDA 通过奇异值分解的方法进

行计算，对于给定的 14504 张图片，转换成 19200 维度的向量之后，对这个 14504*19200 的矩阵进行奇异值分解，所需要的时间太长，性能并不好。

2.2.5. 自编码器方法求解

2.2.5.1. 自编码器算法原理

自编码器方法是一种无监督的降维可视化方法，是通过编码器和解码器调整神经网络，获得在表示层的结果，实现降维。

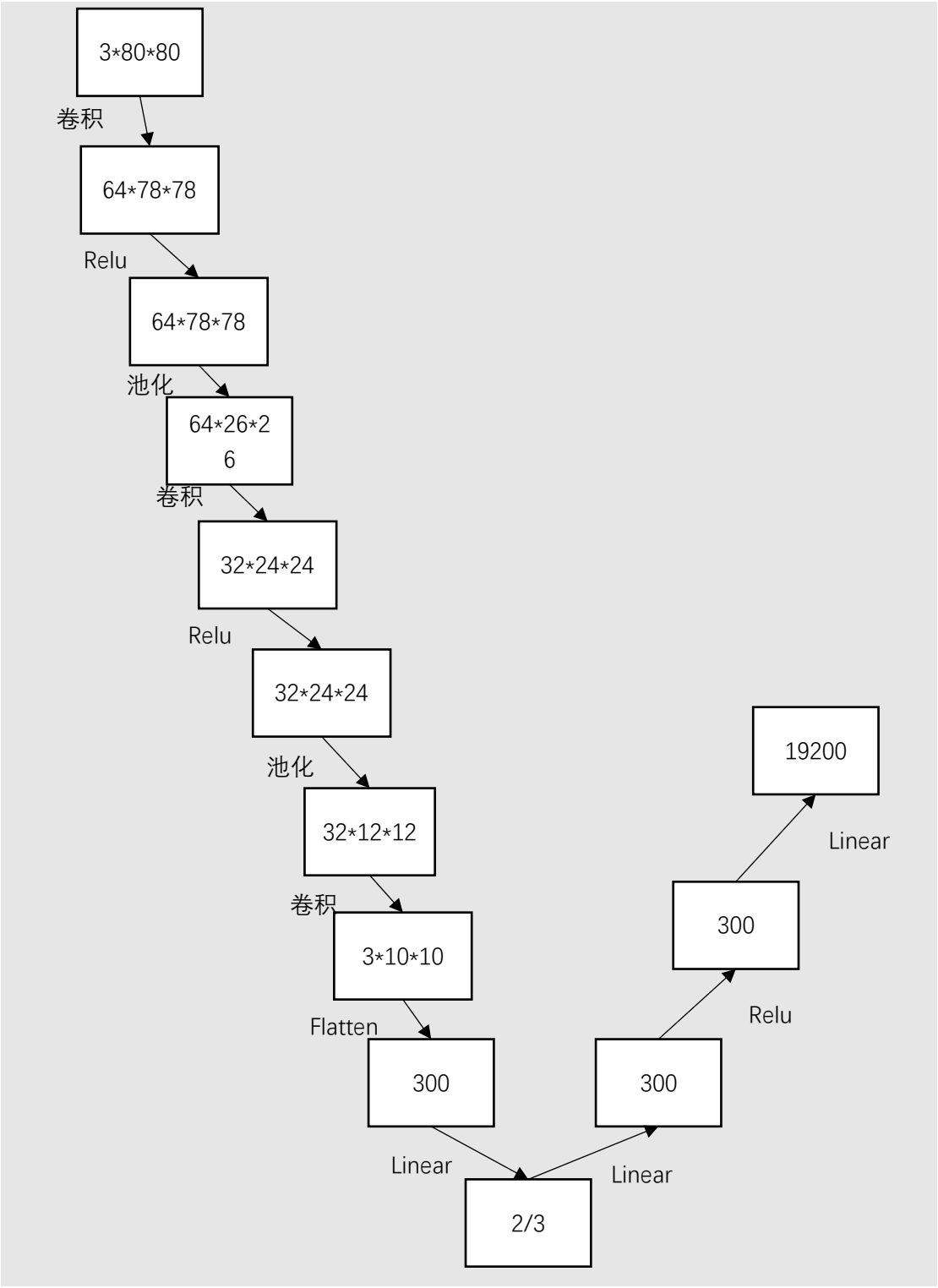
2.2.5.2. 自编码器实现过程

自编码器方法的实现过程如下：

1. 搭建编码器
2. 搭建解码器
3. 训练编码器和解码器
4. 获得训练结果

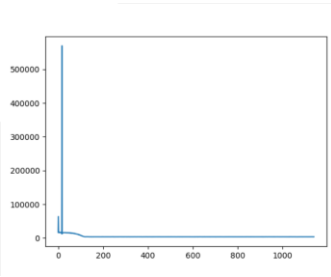
在程序实现中，我们 pytorch 搭建编码器和解码器实现。

编码器和解码器的结构如下：

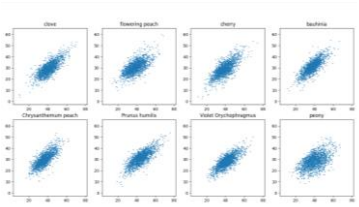


2.2.5.3. 自编码器实现结果

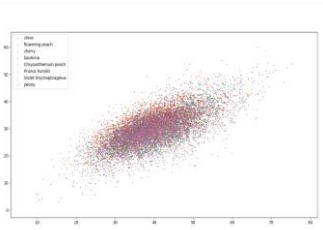
降维结果为了更加明显，将结果先按照标签进行了分开展示，之后又将所有数据的结果放到了一起进行展示，各个花卉数据对应的降维结果分别如下。从下图的结果中可以看到，各个类之间并没有完全分开，效果并不是很好。



