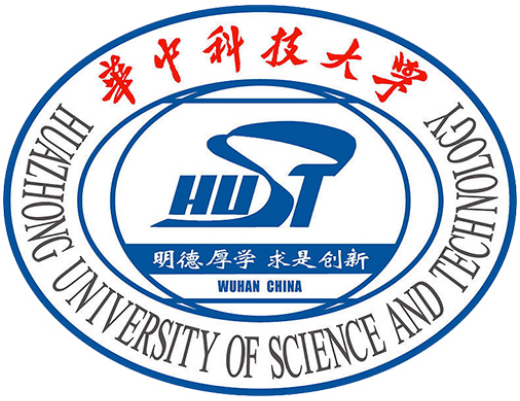


课程大作业

作业名称：基于拓扑感知损失的管状图像分割

参考文献：*Xiaoling Hu, Li Fuxin, Dimitris Samaras, Chao Chen, Topology-Preserving Deep Image Segmentation. NeurIPS(2019).*

课程名称：机器学习基础&生物医学数字信号处理

专业班级：工程科学（生医）1801 班

学　　号：U201813614

姓　　名：罗正潮

指导教师：全廷伟

日　　期：2021/7/14

工程科学学院（国际化示范学院）

目录

实验原理 3
实验背景 3
数学原理 3
算法原理 5
实验设置 11
数据集 11
网络结构 11
训练方法 11
实验结果 12
评估指标 12
样本对比 13
实验代码 15
参考资料 15

实验原理

实验背景

分割算法容易在精细结构上出现拓扑错误，例如小尺寸样本、具有多个连接组件的样本和细连接。这些精细的结构对于分析对象的功能可能至关重要。例如，精确提取细绳和手柄等薄部件对于规划机器人动作至关重要，如拉动或抓取。在生物医学图像中，神经元膜和血管等薄物体的正确描绘对于提供底层系统的精确形态和结构量化至关重要。参考文献提出了一种新的学习正确拓扑分割的方法。

数学原理

下述数学原理参考了参考资料[4][5]以及网课[6]。

Simplicial Complex

d-simplex：d+1 点的凸包

d-simplex 的 face：d-simplex 中 d+1 点的子集构成的凸包

d 维的 Simplicial Complex：维度最高为 d 的 simplex 的集合 K，满足条件：

1、任意 K 中的 simplex 的 face 也属于 K

2、任意 K 中的两个 simplex 的交集要么为空，要么为二者的公共 face

Cubical Complex

elementary interval：unit interval [k, k+1]或者 degenerate interval [k, k]

d 维空间中的 cube：d 个 elementary interval 的乘积$I: \prod_{i=1}^{d} I_i$

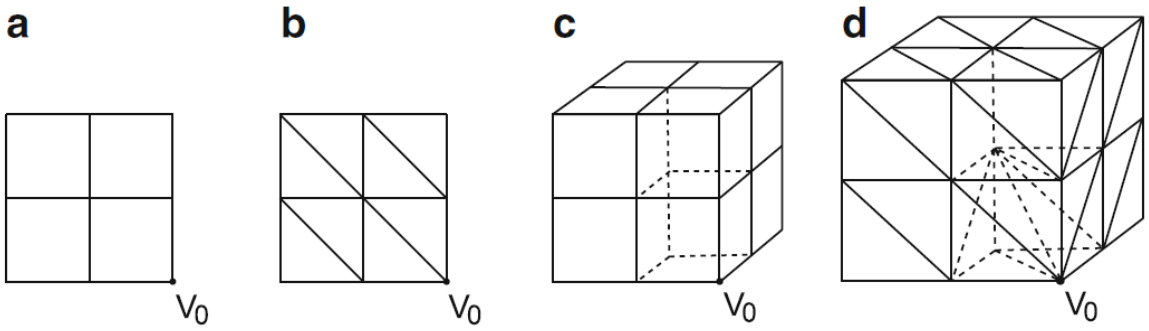
cube 的维度：乘积中非 degenerate interval 的数量

cube 的 face：在 d 为空间内的 cube 的子集

0-cube、1-cube、2-cube、3-cube 分别为 vertex、edge、square 和 3D cube（voxel）

d 维的 Cubical Complex：维度最高为 d 的 cube 的集合，且和 simplicial complex 类似，必须在取 face 和 intersection 下封闭。

2 维、3 维 cubical complex 和 simplicial complex 的例子：



(a)2 维 cubical complex；(b)2 维三角化（三角化是 simplicial complex 的一种表现形式）simplicial complex； (c) 3 维 cubical complex；(d) 3 维三角化 simplicial complex；

此外，当从 Simplicial Complex 转换到 Cubical Complex 时，complexes 的尺寸显著减小。同一数据的 Simplicial Complex 表示和 Cubical Complex 表示的尺寸之比随着数据维度呈指数增长（证明见[1]）。

