学号：18721802

数据分析
课程论文

基于机器学习对三维神经元图像的胞体检测

姓　　名　　李琦

学　　号　　18721802

论文评分

2019 年 3 月 18 日

目录

基于机器学习对三维神经元图像的胞体检测

[摘要]神经元的计算和分析中两个重要的任务是寻找神经元的胞体和重建它的结构。其中寻找胞体更为重要，因为我们需要它来展开图像追踪，从而重建结构。本文描述了一个基于机器学习的自动神经元胞体检测技术。测试图像是随机选择的包含噪声和部分神经元结构的原始图像，图像中的胞体也没有提前标注，实验表明该方法是非常高效的。

[关键字]神经元图像，胞体，机器学习

Abstract: Two important tasks in the calculation and analysis of neurons are to find the soma of the neuron and reconstruct its structure. It is more important to look for the soma because we need it to expand the image tracking to reconstruct the structure. This paper describes an automated neuron soma detection technique based on machine learning. The test image is a randomly selected original image containing noise and part of the neuron structure, and the soma in the image is not labeled in advance. Experiments show that the method is very efficient.

Key words: Images of Neurons; Soma; Machine learning

一 引言

神经系统是人体内起主导作用的功能调节系统。人体的结构与功能均极为复杂，体内各器官、系统的功能和各种生理过程都不是各自孤立地进行，而是在神经系统的直接或间接调节控制下，互相联系、相互影响、密切配合，使人体成为一个完整统一的有机体，实现和维持正常的生命活动。同时，人体又是生活在经常变化的环境中，神经系统能感受到外部环境的变化对体内各种功能不断进行迅速而完善的调整，使人体适应体内外环境的变化。可见，神经系统在人体生命活动中起着主导的调节作用，人类的神经系统高度发展，特别是大脑皮层不仅进化成为调节控制的最高中枢，而且进化成为能进行思维活动的器官。因此，人类不但能适应环境，还能认识和改造世界。

神经元是大脑中的重要神经细胞，也是构成神经系统结构和功能的基本单位。

神经神经元的胞体是神经元的重要组成部分。 神经递质和神经元肽都是在这里合成并运送到神经末梢。关于胞体的研究对神经元分割至关重要并且是神经系统研究中必不可少的一步。

之前已经出现了手动设置胞体检测参数的方法，来帮助专家们寻找胞体位置，以及基于目标样本的自动化方法。Pool[2]和 Ho[3]分别在 2008 年和 2011 年提出了手工设置参数来帮助生物学家确定胞体位置的方法。Pawley 在 2006 年提出直接将图像阈值来作为二进制掩码进行胞体检测的方法，彭汉川[7]在 2011 年提出将最大距离变换，2015 年 Zhou[10]提出了把强度和差异作为阈值，这些方法在分割后的荧光灯图像中都非常有效。此外 CoHen[1]，Lu[5]，Chothani[6]，Donohue[8]和 Yang[9]等人等人都在各自的论文中提出了一些有效的确认神经元形态的方法。

本体提出了一种针对果蝇神经元细胞的基于机器学习的全自动神经元胞体检测算法，相比较和之前他人方法的检测结果，本文的方法更加有效。

二 胞体自动化检测算法

2.1 数据集

本实验的数据库使用了 FlyCircuit 的图像数据库。每个图像都包含使用共聚焦显微镜获得的三维体数据。数据集被分为两部分，第一部分是未经处理的包含噪声和一些不需要的神经元结构的图像（如图 1a），第二部分是经过人为处理的完整的神经元图像（去除了噪声，只保留了干净的神经元信号，如图 1b 和 1c）。

