# Image Segmentation - CVDL hw2

1600012785 朱文韬

## 算法概述

图像分割任务通常包括传统方法和深度学习的方法，传统方法又有基于聚类的方法和基于割集的算法等。

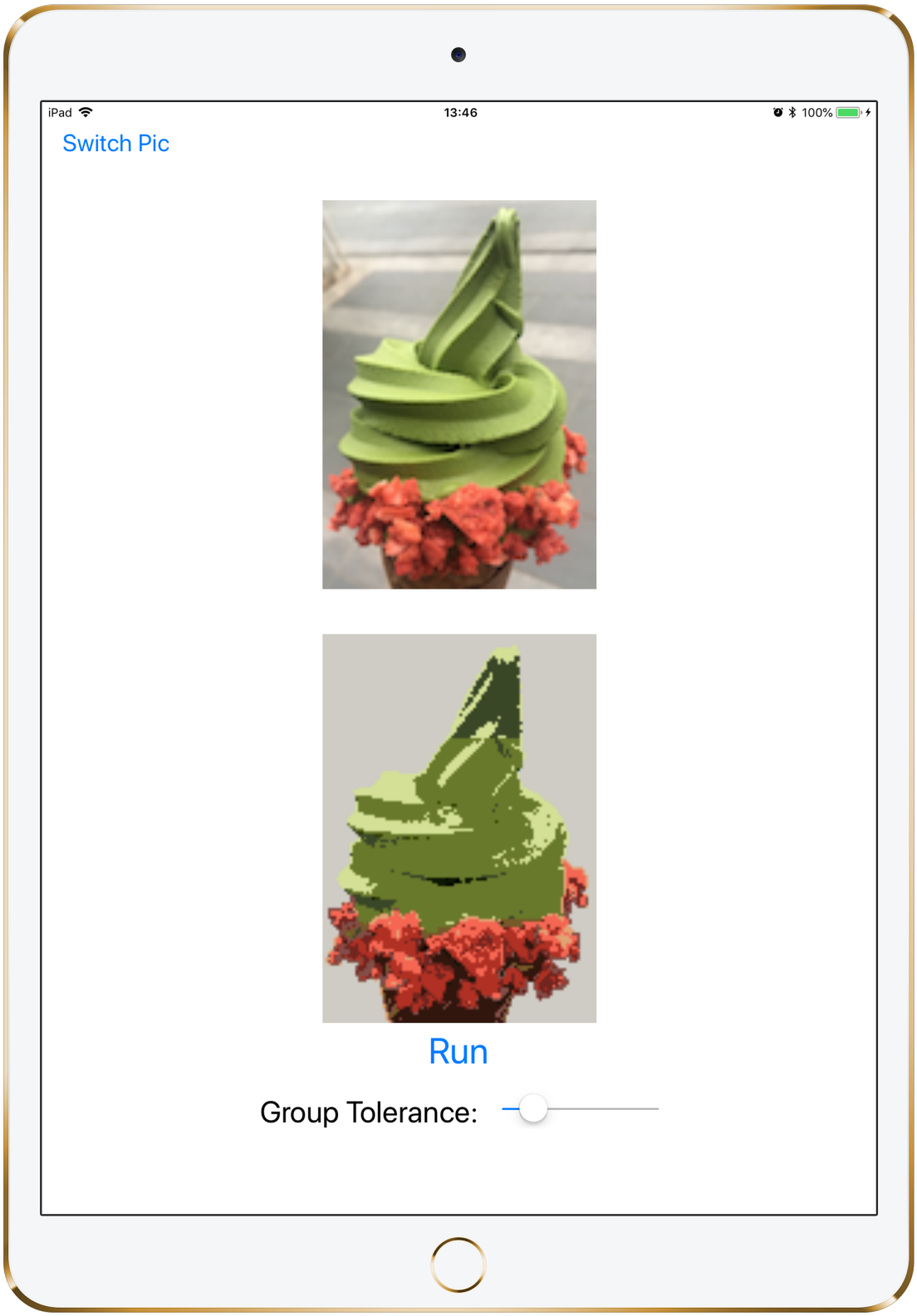
在这一次的图片分割任务中，我采取的是基于聚类的Meanshift方法。MeanShift最初由Fukunaga和Hostetler在1975年提出，但直到2002年PAMI的论文Mean Shift: A Robust Approach Toward Feature Space Analysis将它的原理和收敛性等重新整理阐述，并应用于计算机视觉和图像处理领域之后，才逐渐为人熟知。