Cell

d 维 cell 指代 d 维 simplex 或 cube

d 维 cell 的 boundary：d 维 cell 的（d-1）维 face 的集合

d 维 cell 的 coface：d 维 cell 本身是其 face 的 cell

d-chain：一个 d 维 cell 集合的 boundary，是集合中每个 d 维 cell 的 boundary 的模 2 和

d-cycle：boundary 为 0 的 d-chain

boundary 矩阵：[n_d-1*n_d]，即 d 维 boundary 算子的表达形式。矩阵中每一列为 d 维 cell 的 boundary，每一行包含（d-1）维 cell

n_d：complex 中 d 维 cell 的个数

Persistent Homology

homology：homology 类是一类等价的 cycle（d 维流形），等价的 cycle 差是(d + 1)维 boundary。其代表了拓扑结构信息关注 Z2 homology

homology 类：0 维为 component，1 维为 tunnel，2 维为 void。

filtering 函数：将拓扑空间映射到实数域的函数，在参考论文中即为根据阈值α分割的规则。

persistent homology：给定拓扑空间和 filtering 函数，persistent homology 研究超水平集的 homological 变化，随着依赖时间（参考论文为阈值α）的超水平集生长，算法会捕捉 homology 类的生灭时间（阈值α）（生指 homology 类出现，灭指该类消失或与先生的类融为一体）

在参考论文中，似然图像由神经网络预测。persistence homology 可以用于捕获似然图像在所有可能阈值上携带的拓扑信息。

persistence：生灭时间（阈值α）之差。persistence 越大的 homology 类越能体现拓扑空间的全局结构信息。

persistence diagram：有一组点的二维平面，每个点对应一个 persistence homology 类，点的坐标分别就是生灭时间（阈值α）。是一种广为接受的 persistence 可视化方式。

betti 数：拓扑空间中 homology 类的个数

0 阶 betti 数代表拓扑空间中 component 的数量，1 阶 betti 数代表拓扑空间中 tunnel 的数量

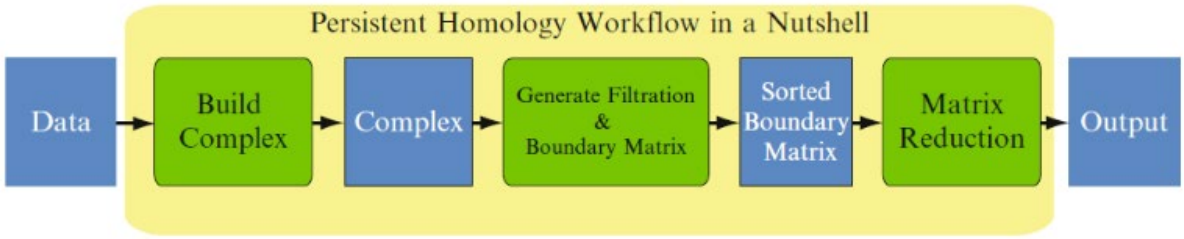
算法原理

下述算法在 python 库 gudhi 中有编写好的接口。下列原理重点在于解释原理，而不在于接口的使用。要了解接口的使用方法，请看参考资料[3]。

构建 Cubical Complex 并获取 persistence diagram：CubPers

gudhi 库中存在类 gudhi.CubicalComplex。依照使用方法输入图像矩阵即可获得 Cubical Complex。

该库的实现依据参考资料[1]中介绍的 CubPers 算法，下图为 CubPers 算法离散化图像、构建 Complex 并获得 persistence diagram 的流程图：



流程分三步，即图中三个绿框。左起第一个绿框为构建 Cubical Complex，余下两个绿框用于输入 Cubical Complex 得到 persistence diagram。

之所以使用 Cubical Complex，是因为这种 Complex 表示不需要细分输入的图像（符合图像矩形的特点）。其次，使用 Cubical Complex 还有两个优点：

1、相比与 simplicial complex，complexes 的尺寸被大幅减小，特别是高维情况（证明见[1]）

2、允许使用更紧凑的数据结构（即 CubeMap）

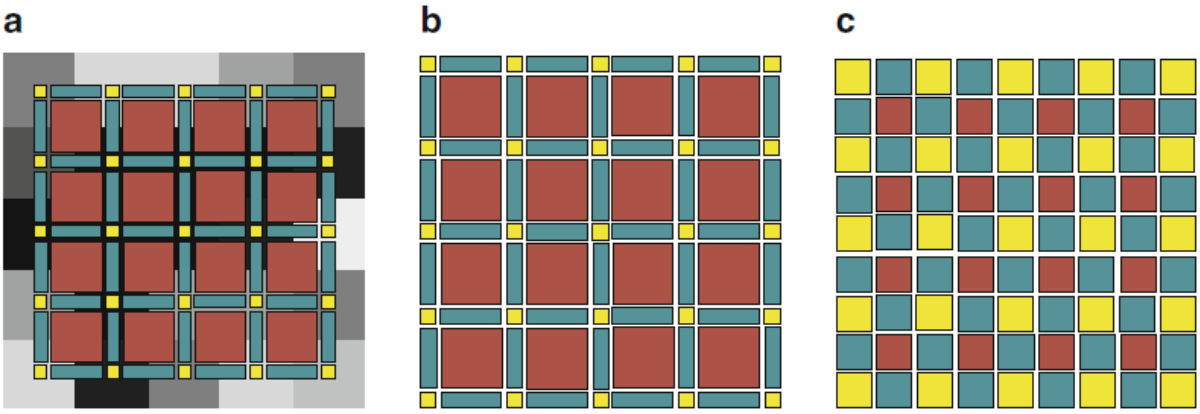
计算拓扑学历史上首个计算 persistence 的算法的时间复杂度正比于 complex 尺寸的立方。计算 persistence 分两类，一种是 exact，另一种是 approximative，CubPers 关注第一种。

