数学软件课程报告

王家蔚

2020 年 7 月 16 日

摘要

这篇报告分成两个部分，第一部分是一个数据图像处理的模型，这个项目输入的是一张人像照片，输出的是人像的面部，左右眼，嘴和鼻子这几个部位的位置，并且用不同颜色把它框出来。第二部分我做的是曲面及其各处法向量的图像结果，输入的是三个等长数组，对应离散的三维点坐标，输出的是拟合出来的曲面和坐标点处的法向量。

1 模型：人脸器官识别

1.1 程序结构

图 1: 程序运行流程图

简单说明一下程序设计的思路。[1] 首先输入是关于想要输出的检测框大小相关的参数，比如检测阈值和边框粗细等，这些参数通过 buildDetect 这个函数，转换成存储检测框信息的数据结构。这个结构再加上一张要判断的人像图片，两个数据一起输入分析五官的函数里。如果人像没有旋转，那么直接用的是 DetectFaceParts 函数，然后输出结果；如果人像是旋转过的，那么就要用 DetectRotFaceParts 这个函数，运行过程中它还会调用另外一个函数 MergeFourPoints, 把定位出来的四个关键位置点存放在一起，从而找到正确的方向。

位置	输入	位置	输出
1	图片	1	面部位置
2	最小搜索尺寸	2	鼻子位置
3	面部阈值	3	眼睛位置
4	五官阈值	4	嘴巴位置

接下来详细地介绍以下具体的实现步骤。这个程序需要用到 matlab 里的 Image Process Package 和 Computer Vision Package，其中最核心的部分是 System Object method 里的 step 函数和 vision.CascadeObjectDetector 函数，后者使用的是 Viola-Jones 算法 [2]。在旋转图像的过程中，用到的主要是 maketform, 得到新的图像之后再调用之前的 DetectFaceParts, 相当于是把图像转回去再处理。在设置参数的时候，旋转而来的图像需要设置更大的边界阈值来应对不规则的变形。

1.2 图像结果

为了更加客观全面地展示算法效果，我找了各国男女人像照片来进行测试，发现效果总体来说非常不错, 但在局部也存在一些问题。

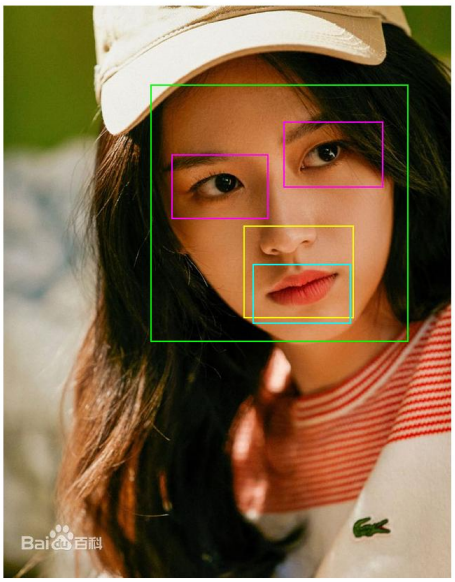

图 2: 识别中国女性正面

比如这张照片的输出结果，显然选取鼻子的框过大，甚至包括了嘴部，但是其他部分还是非常准确的，尽管左眼有部分被头发挡住，发际线也被帽子遮住，也依然没有出现不可接受的偏差。

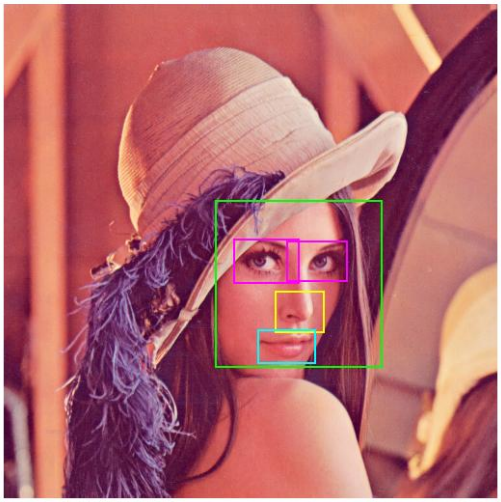

图 3: 识别外国女性侧面

再用外国女像进行测试, 尽管这张照片的角度更大，图像也不是很清晰，脸部占比小，但是效果更好, 尤其是鼻子和嘴的重合部分明显变小了。之后的几张图片各有特点，算法都表现出极强的适应能力。