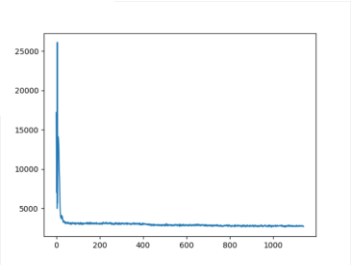
降到二维的训练 loss 曲线



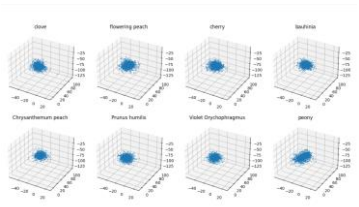
降到二维的分别的结果



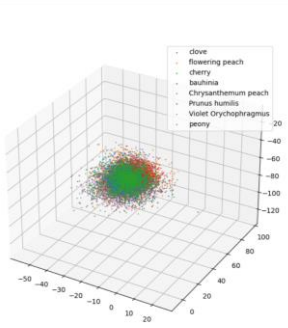
降到二维的汇总的结果



降到三维的训练 loss 曲线



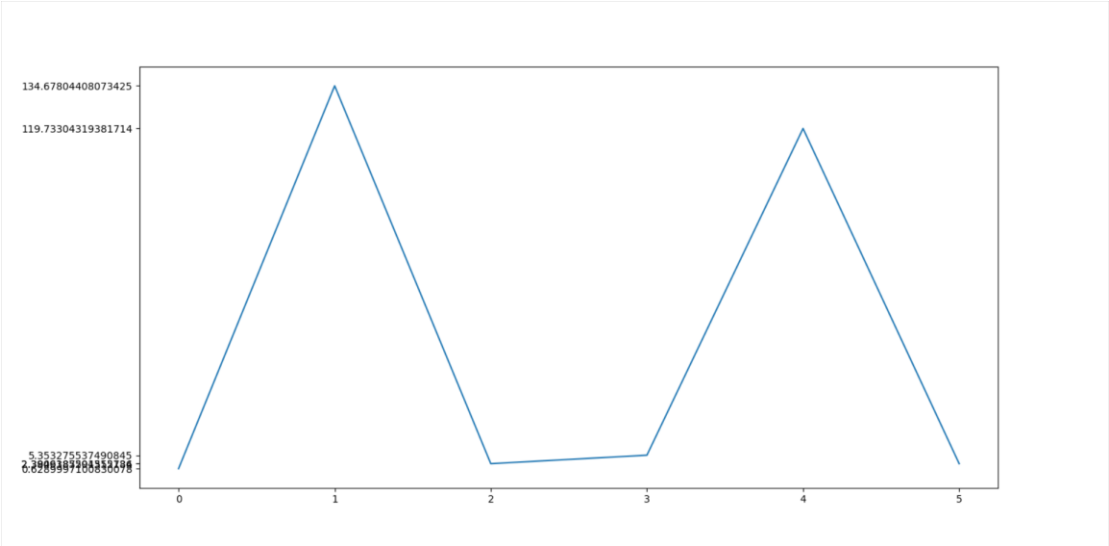
降到三维的分别的结果



降到三维的汇总的结果

2.2.5.4. 自编码器模型性能

各个阶段占用的时间如下，从图中可以看到两次用于训练的时间很长，而读取数据和预测结果用时比较短。



从图中给出的结果得到各个阶段的用时

读取数据	0.629s
自编码器降到两维的模型求解	135.678s

根据自编码器模型求降维后的数据	2.389s
自编码器降到三维的模型求解	119.733s
根据自编码器模型求降维后的数据	2.398s

2.2.6. 不同降维可视化方法比较和分析

2.2.6.1. 不同降维可视化方法比较

将上面的 PCA、t-SNE、LDA、自编码器方法的降维效果、是否有监督以及模型性能比较如下表所示：

降维方法	降维效果	是否有监督	降维性能
PCA	比较差	否	速度很快
t-SNE	比较好	否	速度比较慢
LDA	很好	是	速度非常慢
自编码器	比较差	否	速度比较慢

2.2.6.2. 不同降维可视化方法分析

从上面的比较中，我们可以看出，对于无监督的 PCA、t-SNE 和自编码器方法来说，降维效果相对比较差，在低维空间中很难将各个不同的类的花卉区分开来，这是由于没有利用标签信息，而同时各个花卉之间的特征区分又不是很明显，最终降维的效果比较差。而由于 LDA 方法是有监督的降维方法，所以最终取得的效果也比较好。

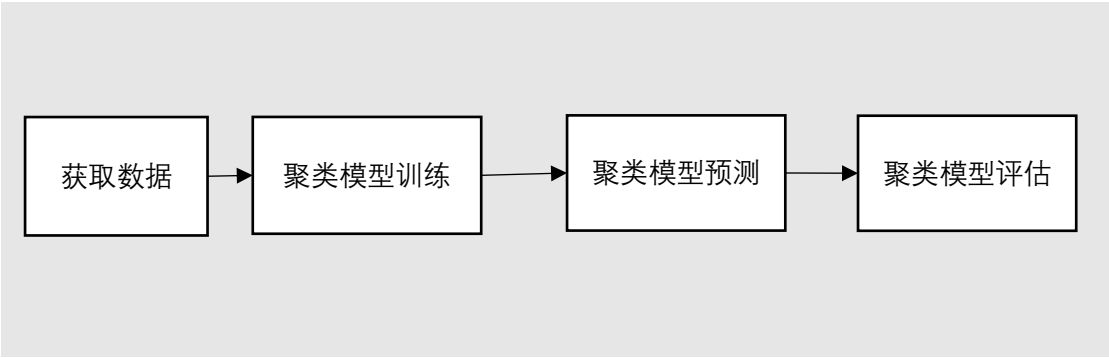
但是就降维性能来说，LDA 需要的计算量特别大，需要对特征和数据构成的矩阵进行奇异值分解，与数据量的三次方成正比，计算用时太长，并不适合用于大规模的数据求解。而其他方法相对来说，需要的计算量并没有如此之大，与数据量成正比，属于可接受的范围。

2.3. 聚类分析

聚类分析是在忽略样本标签的情况下，使用合适的聚类方法，对花卉数据集进行聚类，聚类完成之后，将数据与标签对应，计算聚类的标准化后的互信息、调整后的兰德系数和 Fowlkes-Mallows 分数。