MeanShift是通用的聚类算法，其本质是从离散的抽样点中估计密度函数极大值，这一过程通过迭代地求解概率密度梯度的方向实现。

应用于图像分割这一任务时，具体的实现细节又有细微的区分，例如概率密度函数的选取、核函数的选取，每次迭代移动部分点/全体点，聚类后分割的给出，等等。

## 主要工作

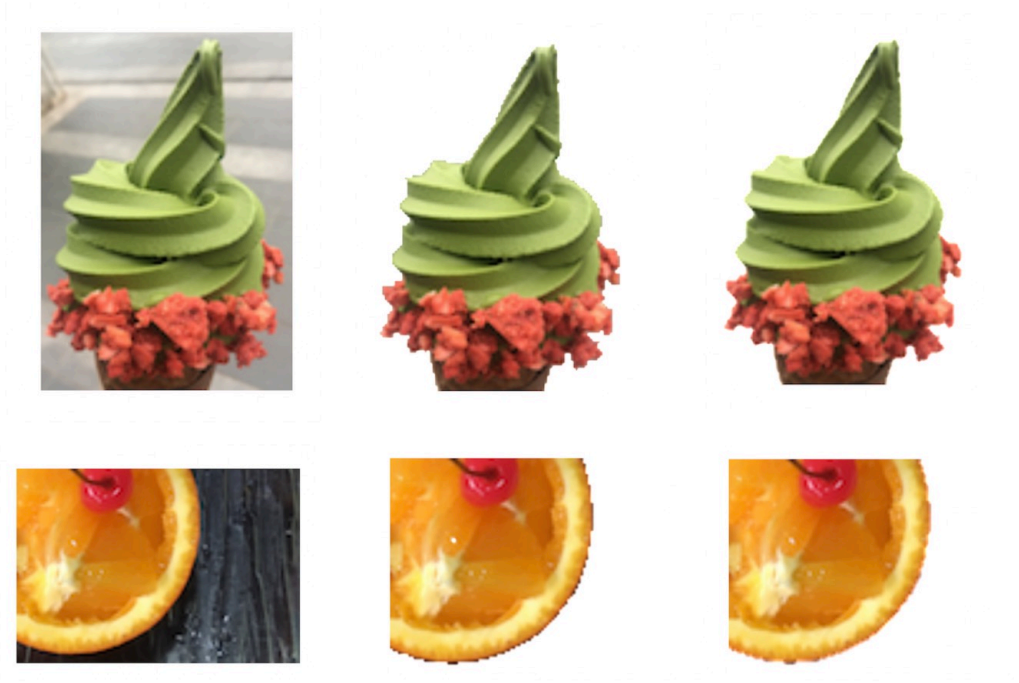
用Swift语言实现了基于Meanshift聚类的图像分割算法并实现了简单的UI，经测试能够在iOS平台上运行。效果如下图所示。




增加了在原图上移除背景的功能：[演示视频]

主要工作包括：

1. 色彩空间的变换和比较（RGB到Luv）
2. 不同核函数的实现和比较（Uniform和Gaussian）
3. 在固定尺寸窗口内进行搜索时通过建立反向索引减小时间复杂度
4. 移除背景时通过aplha通道的调节实现一定程度上的Border Matting
5. 从聚类到分割的Threshold参数可以通过Slider滑动调节，通过一定程度的用户交互调整分割的“程度”
6. UI与算法在不同的线程异步执行，防止运算时造成UI“卡死”的现象



左边一列是原图，中间列是分割后的图像，右列为分割+Border Matting后的图像。可见Border Matting使得分割更平滑自然。

## 文件说明

1. `Meanshift.xcodeproj` 可以在 `macOS` 平台上的 `Xcode` 打开直接运行
2. `Meanshift/ViewController.swift` 主要包括UI部分代码
3. `Meanshift/Meanshift.swift` 主要包括算法部分代码
4. `Meanshift/RGBAImage.swift` 主要来自一个开源的图像处理库SwiftImageProcessing，此处用于操作像素

## 分析改进

### 概率密度

我采用了Slides上介绍的将图像中的像素点在色彩空间中的密度作为概率密度的方法，尝试了RGB色彩空间和Luv色彩空间。

LUV色彩空间全称CIE 1976(L,*u*,v)*色彩空间，L*表示物体亮度，u*和v*是色度。于1976年由国际照明委员会提出，由CIE XYZ空间经简单变换得到，采用数学方式定义，具视觉统一性。在几乎所有的测试图象上，变换到Luv色彩空间后再聚类的表现比在RGB空间中直接聚类更好。可能的原因是RGB对光照、阴影变化不鲁棒，如下图所示。

左为原图，中为RGB色彩空间中的聚类结果，右为Luv色彩空间中的聚类结果。可见，RGB特征不能很好地处理冰淇淋纹理带来的阴影与灰色路面的关系，而Luv特征表现较好。

此外，还有工作指出将像素在色彩空间中的距离和图像中的距离相乘作为密度函数的来源，可以在Meanshift聚类时就引入空间信息，在更复杂的图像上效果较好。

## 核函数

采用了Uniform的核函数（物理意义：重心）和Gaussian核函数$k(x) = e^{-\frac{x^2}{2\sigma^2}}$，并对观察窗口中的点做了归一化。

在测试图像中，大多数情况下采用Gaussian核函数的迭代收敛速度更快，结果也更好。

## 运行速度

提交的代码实现的是论文中的Meanshift方法，也就是每轮对每个点取窗口并移动，运算量较大。实际应用中，有一些近似的方法用一提升计算效率，例如：

- 每次取未被扫过的点作为中心开始移动迭代至收敛，记录其他各点被不同聚类中心扫过的频数作为依据；

- 预处理特征点，以特征点位中心开窗口移动聚类
- 随着迭代进行自适应地调整bandwidth

…等等。

另外，Meanshift每一轮对每个像素的迭代实际上应该是并行的，并且都可以写成矩阵的形式，使用GPU并行计算是一个好的提升方案。就这个程序而言，可以采用iOS上底层的Metal计算框架对GPU编程实现。