对于图像，函数 f 定义在所有像素/体素上。首先， complex 中 vertex 的值被计算。接下来，计算 complex 的 filtration，并生成排好序的 boundary 矩阵。boundary 矩阵即是约简算法的输入。

filtration 可以描述为将按函数值降序排列的 cell 一个接一个地添加到一个 complex 中。为了实现这一点，采用 filtration-building 算法，通过给每个 cell 分配其中 vertex 的最大值，将函数值扩展到 complex 的所有 cell 中。再将所有 cell 按函数值降序排列，使得每个 cell 都在其所有 face 之前被加入 filtration，即可得到被称为 upperstar filtration 的 cell 序列。计算了 cell 的顺序后，可以生成排好序的 boundary 矩阵。

在参考文献中，filtration 即为用所有可能的阈值α对似然图像做阈值分割，在按阈值α降序得到的拓扑分割二值图序列。

CubPers 算法引入了 CubeMap 作为数据结构，存储每个 cube 的附加信息(函数值、filtration 顺序)。设输入维度 d、尺寸为 w^d，w 是每个维度中的 vertex 的数量。用（2w-1）^d 个元素形成 d 维数组存储 cell 的附加信息。该数组由 3^d 大小的数组的重叠拷贝组成。这种结构称为 CubeMap。二维示例如下：



(a)构建在灰度 2D 图像上的 cubical complex，5×5。每个 vertex(黄色)对应一个像素。对应构造 edge(蓝色)和 cube(红色)（即 square)。(注意 0-cube、1-cube、2-cube 分别为 vertex、edge、square) (b)cubical complex 本身。(c)对应的 CubeMap，所有用于 filtration-building 的信息都在 9×9 阵列编码。每个元素对应一个 cube

在上图中，由于结构规则，cell 之间的关系可以立即从它们的坐标读取。在一个 9×9 阵列中为每个 cell 存储必要的信息(如 filtration 中的顺序，函数值)（上图 c)。构建上图之后，就可以立即得到任何 cell(无论是 vertex、edge 还是 square)的维度，以及其 face 和 coface。我们通过以 2 为模检查坐标来做到这一点。偶坐标对应于 cube 的 degenerate interval。

CubeMap 的主要优点是提高了存储效率。boundary 关系隐式编码在 cell 的坐标中。坐标本身也是隐式的。此外，可以随机访问每个单 cell 并快速定位其 boundary。

将图像输入依照 Cubical Complex 理论离散化存储在 CubeMap 之后，可以利用下述算法生成 filtration 与 boundary 矩阵：

Algorithm 1: Computing filtration and sorted boundary matrix

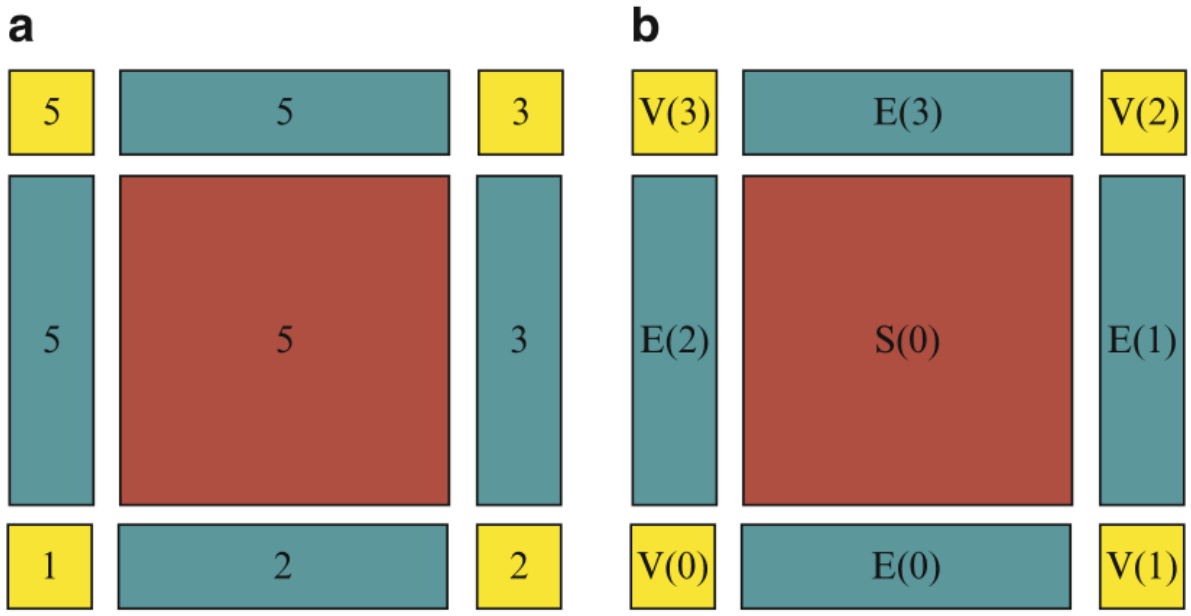
Input: function f, given on vertices of a cubical complex K
Output: sorted boundary matrix, extension of function f to all cubes of K

```
sort vertices of K by values of f (descending)
for each vertex V_i in sorted order do
    for each cube C_j with V_i as one of its vertices do
        if C_j was not assigned filtration index then
            assign next (smaller) filtration index to C_j
            f(C_j) ← f(V_i).
for each cube C_i of K do
    column ← filtration index of C_i
    for each cube B_j in boundary of C_i do
        row ← filtration index of B_j
        boundary matrix(row, column) ← 1
```