2.3.1. 聚类流程分析

以数据预处理的获取数据的方式为基础，聚类流程如下：



2.3.2. 聚类评价标准说明

2.3.2.1. 标准化后的互信息

互信息可用于指征两个随机事件集合之间的相关性，互信息越大，类别基准与聚类结果的相关程度也越大。标准化的互信息将互信息进行了标准化。

2.3.2.2. 调整后的兰德系数

调整后的兰德系数是一种用于评价聚类效果的准则，评价的结果与最终的分类顺序无关。当结果接近 1 时，说明聚类的效果很好。

2.3.2.3. Fowlkes-Mallows 分数

Fowlkes-Mallows 统计的时两两精度和召回率的集合平均值

$$FMI = \frac{TP}{\sqrt{(TP + FP)(TP + FN)}}$$

2.3.2.4. 准确率

当共有 k 个分类结果时，对每个分类结果分别取分类数最多的一个标签作为此类的预测结果，从而计算出最终的准确率。

2.3.3. K-Means 方法聚类

2.3.3.1. K-Means 聚类原理

K-Means 聚类方法是一种动态聚类方法，通过多次迭代，将每一个点都归到一类中，得

到聚类结果。

2.3.3.2. K-Means 聚类实现过程

对于样本集合$D = \{x_1, \ldots, x_n\}$，类别数为 k

1. 初始划分 K 个聚类，选择中心点
2. 对于每一个样本计算到各个类中心的距离，并归到距离最近的类中
3. 当各个类的元素不再改变时，停止迭代

在实现中，使用 sklearn 提供的算法库。

2.3.3.3. K-Means 聚类结果

K-Means 聚类的各个指标如下所示

指标	结果
标准化后的互信息	0.117
调整后的兰德系数	0.792
Fowlkes-Mallows 分数	0.185
准确率	0.260

2.3.3.4. K-Means 聚类性能

K-Means 方法的训练和预测用时分别如下

阶段	用时
训练	122.862s
预测	0.823s

2.3.4. K-Means++方法聚类

2.3.4.1. K-Means++聚类原理

K-Means++方法与上面的 K-Means 方法的不同在于选择初始点不同，初始点需要选择相距远的 k 个初始点，能够更快的收敛和达到更好的聚类效果。

2.3.4.2. K-Means++聚类实现过程

与上面的 KMeans 方法不同的在于，不是随机选择 k 个初始点，而是逐个选择初始点，且后面选择地初始点要离已经选择样本中心点的距离尽可能远。

2.3.4.3. K-Means++聚类结果

K-Means 聚类的各个指标如下所示

指标	结果
标准化后的互信息	0.119
调整后的兰德系数	0.792
Fowlkes-Mallows 分数	0.187
准确率	0.263

2.3.4.4. K-Means++聚类性能

K-Means 方法的训练和预测用时分别如下

阶段	用时
训练	143.896
预测	0.817

2.3.5. SOM 方法聚类

2.3.5.1. SOM 聚类原理

SOM 聚类方法是通过两次的神经网络节点对不同的输入的响应不同，选择响应强度最大的神经元作为胜者，通过胜者调整相关的连接权重，使得获胜神经元对相似输入的后续响应增强的方法。

2.3.5.2. SOM 聚类实现过程

SOM 方法的过程如下：

1. 权重初始化
2. 计算竞争的响应强度
3. 选择响应强度最大的神经元作为获胜神经元
4. 以获胜神经元为中心，选择邻域函数（矩形邻域或高斯近邻函数）
5. 根据权值竞争学习的结果，围绕获胜者调整权重

在实现时，使用 sklearn-SOM 提供的库

2.3.5.3.SOM 聚类结果

SOM 聚类的各个指标如下所示

指标	结果
标准化后的互信息	0.103
调整后的兰德系数	0.801
Fowlkes-Mallows 分数	0.164
准确率	0.261

2.3.5.4.SOM 聚类性能

K-Means 方法的训练和预测用时分别如下

阶段	用时
训练	143.408s
预测	19.397s

2.3.6. PCA+K-Means 方法聚类

针对上面 K-Means 运算时间长的问题，先使用 PCA 算法进行降维，然后使用 K-Means 方法进行聚类，聚类结果和运算性能如下

指标	结果
标准化后的互信息	0.118
调整后的兰德系数	0.792
Fowlkes-Mallows 分数	0.186
准确率	0.260

阶段	用时
训练	23.366
预测	1.843

2.3.7. PCA+K-Means++方法聚类

先使用 PCA 算法进行降维，然后使用 K-Means++方法进行聚类，聚类结果和运算性能如下

指标	结果
标准化后的互信息	0.117

调整后的兰德系数	0.792
Fowlkes-Mallows 分数	0.185
准确率	0.260

阶段	用时
训练	25.302
预测	1.945

2.3.8. PCA+SOM 方法聚类

先使用 PCA 算法进行降维，然后使用 SOM 方法进行聚类，聚类结果和运算性能如下

指标	结果
标准化后的互信息	0.106
调整后的兰德系数	0.801
Fowlkes-Mallows 分数	0.167
准确率	0.260

阶段	用时
训练	24.232
预测	2.580

2.3.9. 聚类方法对比和总结

聚类方法	标准化后的互信息	调整后的兰德系数	Fowlkes-Mallows 分数	准确率	训练时间	预测时间
K-Means	0.117	0.792	0.185	0.260	122.862s	0.823s
K-Means++	0.119	0.792	0.187	0.263	143.896s	0.817s
SOM	0.103	0.801	0.164	0.261	143.408s	19.397s
PCA&K-Means	0.118	0.792	0.186	0.260	23.366	1.843
PCA&K-Means++	0.117	0.792	0.185	0.260	25.302	1.945
PCA&SOM	0.106	0.801	0.167	0.260	24.232	2.580

从上面的对比中我们看到，对于不加降维的三种聚类方法，最终的各个评价指标相差不大，其中 K-Means 比 K-Means++的用时稍微长了一点，这是由于初始值选择时，K-Means++需要做一些运算进行选择。对于加了降维之后的结果，三者差别也不大，但是在运算时间上，要比不加降维操作要短许多。