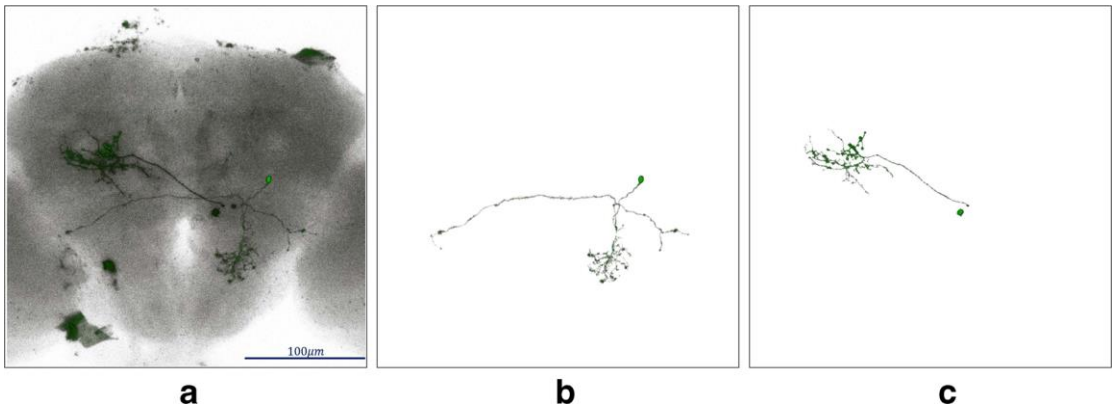


Figure 1

2.2 数据预处理

首先我们把数据按照阈值分别为整个图像强度的 60%，70%，80%，90%，95%来进行处理保留（如图 2），然后进行二值化处理（体素值为 0/1）。整个图像此时有 n 个连通分量，每个连通分量 C 都被一个 Bounding Box（B）所包围。D_x 和D_y， D_z分别为 Bounding Box 的 x,y,z 分量（如图三 b）。如果连通分量有超过一百个体素在 Bounding Box 边缘，那我们认为它是一个不完整的神经元（因为神经元可能正好在图像切割的边缘）。B_f值为图像到连通分量边缘的距离，我们设置的阈值为 5，当$B_f > 5$时，我们才认为它可能是 Soma（如图 3）。如果两个 Soma 距离小于 3 个体素，我们就把它们融合成一个 Soma。