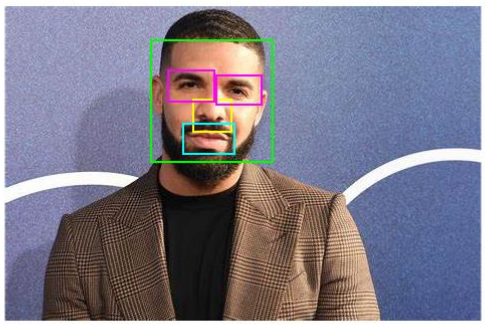

图 4: 识别外国男性正面

这张照片是从正面拍摄的，而且面部的胡子很多很浓，可以发现依然存在鼻子区域过大的问题，可能是嘴唇部分的胡子影响了正常的判定。接下来测试一下有一只眼睛完全被遮住的黑人女性正面照，探究以下此时是否还能够正常识别，进而更深入地探究以下算法的识别的原理。

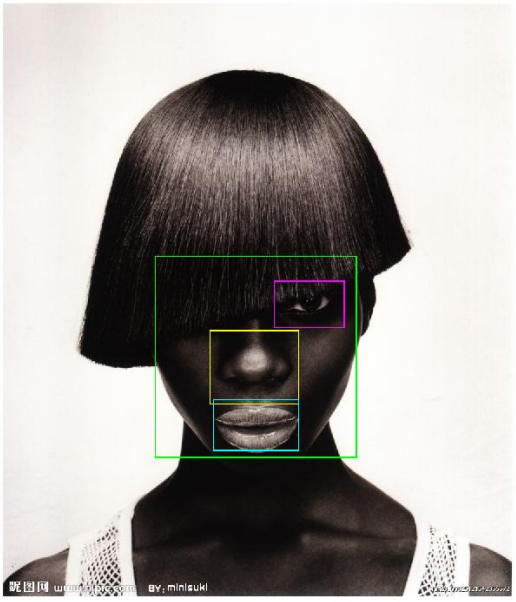

图 5: 识别面部部分被遮盖

从结果上看，四个框的相对位置虽然可以辅助选择正确的区域，但并不是有了三个位置后就一定得到了第四个位置，matlab 的面部识别每个部分是单独训练出来的，可以独立完成任务。

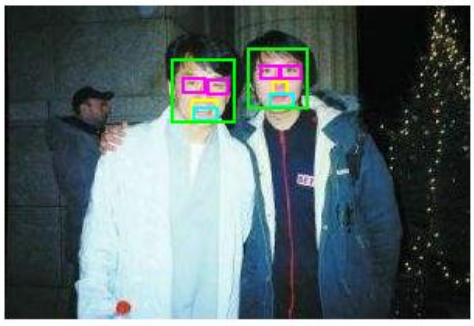

图 6: 识别模糊的合照

这张图片里有三个人脸，但是由于我的面部识别阈值设置的稍微大了一点，所以把最后面的人忽略了，只显示出前面两个人的正面五官。尽管照片

模糊，即使肉眼也很难看清，但还是精确地得到了结果。

1.3 部分源代码

Listing 1: **mergeFourPoints.m**

```
% mergeFourPoints
%Input parameter:
% src: fourpoints data to be merged
%Output parameter:
% dst: merged fourpoints data
 function dst = mergeFourPoints(src)
     if(  size(src,1) > 0 )
        pos =  zeros( size(src,1),2);
        pos(:,1) =  mean(src(:,[1,3,5,7]),2);
        pos(:,2) =  mean(src(:,[2,4,6,8]),2);
        pos = horzcat(pos, [1: size(src,1)]');

         while( ~ isempty(pos) )
             if(  size(pos,1) == 1 )
                dst = vertcat(dst,src(pos(1,3),:)
                    );
                pos = [];
             else
                tmp = src(pos(1,3),:);
                rad = 0;

                 for i = 1:4
                   rad =  norm(pos(1,1:2)-tmp
                        (1,2*i-1:2*i));
                 end
                th = rad / 8; p = 1;
                 for i=2: size(pos,1)
                    rad =  norm(pos(1,1:2) - pos(
                         i,1:2));
                     if( rad < th )
                        tmp = vertcat(tmp,src(pos
                             (i,3),:));
                        p = horzcat(p,i);
                     end
                 end
                num = tmp > 0;
                tmp =  sum(tmp) ./  sum(num);
                dst = vertcat(dst,tmp);
                pos(p,:) = [];
             end
         end
     else
        dst = src;
     end
 end
```