上述 filtration-building 算法背后的直觉是，当我们以降序迭代所有 vertex 时，我们知道没有加

入 filtration 的 vertex 的 coface 属于它们的 lower-star，并且可以加入 filtration。由于相邻 cell 的索引可能还没有计算出来，故不能在同一步中建立 boundary 矩阵。在第 5 行，filtration 索引从高到低进行分配。算法的结果是一个排好序的 boundary 矩阵，每一个 cell 和它们的似然函数值之间的邻接关系被编码在 boundary 矩阵中，可以作为约简步骤的输入。

下图为算法的图示：



(a)赋给 vertex 并扩展到所有 cube 的函数值。(b)cell 在 filtration 中被分配索引。每个维度的索引分开。vertex 标记为 V，edge 标记为 E，square 标记为 S

在 filtration-building 算法的代码实现中，反转了在 vertex 上迭代的顺序，简化了算法的第一次迭代。用可变的 d 维数组存储数据，随机存取的时间复杂度为 O(d)。设离散化后的 Cubical Complex 中 vertex 的个数为 n，则在使用 CubeMap 数据结构的情况下，算法可以在 O(3^dn+d2^dn)时间和 O(d2^dn)内存下实现。(具体分析见参考资料[1])

在约简步骤中，使用改进的高斯消去算法作为约简算法。约简后的矩阵对所有的 persistent homology 信息进行了编码，矩阵中的 pivoting entry 对应于 persistent diagram 中的所有点。该算法的复杂度是矩阵维度的立方，而矩阵维度又与图像大小成线性关系。

依据 persistence diagram 计算 betti 数

由于 persistent homology 关注 complex 的拓扑特征，即使两张图片的 betti 数相等也并不能保证两张图片的几何特征一致。参考文献采取了将输入图像采样多个 patch 的做法。patch 大小为[65×65]，这样得到的结果在保证拓扑特征一致的同时将几何特征限制在了 patch 范围内。

输入图像的 patch 之后，先在 patch 的四周像素上赋 0 值，相当于给 patch 加了一圈黑色边框。这有效解决了 patch 常常只包含几个分叉，难以形成 tunnel 的问题。

加入边框后，生成 patch 的 1 维 persistence diagram，则 patch 对应的 1 维 betti 数即为 1 维 persistence diagram 列表的长度。

依据 persistence diagram 构建 topoloss 及其梯度

参考文献设计了一个连续值损失函数，该函数强制分割具有与 ground truth 相同的拓扑结构，即具有相同的贝蒂数。用该损失训练的神经网络将在不牺牲每像素精度的情况下实现高拓扑保真度。