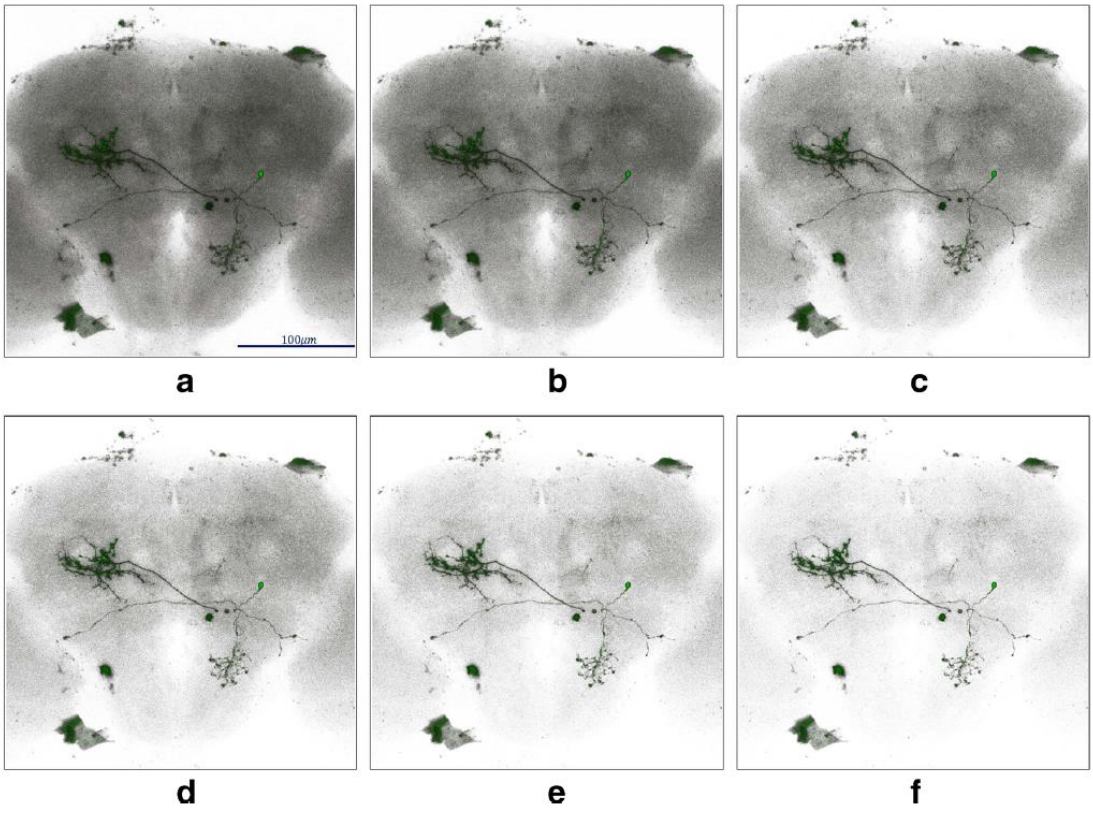


Figure 2

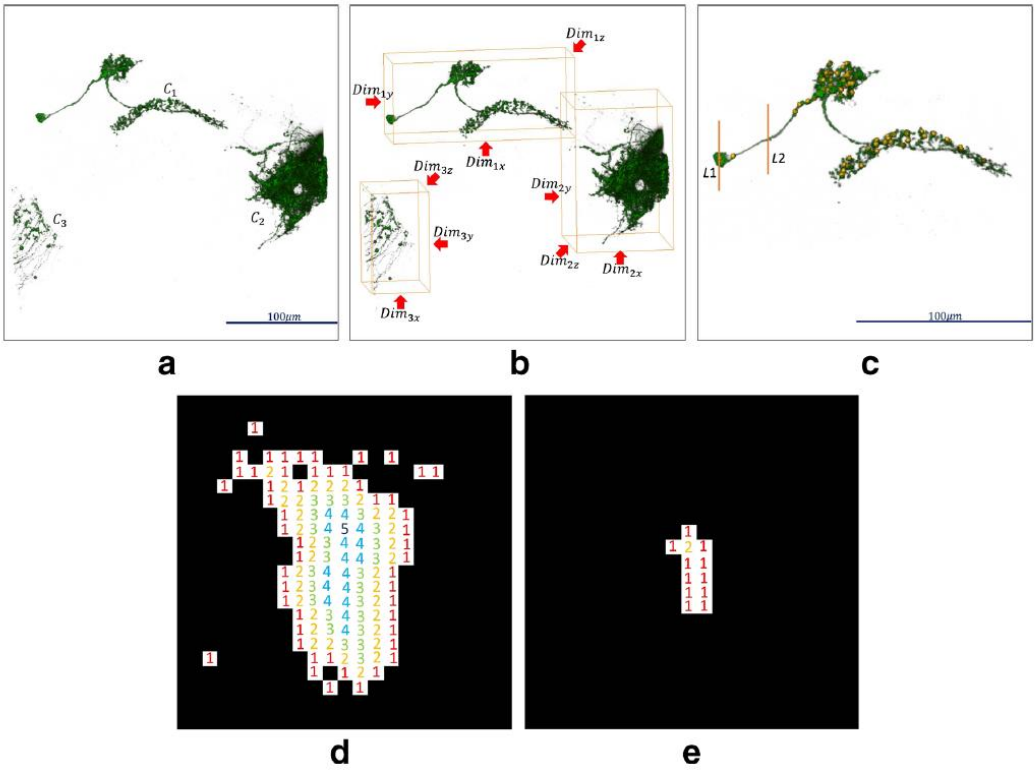


Figure 3

2.3 胞体特征

胞体候选者需满足以下特征才可能是一个神经元的中心点。

让U_i和M_i分别为第 i 个连通分量体素的平均值和最大值，S_{ij}为第 i 个连通分量的第 j 个 Soma 候选者，其 Bf 值为BF_{ij}。我们提出了以下三个特征，这些特征均被归一化，其值越接近 1，表明为 Soma 的可能性越大。

Soma 比神经元其他部分具有更高的强度值。归一化强度值U_i是以候选者为中心的立方体的平均强度值。

$$\mu_{ij} = \frac{1}{27 \cdot M_i} \sum_{k=1}^{27} I(v_k),\ v_k \in 3\times3\times3 \text{ box centered at } S_{ij},$$

Soma 除了体素值外，其外形更接近于球体，并且体素值更加紧凑。我们定义紧凑值为候选者周围的体素超过平均值的数量

$$\rho_{ij} = \frac{1}{27}\sum_{k=1}^{27} den(v_k),\ v_k \in 3\times3\times3 \text{ region centered at } S_{ij},$$

$$den(x_k) = 0, \text{ if } I(v_k) < \mu_i,$$

$$den(x_k) = 1, \text{otherwise}$$

Soma 的位置通常也是一个特征，其一般情况不会在 Bounding Box 的中心位置，所以我们可以定义一个特征，x，y，z 分别是候选者在三个维度到 Bounding Box 中心点的位置

$$d_{ij} = 2 \cdot \max\{\frac{\overline{x}}{Dim_{ix}}, \frac{\overline{y}}{Dim_{iy}}, \frac{\overline{z}}{Dim_{iz}}\}.$$