从上面的结果看，综合评价指标和模型性能，我们采用 PCA 和 K-Means++的方法可以兼顾分类效果和模型性能，采用这种方法更适合此问题。

2.4. 深度学习和非深度学习识别分类

2.4.1. 非深度学习识别分类

在本次非深度学习识别分类中，针对给出的 14504 张图片，我们采用适合小样本的 SVM 方法进行分类。

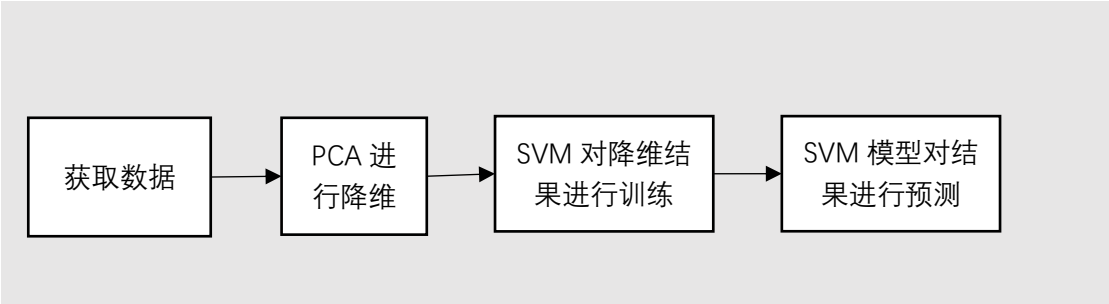
2.4.1.1. 数据处理

2.4.1.2. 算法原理

1. **PCA 降维原理：**在降维中已经做过说明，这里不再阐述
2. **SVM 分类原理：** SVM 时通过最大化类间间隔来找到最优超平面，根据最优超平面来实现分类。面对线性不可分的情况，通过引入松弛变量，来达到更好的分类效果；同时，使用了核函数的方法来应对非线性情况，能够取得更加符合实际任务需求的结果。

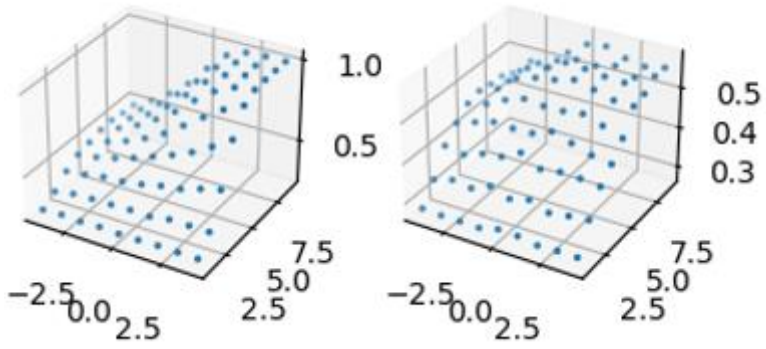
2.4.1.3. 实现过程

程序流程如下：



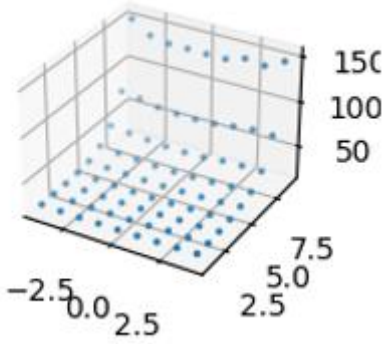
2.4.1.4. 实验结果

实验中由于可以调整 PCA 降维的维度和 SVM 的惩罚系数，我对这两个值以循环的方式进行了实验，得到训练集和验证集上的正确率与两个系数的关系，画在三维图中如下所示（图中 x，y 轴均取了以 2 为底的对数，-2.5 到 2.5 表示$2^{-2.5}$到$2^{2.5}$，是 SVM 的惩罚系数，2.5 到 7.5 表示$2^{2.5}$到$2^{7.5}$，是 PCA 降维的维度）：
从下图中可以看出，训练集上的准确率随着惩罚系数和 PCA 降维的维数的增大而在增大，最终接近 100%，但是验证集上却上升到 50%左右时不再上升。经过统计，当惩罚系数为 16，降维的维度为 256 时，达到验证集上的最好效果，达到 55.5%。