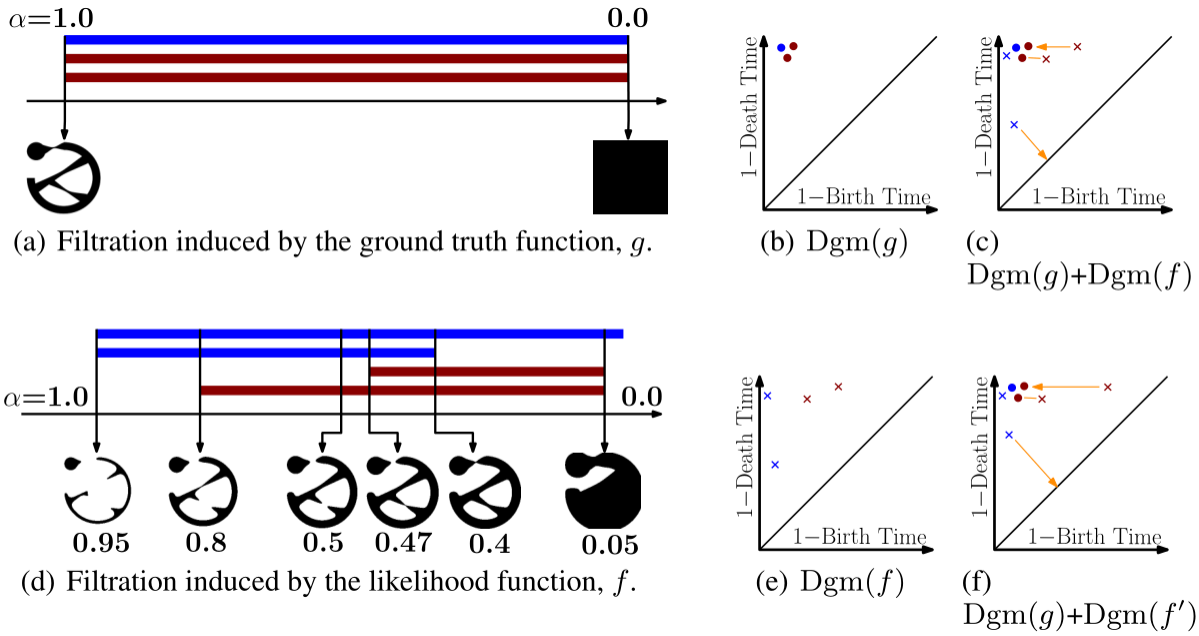
新的拓扑损失如下：

$$L(f,g) = L_{bce}(f,g) + \lambda L_{topo}(f,g)$$

这里 f 是网络预测的似然图，g 是 ground truth，λ控制拓扑损失的权重。每个训练图像上的损失函数是每像素交叉熵损失和拓扑损失的加权和。

根据 CubPers 算法得到 Dgm(f)和 Dgm(g)后，在两组点之间找到最佳的一对一对应关系，并测量它们之间的总平方距离。再将不匹配的点与对角线（对角线代表 persistence 为 0，生灭时间（阈值α相等））匹配。与对角线匹配的点对应于噪声 component 或噪声 loop。被匹配到对角线即是指点对应的 component/loop 将被移除或与其他 component/loop 合并。

匹配算法如下。Dgm(g)中总共 k (=Betti 数)个点位于左上角(0，1)（α=1 时生，α=0 时灭）在 Dgm(f)中找到最靠近(0，1)的 k 个点，并将它们与(0，1)进行匹配。Dgm(f)中剩余的点与对角线匹配。该算法计算并排序 Dgm(f)中所有点到(0，1)的平方距离。复杂度为 O(nlogn)，n = Dgm(f)中的点数。一般来说，最快能在 O(n3/2)时间内匹配两个图。下图是参考论文中示例图片离散化后进行 filtration 得到的 diagram 和匹配情况：



(a) Filtration induced by the ground truth function, g. (b) Dgm(g) (c) Dgm(g)+Dgm(f)

(d) Filtration induced by the likelihood function, f. (e) Dgm(f) (f) Dgm(g)+Dgm(f')

persistent homology 的图示。上图左为 f 与 g 的 filtration。蓝色和红的条分别代表 component 和 tunnel。(a)对于 g，所有结构在 α = 1.0 时诞生，在 α = 0 时消亡。(d)对于 f，从左到右依次为：两个 component 的生，较长 tunnel 的生，在 α = 0.5 处的正常分割图像，较短 tunnel 的生，较短 component 的灭，两个 tunnel 的灭。(b)和(e)为 g 和 f 的 persistence diagram。(c)为两个 diagram 的重叠。橙色箭头表示点之间的匹配。f 中的较短 component(蓝色十字)与对角线匹配。 (f)g 和最差似然 f‘的 diagram 的叠加。显然 f’下的橙色箭头更长。

设Γ是 Dgm(f)和 Dgm(g)之间所有可能双射的集合。拓扑损失为:

$$\min_{\gamma\in\Gamma}\sum_{p\in\mathrm{Dgm}(f)}||p-\gamma(p)||^2=\sum_{p\in\mathrm{Dgm}(f)}[\mathrm{birth}(p)-\mathrm{birth}(\gamma^*(p))]^2+[\mathrm{death}(p)-\mathrm{death}(\gamma^*(p))]^2$$

其中 γ*是由匹配算法得到的两个点集之间的最佳匹配。

拓扑损失取决于发生拓扑变化的生灭阈值。这些阈值由拓扑变化发生的位置唯一确定。当基础函数可微时，这些生灭位置就是临界点，即梯度为零的点。在网络训练过程中，似然函数 f 是由像素处的网络预测值控制的分段线性函数。对于这样的 f，临界点总为一个像素，因为拓扑变化总是发生在单个像素上。用 ω 表示神经网络参数。对于 Dgm(f)中的每个点 p，用 c_b(p)和 c_d(p)表示点对应的拓扑结构的临界点，有拓扑损失的梯度$\nabla_\omega L_{topo}(f,g)$为:

$$\sum_{p\in\mathrm{Dgm}(f)}2[f(c_b(p))-\mathrm{birth}(\gamma^*(p))]\frac{\partial f(c_b(p))}{\partial\omega}+2[f(c_d(p))-\mathrm{death}(\gamma^*(p))]\frac{\partial f(c_d(p))}{\partial\omega}$$