Soma 的外形类似于球体，所以我们可以为之确立一个均衡性特征。

设置一个平面，穿过候选者体素，左边点个数为 f，右边为 r，则均衡性计算方法为

$$1-(f/r)$$

（若 r<f，则 r 为分子）

2.4 机器学习算法检测胞体

我们从清理过的数据集中选择训练样本，其中包括正向和反向样本。我们选择连通分量中具有最大 Bf 值的体素，如果其和真实位置相差小于 6 微米，那么其为正向样本，否则为反向样本。我们重复五次操作，可以获得 5 个例子

上节我们提到了四个特征，让 Xi 分别代表这四个特征，我们可以创建出如下的回归方程学习模型：

$$y(\boldsymbol{w}, \boldsymbol{x}^k) = w_0 + \sum_{i=1}^{4} w_i x_i^k + \sum_{i=1}^{4}\sum_{j=1}^{4} w_{ij} x_i^k x_j^k,$$

其中 w 为权重，k 代表第 k 个训练实例，y 函数结果为 0/1，表明其是否为一个 Soma。应用逻辑回归，将连续目标转换为分类目标，h 函数定义如下：

$$h_w(x) = \frac{1}{1+e^{-y(w,x)}}.$$

针对 h 函数，需要一个阈值来确立输出：

$$\text{Output} = \begin{cases} 0 \text{ (not soma)} , \text{if } (h_w(x)) < th, \\ 1 \text{ (soma)} , \text{if } (h_w(x)) \geq th. \end{cases}$$

本文中设 th 为 0.5

我们的目标是计算权重 w，来使之匹配 y 输出，使得判断结果为最佳。

以下提出两个函数，代价函数和误差函数，每次迭代计算之后，结果都需要减去一个误差函数。

代价函数：

$$\text{Cost}(h_w(x), y) = \begin{cases} -\log h_w(x) , \text{ if } y = 1, \\ -\log (1 - h_w(x)) , \text{ if } y = 0. \end{cases}$$

误差函数：

$$E(\boldsymbol{w}) = \frac{1}{m} \sum_{k=1}^{m} \text{Cost}(h_w(x^k), y^k) + \frac{\lambda}{2m} \sum_{j=1}^{n} w_j^2,$$

其中 m 是训练实例数，n 是权重个数。等式右侧第二个表达式是为了避免过度拟合。误差函数是为了累积对目标预测的误差，我们的目标是找到 w 的集合，使得代价函数最小化，可以通过求解误差函数的偏导来实现。

$$\frac{\partial}{\partial w_j} E(\boldsymbol{w}) = \frac{1}{m} \sum_{k=1}^{m} \frac{\partial}{\partial w_j} \text{Cost}(h_w(x^k), y^k) + \frac{\lambda}{m} w_j = 0.$$

联立之前等式最终可以得到 w 的更新式：

$$w_j^{i+1} = w_j^i - \alpha \frac{\partial}{\partial w_j} E(\boldsymbol{W}^i),$$

.