我对这段代码改动最多，主要是重写了大量变量赋值和运算的操作，使之符合向量化和矩阵化的操作习惯，而且可以多用循环来简化代码，使它逻辑更加清晰明了。由于文件比较多，囿于篇幅不在此过多展示。

2 模型：曲面及法向量可视化

2.1 程序结构

这个项目的程序结构比较简单 [3]，首先定义三个等长数组，输入 FindPointNormal 函数，这个函数的核心是 kd-tree 算法，具体操作是先调用 KDTreesearcher 函数，使用欧几里得度量

$$d = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

然后用 knnsearch 函数计算出相近的点。接下来，计算出协方差矩阵，为之后计算法向量和曲率作准备。最后一步，先初始化两个矩阵，一个存储曲率，另一个存储法向量，依次遍历协方差矩阵的每一个维度，根据公式

$$cov(\sum_{i=1}^{n} X_i, \sum_{j=1}^{m} Y_j) = \sum_{i=1}^{n}\sum_{j=1}^{m} cov(X_i, Y_j)$$

$$var(\sum_{i=1}^{n} X_i) = \sum_{i=1}^{n} var(X_i) + 2\sum_{i,j:i<j} cov(X_i, X_j)$$

计算出这个维度下的协方差矩阵，然后得到这个矩阵的特征值和特征向量并存储。

位置	输入	位置	输出
1	三维点集	1	法向量
2	邻近点数量	2	曲率
3	视角位置		

为了给输出的曲面图像上色，我使用了 reshape 函数来把以数组形式输出的曲率变成一个方阵。我采用了 quiver3 函数来将向量可视化输出，并标上鲜艳的颜色。

2.2 图像结果

首先我们来看一个经典的例子

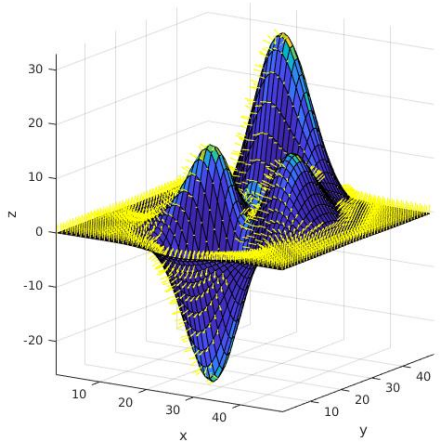

图 7: peak 函数乘以常数倍

在 demo.m 文件里自定义输入的三个在最后展示图像的环节，还需要让法向量跟随视角正确显示出来，所以 FindPointNormal 函数的最后一个部分这就解决这个需求的。再看一个例子，

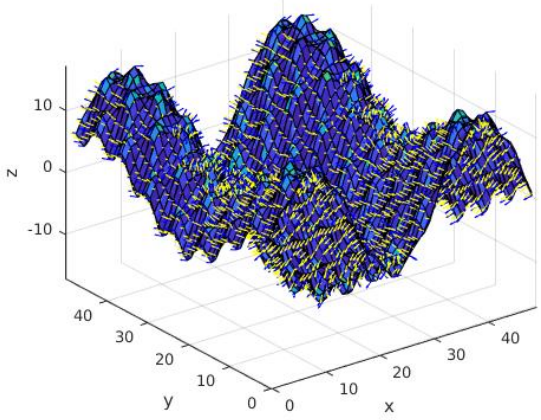

图 8: 用不同周期的正余弦构造出的函数

具体来说，这里用来生成图像的函数是

$$z = 10\sin\frac{x}{5} + 5\cos\frac{y}{7} + \sin x\cos y$$

沿着 x 轴和 y 轴每一个方向都有周期性。

2.3 部分源代码

这段代码是程序的核心，我针对它做了很多性能上地优化，并且已经以注释的形式，把每一步想要求得的量标注了出来，在开头出给出了整体的输入和输出。值得注意的是，这里有一个输入量是可选的，而前面一个模型的每一个输入都是必须的。