相关证明见参考文献。

实验设置

数据集

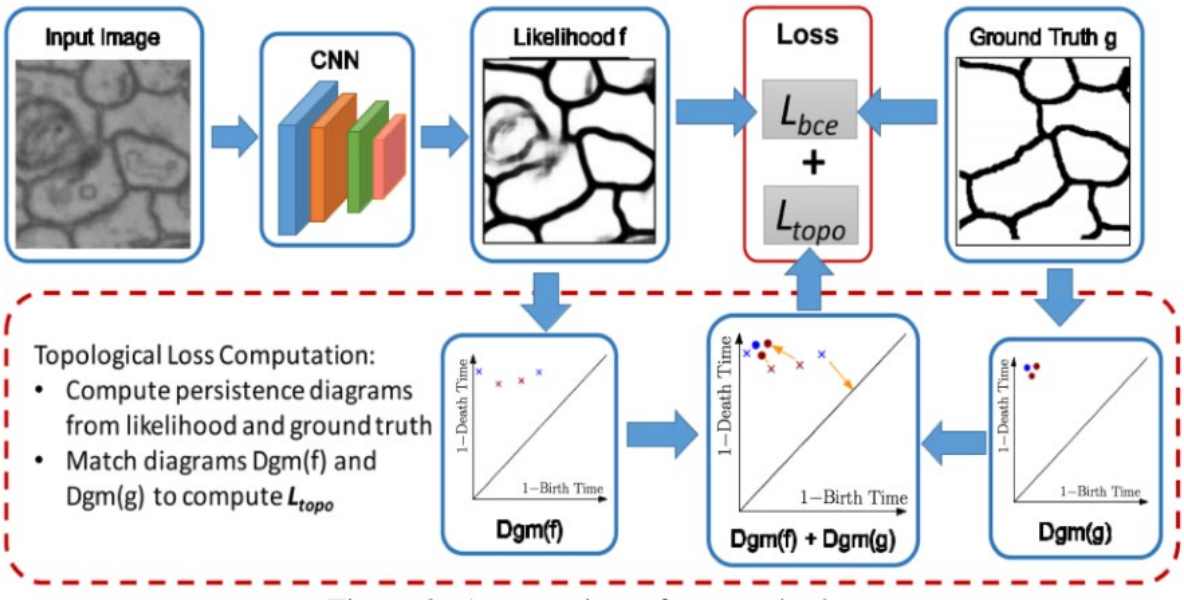
Road[2]数据集有来自马萨诸塞州道路数据集的 1108 幅训练图像、14 幅验证图像以及标签。单幅图像的分辨率为 1500×1500。图像都为 3 通道 tiff 图像，标签为单通道二值图像。实验训练任务为输入训练图像和标签，训练神经网络，将神经网络用于输出验证图像的标签。

网络结构

训练使用 U-Net 架构。网络包含六个可训练的权重层、四个卷积层和两个全连接层。第一、第二和第四个卷积层之后都跟一个 2 × 2 的最大池化层，在该层结束时跨度为 2。

训练方法

参考文献所提出的拓扑损失是可微的，具体公式见报告算法原理部分。下图为训练的定性流程图：



训练中一共经历了 120 次 epoch。其中，前 100 次 epoch 仅使用了交叉熵损失，后 20 次 epoch 加入了拓扑损失继续训练。之所以不直接使用拓扑损失，是因为获取 persistence diagram 需要一定可信度的似然图像。在一开始就使用拓扑损失会使得模型无法获得生成大致正确似然图像的能力。每当训练次数到达记录点，就通过将预测似然图的阈值设定为 0.5 来生成分

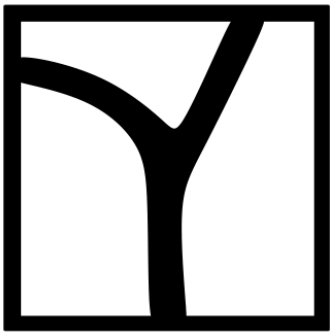
割。

由于计算的复杂性，在整个训练过程中使用 65 × 65 的 patch 进行训练。原因有二：

1、拓扑信息的计算相对昂贵。

2、似然图的 persistence diagram 和 ground truth 之间的匹配过程可能相当困难。如果补丁大小太大，Dgm(g)中会有很多点，Dgm(f)中甚至会有更多。匹配过程过于复杂，容易出错。通过聚焦于更小的 patch，方便网络定位拓扑结构并逐个优化。

patch(65 × 65)通常只包含分支结构，而不包含 loop。为了进行有意义的拓扑度量，在 patch 边界填充一个黑色框架后计算拓扑，以避免琐碎的拓扑结构。图示如下：



通过随机 patch 采样，拓扑损失可以在每个采样的 patch 中的拓扑特征与 ground truth 一致。另一方面，尽管 patch 相对于 ground truth 可能存在变形，但是变形被限制在 patch 内。

在训练过程中，即使对于同一个 patch，Dgm(f)、临界点和梯度也会发生变化。在每个 epoch 都对斑块重新取样，获取其 persistence diagram 和拓扑损失梯度。在计算了一个 batch 输入的所有 patch 的拓扑梯度之后，将梯度聚合起来反向传播。

实验结果

本实验在 GTX1080 显卡上运行，计算机系统版本为 Ubuntu 18.04.2 LTS。前 100 次 epoch 用时 1 天，后 20 次 epoch 共用时 2 两天。

评估指标

使用两种不同的评估标准。一个是平均像素精度，像素精度是正确分类像素的百分比。一个是平均交叉熵损失，定义如下：

$$\text{loss}(x, class) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right) = -x[class] + \log\left(\sum_j \exp(x[j])\right)$$