α = 0.1，为学习率，学习的停止条件为误差函数提高小于 0.0001 或者迭代次数大于 5000。最终可以获得方程中的权重。

三 实验结果

从 FlyCircuit 数据库中随机选择 200 个原始图像，由人手工提前标注。（但有些 Soma 没有被人标注，也可能被算法检测出来）

为了分析我们算法的效果，我们定义了几个变量：

TP:220 个中选择正确的 Soma 个数

TS:220 个之外选择正确的 Soma 个数

FN：220 个中选择没有被检测出的 Soma 个数

FP：选择错误的 Soma 个数

我们使用灵敏度，F1 分数和准确度来估算算法表现：

$$\text{Sensitivity (Recall)} = \frac{TP}{TP + FN}.$$

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

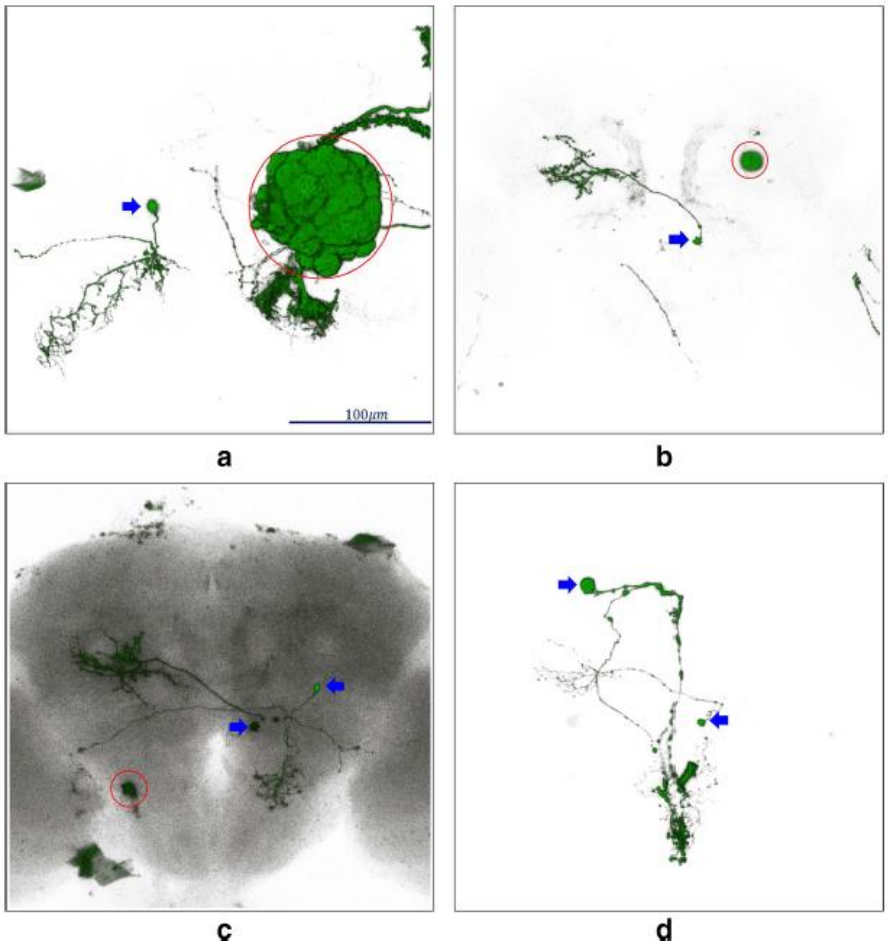
$$\text{F1 score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

因为图像中包含的 Soma 并没有提前被标注出来，而且数量未知，所以灵敏度和 F1 分数没有包含 TS。而又因为算法识别出了不在数据库中的 Soma，所以我们将精度测量值修改为：

$$\text{Accuracy} = \frac{TP + TS}{TP + TS + FP}.$$

灵敏度和准确度取决于方程式中的阈值 th。如果只有一个 soma，则将 th 设置为最高 hw（x）。 在这种情况下，我们得到 TP = 156，T S = 20，FP = 21，FN = 64。敏感度为 70.9％，F1 评分为 78.6％，准确率为 89.3％。 考虑到图像中可能存在多个 soma，th 被设置为 0.5 作为 hw（x）的阈值。 然后我们得到 TP = 195，T S = 55，FP = 57，FN = 25.灵敏度为 88.6％，F1 得分为 82.6％，准确度为 81.4％。

下图展示了大多数情况下的四个疑难案例体细胞检测算法。 图 7a-c 包含明亮的物体可导致体细胞检测到错误的位置。



本文还调用了其他的方法针对上图进行检测，包括 APP2，Rivulet，NIA。

对比结果如下

Methods	TP	FP	Sensitivity	Accuracy	F1 score
APP2	16	14	0.432	0.533	0.477
Rivulet	21	13	0.567	0.617	0.591
NIA	21	32	0.567	0.407	0.466
Our method *th*=Highest	24	3	0.649	0.906	0.750
Our method *th*=0.5	34	7	0.918	0.844	0.872

Table 1

除去比较特殊的情况，我们随机选择 200 个测试数据集中的 33 个体积数据集。 然后我们将方法 APP2，Rivulet 和 NIA 应用于这组测试数据。表 1 列出了所有的方法。本文提出的方法显然更加有效。

四 结论

我们描述了一种机器学习方法来检测果蝇大脑的共聚焦显微图像上的体细

胞。所提出的方法是有效的并且可以应用于原始原始图像，其中噪声未被去除，并且胞体的数量是未知的。根据不同的显微镜，为获取图像，操作员和图像特征而设置的参数可能不同。为了获得最佳结果，可以更改功能，并且应该重新训练模型。

在这项研究中，一旦机器训练完成，胞体识别任务就会完全自动执行。 随机选择 200 个测试数据集，灵敏度和准确度均令人满意。 该算法具有鲁棒性和实用性。

参考文献

[1] [Cohen, A.R., Roysam, B., & Turner, J.N. (1994). Automated tracing and volume measurements of neurons from 3-d confocal fluorescence microscopy data. Journal of Microscopy, 173(Pt2), 103–114.