2.4.1.5. 模型性能

画出总的运算时间随降维维数和惩罚系数的变化如下(同样是取了对数),从图中可以看出,运算时间随着降维维数的升高而在增大，惩罚系数对运算时间影响不大。



2.4.2. 深度学习识别分类

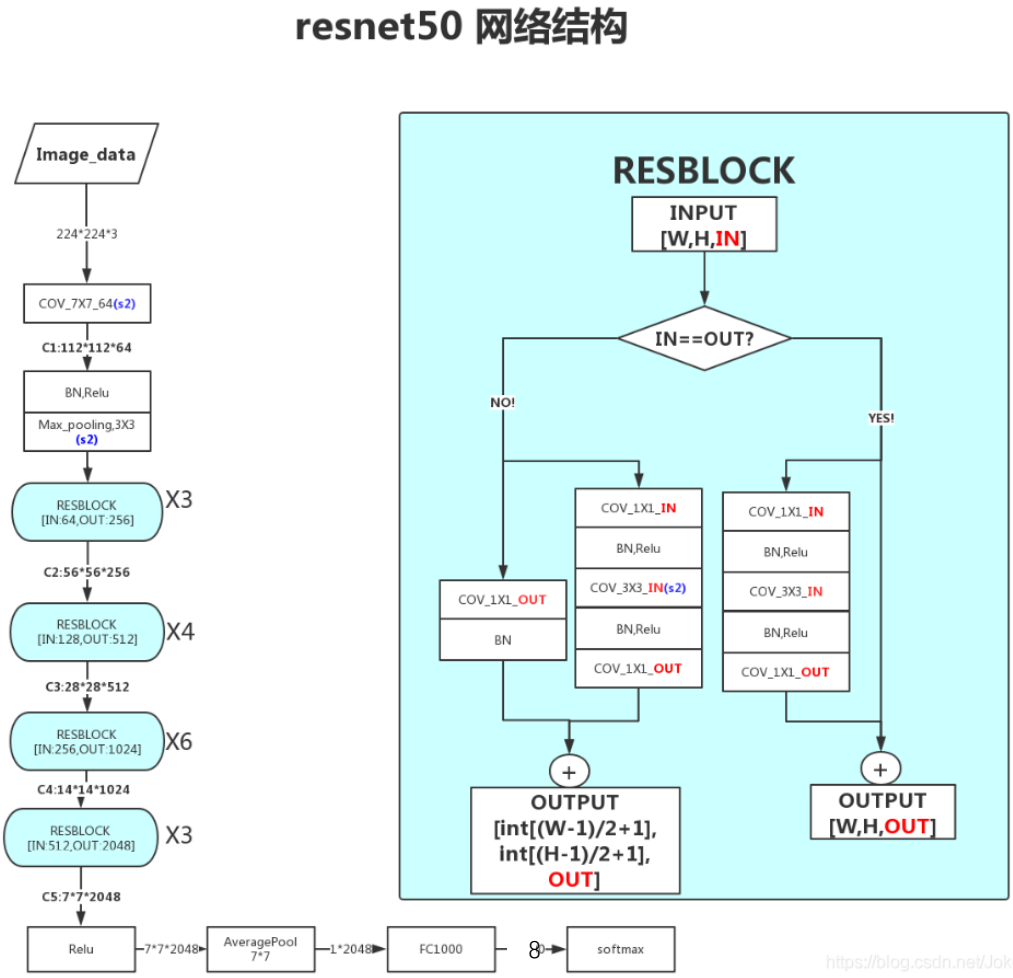
2.4.2.1. 数据处理

在使用神经网络进行分类时，将原来的 0-255 的像素数据除以 255 转化到了 0-1，然后又进行了以 0.5 概率的水平翻转，并且利用随机选择的方法将给定的数据以 1：4 划分为训练集和测试集。

2.4.2.2. 算法原理

在深度学习网络中，我采用了残差网络结构。利用残差结构，我们可以构建一个个的残差块，将输入通过一条旁路连接到输出中，通过直接相连，既保留了上一级中已经得到的特

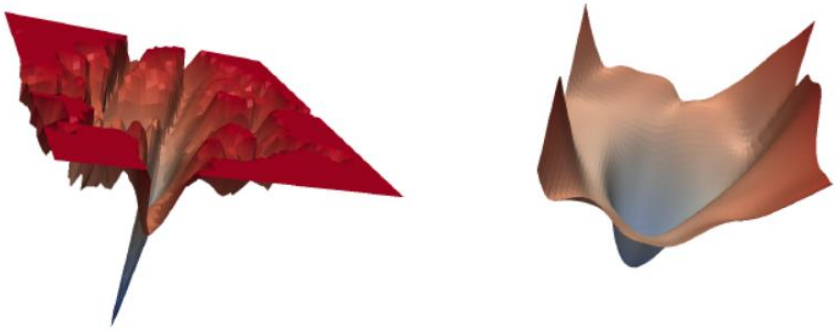
征，同时解决了梯度消失的问题，大大提高了可训练模型的深度。我采用的 resnet50 的网络结构如下图所示（在原有的网络基础上修改了输出层的网络输出类别数）：