Listing 2: **FindPointNormal/FindPointNormal.m**

```
%  Required Inputs:
%   points- nx3 set of 3d points (x,y,z)
%  Optional Inputs: (will give default values on
    empty array [])
%  Outputs:
%   normals- nx3 set of normals (nx,ny,nz)
%   curvature- nx1 set giving the curvature

 function [ normals, curvature ] =
    FindPointNormal(points, numNeighbours,
    viewPoint, ~)
%ensure inputs of correct type
points = double(points);
viewPoint = double(viewPoint);
%create kdtree
kdtreeobj = KDTreeSearcher(points,'distance','
    euclidean');
%get nearest neighbours
n = knnsearch(kdtreeobj,points,'k',(numNeighbours
    +1));
%remove self
n = n(:,2: end);
%find difference in position from neighbouring
    points
p = repmat(points(:,1:3),numNeighbours,1) -
    points(n(:),1:3);
p =  reshape(p,  size(points,1),numNeighbours,3);
%calculate values for covariance matrix
C =  zeros( size(points,1),6);
C(:,1) =  sum(p(:,:,1).*p(:,:,1),2);
C(:,2) =  sum(p(:,:,1).*p(:,:,2),2);
C(:,3) =  sum(p(:,:,1).*p(:,:,3),2);
C(:,4) =  sum(p(:,:,2).*p(:,:,2),2);
C(:,5) =  sum(p(:,:,2).*p(:,:,3),2);
C(:,6) =  sum(p(:,:,3).*p(:,:,3),2);
C = C ./ numNeighbours;
% normals and curvature calculation
normals =  zeros( size(points));
```

```
curvature =  zeros( size(points,1),1);
 for i = 1:( size(points,1))
    %form covariance matrix
    Cmat = [C(i,1) C(i,2) C(i,3);...
        C(i,2) C(i,4) C(i,5);...
        C(i,3) C(i,5) C(i,6)];
    %get eigen values and vectors
    [v,d] =  eig(Cmat);
    d =  diag(d);
    [lambda,k] =  min(d);
    %store normals
    normals(i,:) = v(:,k)';
    %store curvature
    curvature(i) = lambda /  sum(d);
 end

% flipping normals
points = points-repmat(viewPoint,
                 size(points,1),1);
 if(dirLagest)
    [~,idx] =  max( abs(normals),[],2);
    idx = (1: size(normals,1))'+
         (idx-1)* size(normals,1);
     dir = normals(idx).*points(idx)>0;
 else
     dir =  sum(normals.*points,2)>0;
 end
normals( dir,:) = -normals( dir,:);
 end
```

3 总结

最后来总结一下这篇论文里我做的两个工作的，主要是想谈谈其中涉及的算法。第一个模型的核心是 matlab 封装好的 step 函数，它使用 System Object 实现的，经过仔细研究之后，发现它是由四个模块前后衔接而成，每个模块不仅针对图像，而且针对音频和文本都有对应的参数可以调整，值得进一步仔细研究。

第二个模型是一个几何问题，比较适合在研究比较初等的解析几何问题时，能用这程序更直观地理解正在计算的图形究竟长成什么样，帮助发现一些不太容易从解析表达式中看出来的性质。它是一个典型的利用最临近节点算法的例子，主要工作分成三步，首先是计算出样本点到其他所有点的距离，然后按照距离从近到远排序，并且选出前 k 个，最后少数服从多数，将这个点归并到选出的 k 个点中，最多的那一类里。

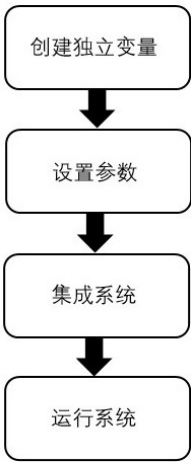

图 9: System Object 运行流程

针对优化 KNN 算法本身，查阅资料后发现还可以尝试用不同的度量空间进行聚类操作，此外还可以不完全按照少数服从多数的原则，从全局角度来，用概率估计的方式来进行分类判定。这个算法复杂度阶数太高，对于大数据输入极度以来计算能力，可以通过改进 KDTree 来提高运行效率。

参考文献

[1] Face Parts Detection, Masayuki Tanaka

[2] Rapid Object Detection using a Boosted Cascade of Simple Features, Viola, Paul and Michael J. Jones

[3] Find 3D Normals and Curvature, Zachary Taylor

[4] Multi-modal sensor calibration using a gradient orientation measure，Zachary Taylor