[2] Pool, M., Thiemann, J., Bar-Or, A., & Fournier, A.E. (2008). Neuritetracer: a novel imagej plugin for automated quantification of neurite outgrowth. Journal of Neuroscience Methods, 168(1), 134–139.

[3] Ho, S.Y., Chao, C.Y., Huang, H.L., Chiu, T.W., Charoenkwan, P., & Hwang, E. (2011). Neurphologyj: an automatic neuronal morphology quantification method and its application in pharmacological discovery. BMC Bioinformatics.

[4] Pawley, J.B. (2006). Handbook of biological confocal microscopy. New York: Springer.

[5] Lu, J., Fiala, J.C., & Lichtman, J.W. (2009). Semi-automated reconstruction of neural processes from large numbers of fluorescence images. PLoS One, 4, 9e5655.t.

[6] Chothani, P., Mehta, V., & Stepanyants, A. (2011). Automated tracing of neurites from light microscopy stacks of images. Neuroinform, 9, 263–278.

[7] Peng, H., Meijering, E., & Ascoli, G.A. (2015b). From diadem to bigneuron. Neuroinformatics, 13(3), 259–260.

[8] Donohue, D.E., & Ascoli, G.A. (2011). Automated reconstruction of neuronal morphology: an overview. Brain Research Reviews, 67, 94–102.

[9] Yang, J., Gonzalez-Bellido, P.T., & Peng, H. (2013). A distance-field based automatic neuron tracing method. BMC Bioinformatics, 14(1), 93.

[10] Zhou, Z., Sorensen, S., Zeng, H., Hawrylycz, M., & Peng, H. (2015). Adaptive image enhancement for tracing 3d morphologies of neurons and brain vasculatures. Neuroinform, 13,153–166