交叉熵损失在分类问题中得到广泛的运用。

样本对比

参考文献的方法在准确度和损失上取得了更好的性能，这直接说明了拓扑的正确性。在 Road 数据集上说明了该方法的有效性：

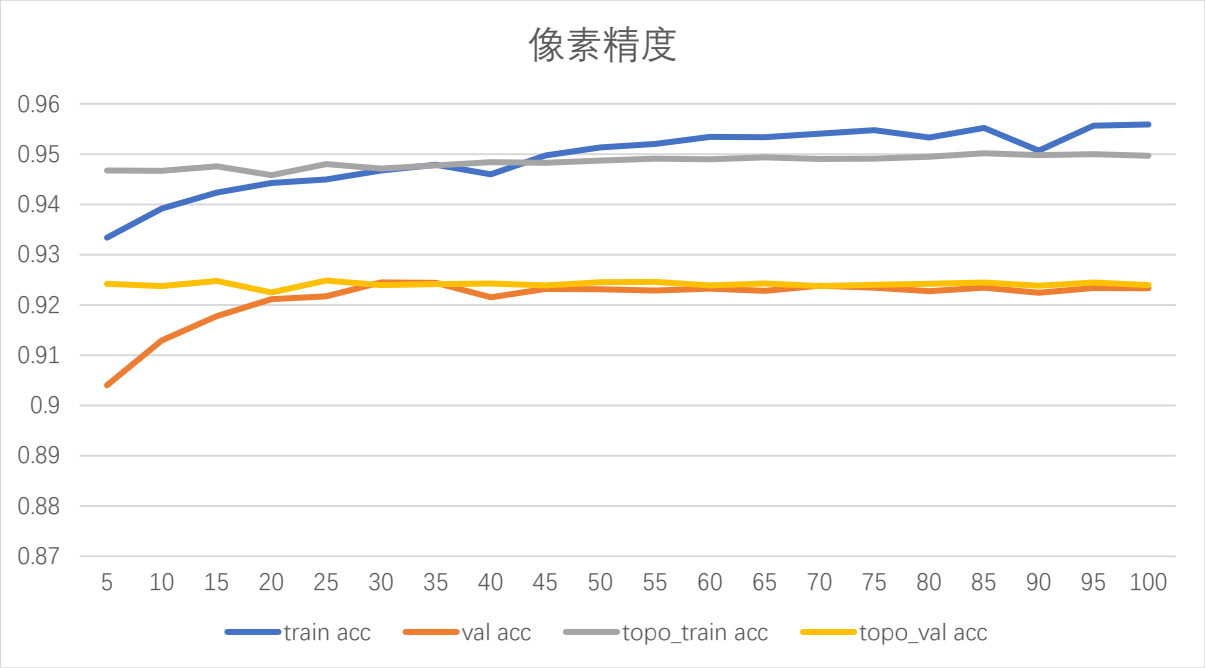
上图为从验证集中选取的 14 张图片中较有代表性的结果。上图左为 ground truth，上图中为加入拓扑损失后训练得到的分割，上图右为仅使用交叉熵损失训练得到的分割。可以看到，ground truth 中连通而仅用交叉熵函数训练得到分割中断开的像素块在加入拓扑损失之后得到了明显改善。（PS：120 个 epoch 的验证集分割结果在代码.7z 中，路径为：代码\Road_2d_cross1\history3\result_images3，可以自行查看比较所有结果。）

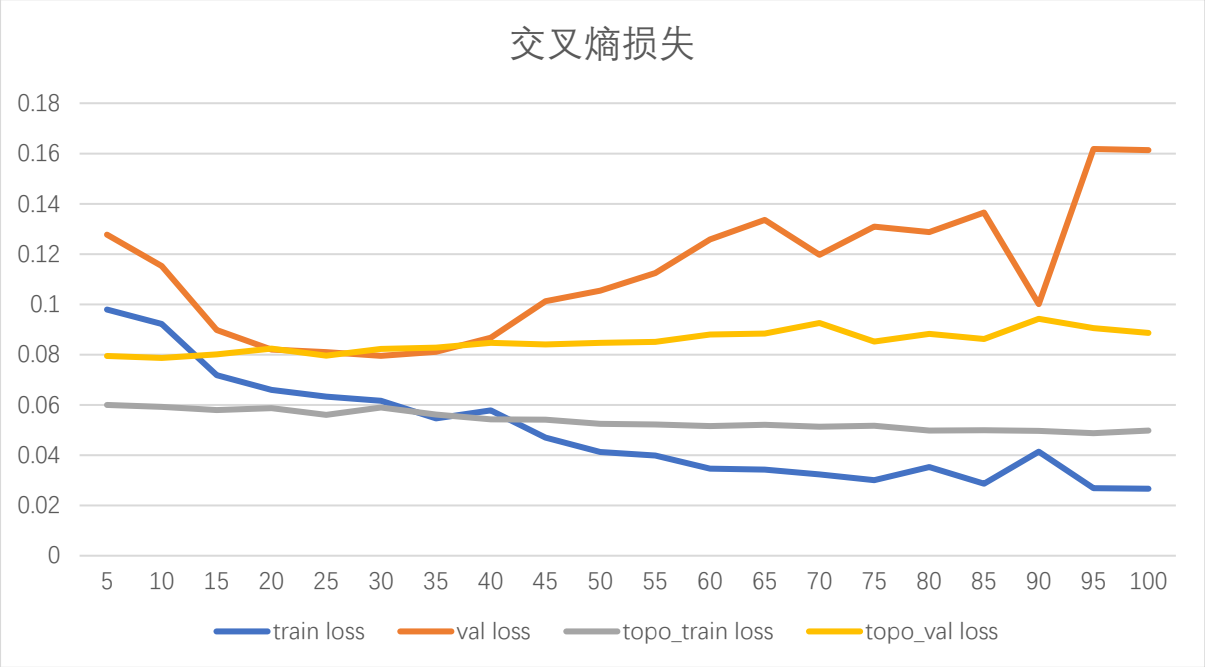
下表为 120 个 epoch 训练过程中训练、验证集的交叉熵损失和像素精度的记录：