在优化器的选择上，我选择了 SAM 优化器。SAM 优化器是为了解决模型过度参数化而提出的。在平常的只以损失值作为优化目标时很容易导致模型不够好，而 SAM 方法则是引入了一种同时优化损失值和损失锐度的方式来达到更好的分类效果。通过加入损失锐度的优化，可以大大提高模型的泛化能力，下图分别是一维情况下的损失函数示意图，在尖锐的极小值处，可以看出损失值会随着自变量的微笑改变而有较大的变化，泛化能力比较弱，而在宽的极小值点泛化能力比较强。

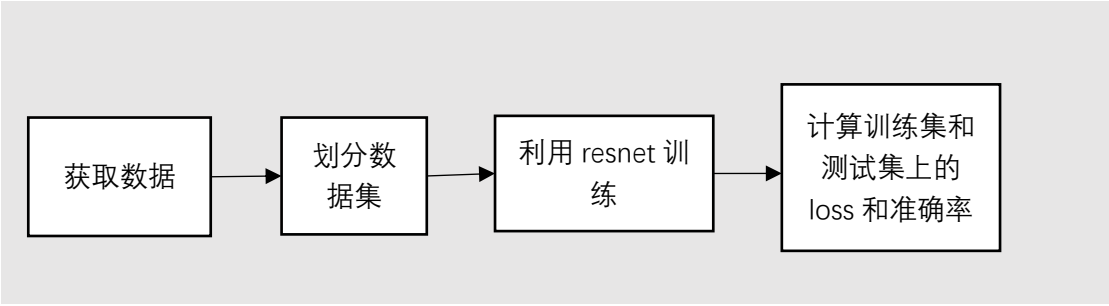
SAM 优化器的代码参考：moskomule/sam.pytorch: A PyTorch implementation of Sharpness-Aware Minimization for Efficiently Improving Generalization (github.com)