附录

主函数代码

```
int main(int argc, char* argv[])
{
    // you can use your weights
    weight[0] = -2.43992;
    weight[1] = 0.346942;
    weight[2] = 0.303959;
    weight[3] = -1.81573;
    weight[4] = 0.307872;
    weight[5] = 1.71534;
    weight[6] = 0.705672;
    weight[7] = 1.0054;
    weight[8] = 0.486977;
    weight[9] = 1.27172;
    weight[10] = 0.197391;
    weight[11] = 0.321255;
    weight[12] = -1.56598;
    weight[13] = 0.173972;
    weight[14] = 0.385298;

    iSoma_update = new struct_Voxel[1000000];
    iSoma_ALL = new struct_Voxel[1000000];
    iSoma_ALL_attr = new struct_Attr[1000000];

    int raw_voxels = 0;
    //int icount_update = 0;                    //control candidate to attributes
    sFileName = string(argv[1]);
```

```
std::ifstream fin_in;
fin_in.open(sFileName);
int id;
char tmp_s[50];
while (fin_in >> sFileName)
{
    if (strcmp(sFileName.c_str(), "") != 0)
    {
        clock_t start_time = clock();
        change_filename();
        AM_all_data.Load_AM(sFileName, 1);
        int yy = AM_all_data.idim_y / 4 * 4;
        int xx = AM_all_data.idim_x / 4 * 4;
        int** z_plane = new int*[yy];
        for (int i = 0; i < yy; i++)        z_plane[i] = new int[xx];
        for (int i = 0; i < yy; i++)
        for (int j = 0; j < xx; j++)
            z_plane[i][j] = 0;
        for (int i = 0; i < AM_all_data.idim_z; i++)
        for (int j = 0; j < yy; j++)
        for (int k = 0; k < xx; k++)
        if (z_plane[j][k] < AM_all_data.Data_Ori[i][j][k]) z_plane[j][k] =
AM_all_data.Data_Ori[i][j][k];
        int hisgram[4096] = { 0 };
        for (int z = 0; z < AM_all_data.idim_z; z++)
        for (int y = 0; y < AM_all_data.idim_y; y++)
        for (int x = 0; x < AM_all_data.idim_x; x++)
        {
            hisgram[(int)AM_all_data.Data[z][y][x].value]++;
```

```
}
std::ofstream foutattr(sFileName_attr.c_str());
//cout << AM_all_data.iVoxels << endl;
double sum = 0;
int c = 1;
for (c = 1; c < 4096; c++)
{
    sum += hisgram[c];
    if (sum / AM_all_data.iVoxels > TH*0.01 ) break;
}
bool stop = false;
for (int dotime = 0; dotime < 1 && !stop; dotime++)
{
    cout << "run " << dotime + 1 << endl;
    cout << "threshold is " << TH    << endl;
    int components;
    int bound_count = 0;
    components = AM_all_data.Find_Component(500);
    icount_update = 0;
    iSoma_ALL_index = 0;
    do_candidate(components);
    do_attributes(components, raw_voxels);
    double judge = 0;
    double x_train[4] = { 0 };
    cout <<" total " << iSoma_ALL_index << " candidates" << endl;
    foutattr <<" total " << iSoma_ALL_index << " candidates" <<
endl;
    int first_time = 0;
    double max_judge = 0;
    while (first_time < 2)
```

```
{
    first_time++;
    icount_update = 0;
    for (int i = 0; i < iSoma_ALL_index; i++)
    {
        x_train[0] = iSoma_ALL_attr[i].attr1;
        x_train[1] = iSoma_ALL_attr[i].attr2;
        x_train[2] = iSoma_ALL_attr[i].attr3;
        x_train[3] = iSoma_ALL_attr[i].attr4;
        judge = weight[0] + weight[1] * x_train[0] + weight[2] *
x_train[1] + weight[3] * x_train[2] + weight[4] * x_train[3];
        judge += weight[5] * x_train[0] * x_train[0] + 2 *
weight[6] * x_train[0] * x_train[1] + 2 * weight[7] * x_train[0] * x_train[2];
        judge += 2 * weight[8] * x_train[0] * x_train[3] +
weight[9] * x_train[1] * x_train[1] + 2 * weight[10] * x_train[1] * x_train[2];
        judge += 2 * weight[11] * x_train[1] * x_train[3] +
weight[12] * x_train[2] * x_train[2] + 2 * weight[13] * x_train[2] * x_train[3];
        judge += weight[14] * x_train[3] * x_train[3];
        judge = 1 / (1 + pow(exp(1), -1 * judge));
        if (judge > max_judge && x_train[0]>0.3 )    max_judge
= judge;
        if (first_time == 1)
            foutattr << iSoma_ALL_attr[i].attr1 << "\t" <<
iSoma_ALL_attr[i].attr2 << "\t" << iSoma_ALL_attr[i].attr3 << "\t" <<
iSoma_ALL_attr[i].attr4 << "\t" << iSoma_ALL[i].x << "\t" << iSoma_ALL[i].y << "\t" <<
iSoma_ALL[i].z << "\t" << iSoma_ALL[i].value << "\t" << judge << "\t" <<
iSoma_ALL[i].bf_value << std::endl;
        if (first_time == 2 && judge == max_judge)
        {
            //cout << icount_update << endl;
```

```
                iSoma_update[icount_update] = iSoma_ALL[i];
                icount_update++;
            }
        }
    }
    //cout << "icount_update is ... " << icount_update << endl;
    change_filename(9);
    std::ofstream fsoma(sFileName_output.c_str());
    if (icount_update == 0)
    {
        fsoma << sFileName << "\r";
        fsoma << 0 << "\r";
        fsoma.close();
    }
    else
    {
        //cout << "start checking repeats" << endl;
        int total_soma = 0;
        struct_Voxel tmp_maxsoma;
        struct_Voxel *repeats;
        repeats = new struct_Voxel[icount_update];
        for (int i = 0; i < icount_update; i++){
            int repeat_index = 0;
            if (iSoma_update[i].x != -1)
            {
                repeats[repeat_index++] = iSoma_update[i];
                iSoma_update[i].x = -1;
                for (int j = i + 1; j < icount_update; j++)
                {
                    if (iSoma_update[j].x != -1)
```

```
                                        {
            for (int k = 0; k < repeat_index; k++)
                                {
                                    if ((abs(iSoma_update[j].x - repeats[k].x) < 6) &&
(abs(iSoma_update[j].y - repeats[k].y) < 6) && (abs(iSoma_update[j].z - repeats[k].z)
< 6))
                                                {

                                                    repeats[repeat_index++] =
iSoma_update[j];
                                                    iSoma_update[j].x = -1;
                                                    break;
                                                }