epoch	train loss	train acc	val loss	val acc	topo_train loss	topo_train acc	topo_val loss	topo_val acc	topo_epoch
5	0.097928233	0.933369901	0.127682698	0.9040003	0.059980691	0.946757076	0.079454834	0.92418495	101
10	0.092161498	0.939142446	0.115260239	0.912941502	0.059253332	0.946639521	0.078673239	0.923748171	102
15	0.07187584	0.94236162	0.089837748	0.91778118	0.057878842	0.947571421	0.08003956	0.924741848	103
20	0.065912221	0.944227116	0.082064543	0.921098954	0.058763208	0.945796859	0.082420194	0.922467897	104
25	0.063331752	0.944959802	0.081026687	0.921718755	0.056021127	0.947987608	0.079526568	0.924839233	105
30	0.061693436	0.946747261	0.079510736	0.924465862	0.058934234	0.947094491	0.082258272	0.923930941	106
35	0.054635387	0.947884196	0.081070522	0.924370372	0.05613969	0.947750086	0.082786156	0.92409628	107
40	0.057852952	0.945968654	0.086733275	0.921518049	0.054223198	0.948356283	0.084696376	0.92427425	108
45	0.046950314	0.949711968	0.101261554	0.923158937	0.054120185	0.948245446	0.084102713	0.923853134	109
50	0.04131163	0.951328566	0.105391619	0.923089972	0.05246975	0.948711401	0.084735623	0.924487082	110
55	0.039794725	0.952036477	0.112415815	0.922865899	0.052177248	0.949067294	0.085087256	0.924537606	111
60	0.034592712	0.953437839	0.125755576	0.923217671	0.051623854	0.948957134	0.088002571	0.923896585	112
65	0.034208171	0.953377224	0.133549345	0.922807038	0.052131969	0.949319601	0.088327462	0.924239894	113
70	0.032378705	0.95406024	0.119738195	0.923828378	0.051371716	0.94899128	0.09254628	0.923732256	114
75	0.03006475	0.95472897	0.130860445	0.923446039	0.051647314	0.949082858	0.085158018	0.923982854	115
80	0.035210805	0.953273021	0.128776369	0.922727463	0.049798937	0.949479981	0.088264842	0.924161456	116
85	0.028631244	0.955217408	0.136559914	0.923405872	0.049960375	0.950155292	0.086168793	0.924458789	117
90	0.041414596	0.950659288	0.100127934	0.922421036	0.049619819	0.949789039	0.0942409	0.9237968	118
95	0.026924532	0.955626347	0.161795943	0.923381494	0.048746403	0.949964192	0.090584102	0.924417991	119
100	0.026653893	0.955884655	0.161401546	0.923272868	0.049851361	0.949674565	0.088677435	0.923947235	120

上表左半边为仅使用交叉熵损失训练的结果，每 5 个 epoch 记录一次；上表右半边为加入拓扑损失训练的结果，每 1 个 epoch 记录一次。

由上表作图如下：





由表与上两图易知，仅用交叉熵损失训练时，神经网络在验证集的像素精度和交叉熵损失都在 epoch30 达到最佳（像素精度越高越好，交叉熵损失越低越好），往后由过拟合趋势。

加入拓扑损失训练之后，验证集上的像素精度和交叉熵损失在较小范围内波动（由于使用的是 epoch30 时的模型继续 finetune，可以认为此时已经达到了局部最优，在定量指标上难有寸

进)，但是对验证集分割人工判别可以看出，相比于仅用交叉熵损失训练，分割结果的拓扑特征得到了良好保留。

加入拓扑损失之后的训练持续了 20 个 epoch，对比仅用交叉熵损失训练的 30-50epoch，可见训练集上的交叉熵损失和像素精度优化速度都较慢，可见加入拓扑损失之后，神经网络注意训练自身保留拓扑特征的能力，而不是一味提升像素精度。因此，加入拓扑损失也减缓了过拟合的趋势。

实验代码

请见附件，代码.7z。

参考资料

[1]: Wagner H., Chen C., Vuçini E. (2012) Efficient Computation of Persistent Homology for Cubical Data. In: Peikert R., Hauser H., Carr H., Fuchs R. (eds) Topological Methods in Data Analysis and Visualization II. Mathematics and Visualization. Springer, Berlin, Heidelberg.
https://doi.org/10.1007/978-3-642-23175-9_7

[2]: Volodymyr Mnih. Machine learning for aerial image labeling. University of Toronto (Canada), 2013.

[3]: Cubical complex reference manual
https://gudhi.inria.fr/python/latest/cubical_complex_ref.html

[4]: T. Kaczynski, K. Mischaikow, and M. Mrozek. Computational Homology. Applied Mathematical Sciences. Springer New York, 2004. ISBN 9780387408538.
https://books.google.fr/books?id=AShKtpi3GecC

[5]: Freedman D., Chen C.: Algebraic topology for computer vision, Computer Vision. Nova Science.
https://boole.cs.qc.cuny.edu/cchen/publications/chen_survey_2011.pdf

[6]: Math 529: Introduction to Computational Topology (Spring 2020)
http://www.math.wsu.edu/math/faculty/bkrishna/Math529.html