下图分别是二维情况下的锐度大和锐度小的损失曲线，使用 SAM 方法会更容易找到第二种所示的泛化能力强的结果。



2.4.2.3. 实现过程

程序流程如下：



2.4.2.4. 实验结果与模型性能结果

进行 30 次迭代，得到的训练集和测试集上的损失以及准确率和每一轮迭代的时间如下所示。从下面的表中可以看出，随着训练轮数的上升，训练集和测试集上的准确率在逐渐上升，虽然有时会出现下降，但是整体仍呈上升趋势；而损失则是呈下降趋势。最终达到的比较稳定的验证集模型分类准确率为 96%

```
epoch   0 totol time 275.7 epoch time 275.7 train loss 0.56762 train acc 0.79092 val loss 0.57144 val accuracy 0.79352
epoch   1 totol time 557.4 epoch time 281.7 train loss 0.33045 train acc 0.87882 val loss 0.37406 val accuracy 0.86694
epoch   2 totol time 837.6 epoch time 280.2 train loss 0.29347 train acc 0.89942 val loss 0.38776 val accuracy 0.86556
epoch   3 totol time 1118.0 epoch time 280.4 train loss 0.24546 train acc 0.92200 val loss 0.36624 val accuracy 0.87901
epoch   4 totol time 1399.0 epoch time 280.9 train loss 0.12936 train acc 0.95596 val loss 0.28023 val accuracy 0.89935
epoch   5 totol time 1678.7 epoch time 279.8 train loss 0.18414 train acc 0.94355 val loss 0.30825 val accuracy 0.90107
epoch   6 totol time 1958.9 epoch time 280.2 train loss 0.02832 train acc 0.99423 val loss 0.15021 val accuracy 0.95346
epoch   7 totol time 2239.6 epoch time 280.6 train loss 0.06199 train acc 0.98121 val loss 0.22730 val accuracy 0.92865
epoch   8 totol time 2520.3 epoch time 280.7 train loss 0.32325 train acc 0.91304 val loss 0.53711 val accuracy 0.86246
epoch   9 totol time 2801.3 epoch time 281.0 train loss 0.18035 train acc 0.94993 val loss 0.37366 val accuracy 0.90452
epoch  10 totol time 3082.7 epoch time 281.4 train loss 0.01277 train acc 0.99785 val loss 0.15513 val accuracy 0.95071
epoch  11 totol time 3364.0 epoch time 281.4 train loss 0.01163 train acc 0.99785 val loss 0.16981 val accuracy 0.95277
epoch  12 totol time 3645.5 epoch time 281.5 train loss 0.00712 train acc 0.99948 val loss 0.15043 val accuracy 0.95415
epoch  13 totol time 3926.8 epoch time 281.3 train loss 0.03402 train acc 0.99009 val loss 0.22607 val accuracy 0.92623
epoch  14 totol time 4208.1 epoch time 281.3 train loss 0.14105 train acc 0.95786 val loss 0.30869 val accuracy 0.90969
epoch  15 totol time 4488.8 epoch time 280.7 train loss 0.02179 train acc 0.99242 val loss 0.20321 val accuracy 0.93692
epoch  16 totol time 4769.5 epoch time 280.7 train loss 0.12702 train acc 0.96527 val loss 0.32046 val accuracy 0.91624