                                            }
                                        }


                                    }
                                    tmp_maxsoma = repeats[0];

                                    for (int k = 1; k < repeat_index; k++)
                                    {

                                        if (repeats[k].connect_flag >
tmp_maxsoma.connect_flag)
                                            tmp_maxsoma = repeats[k];

                                    }
```

```
                                    //if (tmp_maxsoma.x > 6 && tmp_maxsoma.y >
6 && tmp_maxsoma.z > 3 && tmp_maxsoma.x < 249 && tmp_maxsoma.y < 249 &&
tmp_maxsoma.z < AM_all_data.idim_z - 3)
                                        iSoma_update[total_soma++]          =
tmp_maxsoma;
                                    //cout << total_soma << endl;
                                }
                            }
                            //    cout << "done repeats" << endl;
                            delete[] repeats;
                            icount_update = total_soma;
                            //          cout << "total " << icount_update << endl;
                            fsoma << sFileName << "\n" << icount_update << "\n";

                            for (int i = 0; i < icount_update; i++)
                                fsoma << iSoma_update[i].x    << " " <<
iSoma_update[i].y    << " " << iSoma_update[i].z << " " << iSoma_update[i].value <<
"\n";
                            fsoma.close()
                            int ysize = AM_all_data.idim_y;
                            int xsize = AM_all_data.idim_x;
                            if (xsize % 4 == 1)xsize = xsize + 3;
                            else if (xsize % 4 == 2)xsize = xsize + 2;
                            else if (xsize % 4 == 3)xsize = xsize + 1;
                            if (ysize % 4 == 1)ysize = ysize + 3;
                            else if (ysize % 4 == 2)ysize = ysize + 2;
                            else if (ysize % 4 == 3)ysize = ysize + 1;

                            int file_size = ysize * xsize * 3 + 54;
                            header[2] = (unsigned char)(file_size & 0x000000ff);
```

```
header[3] = (file_size >> 8) & 0x000000ff;
header[4] = (file_size >> 16) & 0x000000ff;
header[5] = (file_size >> 24) & 0x000000ff;

int width = xsize;
header[18] = width & 0x000000ff;
header[19] = (width >> 8) & 0x000000ff;
header[20] = (width >> 16) & 0x000000ff;
header[21] = (width >> 24) & 0x000000ff;

int height = ysize;
header[22] = height & 0x000000ff;
header[23] = (height >> 8) & 0x000000ff;
header[24] = (height >> 16) & 0x000000ff;
header[25] = (height >> 24) & 0x000000ff;

std::ofstream file_z(sFileName_bmp_z.c_str(),
std::ofstream::binary);

char pixel = 0;
double pp = 0;
file_z.write(header, 54);
// write body
//for (int i = 0; i < ysize; i++) {
for (int i = ysize - 1; i >= 0; i--) {
    for (int j = 0; j < xsize; j++) {
        if (j < AM_all_data.idim_x && i <
AM_all_data.idim_y)
        {
            bool find = false;
```

```
                                        for (int c = 0; c < icount_update; c++)
                                        {
                                            if ((abs((float)(i - iSoma_update[c].y))
+ abs((float)(j - iSoma_update[c].x))) < 2)

    //if(abs((float)(i-iSoma_ALL[c].y*4+2))<3 &&
abs((float)(j-iSoma_ALL[c].x*4+2))<3 )
                                            {
                                                find = true;
                                                break;
                                            }
                                        }
                                        if (find)
                                            file_z.put(0).put(0).put(255);
                                        else
                                        {
                                            if (j < xx && i < yy)
                                            {
                                                pp = z_plane[i][j];
                                                pixel= pp/AM_all_data.iIty_Max*
255;

    file_z.put(pixel).put(pixel).put(pixel);
                                            }
                                            else{
                                                pixel = 0;

    file_z.put(pixel).put(pixel).put(pixel);
                                            }
                                        }
                                    1
```

```
                                        }
                                        else
                                        {
                                            pixel = 0;
                                            file_z.put(pixel).put(pixel).put(pixel);
                                        }
                                    }
                                }
                                file_z.close();
                                std::cout << "done pic" << endl;
                                stop = true;
                                //    break; //break to    dotime
                            }
                                //end for th
                        } //end for do times
                        clock_t end_time = clock();
                        std::cout << "total time is " << difftime(end_time, start_time) /
CLOCKS_PER_SEC << std::endl;
                        foutattr.close();
                        for (int i = 0; i < yy; i++)    delete[] z_plane[i];
                        delete[] z_plane;
                    }
                }     //End all files
                delete[] iSoma_update;
                delete[] iSoma_ALL;
                delete[] iSoma_ALL_attr;
            }
            }
```