epoch  17 totol time 5049.0 epoch time 279.5 train loss 0.08888 train acc 0.97208 val loss 0.31498 val accuracy 0.91658
epoch  18 totol time 5329.5 epoch time 280.6 train loss 0.01610 train acc 0.99517 val loss 0.18380 val accuracy 0.94450
epoch  19 totol time 5610.5 epoch time 281.0 train loss 0.00440 train acc 0.99948 val loss 0.15776 val accuracy 0.95519
epoch  20 totol time 5891.4 epoch time 280.8 train loss 0.00300 train acc 0.99991 val loss 0.14677 val accuracy 0.95691
epoch  21 totol time 6172.0 epoch time 280.7 train loss 0.00395 train acc 0.99966 val loss 0.15095 val accuracy 0.95967
epoch  22 totol time 6453.2 epoch time 281.2 train loss 0.00255 train acc 0.99991 val loss 0.13561 val accuracy 0.96139
epoch  23 totol time 6734.7 epoch time 281.5 train loss 0.02444 train acc 0.99155 val loss 0.20468 val accuracy 0.94554
epoch  24 totol time 7016.0 epoch time 281.3 train loss 0.02129 train acc 0.99483 val loss 0.16213 val accuracy 0.95588
epoch  25 totol time 7297.2 epoch time 281.2 train loss 0.00170 train acc 1.00000 val loss 0.14372 val accuracy 0.96346
epoch  26 totol time 7578.5 epoch time 281.3 train loss 0.00667 train acc 0.99828 val loss 0.15481 val accuracy 0.95795
epoch  27 totol time 7859.5 epoch time 281.1 train loss 0.00234 train acc 0.99983 val loss 0.15141 val accuracy 0.96001
epoch  28 totol time 8140.7 epoch time 281.1 train loss 0.09119 train acc 0.97208 val loss 0.28448 val accuracy 0.92623
epoch  29 totol time 8421.8 epoch time 281.1 train loss 0.00167 train acc 0.99983 val loss 0.14665 val accuracy 0.96070
```

2.4.2.5. 模型性能分析

由于采用了 resnet50 网络进行训练，训练用时比较长，每轮用时约 280s，但是最终的模型预测效果很好，能够达到比较稳定的 96%。

3. 总结

这次实验中，我分别针对降维可视化、聚类以及图片分类问题做了研究。在读入图片数据方面，我才用了事先将图片存为 numpy 数组的方式加速数据读取；降维可视化中，我采用了 PCA、t-SNE、LDA、自编码器等四种方式进行，其中，有监督的降维方法 LDA 的降维后的图片分类效果很好，而其余的非监督方法则相对比较差；在聚类问题中，我采用了 K-Means、K-Means++、SOM 的方法进行聚类，之后我又结合了降维方法，对上面的三种聚类方法进行了加速，最终取得了和直接聚类相似的聚类效果、同时大大提升了运算效率；在图像分类问题中，在非深度学习方法中，我使用了 SVM 进行多分类任务，考虑到 SVM 的分类效率，我先对数据进行了降维，由于其中需要调整两个参数，我采用了两重循环遍历的方法对两个参数进行了调整，最后发现当降维数为 256、SVM 分类器的惩罚系数设为 16 时，取得的分类效果最好，能够达到 55.5%；在深度学习方法中，我采用了 resnet50 进行分类，在数据处理阶段，加入了 resize 和随机水平翻转的方式对数据进行增强，同时我才用了 SAM 优化器，得到了泛化能力很好的网络结果。

从这次实验中，我又系统地回顾了模式识别过程课上学习到的各种降维、聚类以及分类方法，有了很大的收获。