[CS 4365/5354] Computer Vision - Lab 5 Report

Jose Perez

November 28, 2016

Contents

1. Lab 5

1.1 Introduction

Problem Brain atlases from model organisms such as rats and mice have revolutionized neuroscience. There are several databases online for different organisms but researchers don't map their spatial datasets to them. Mapping a spatial dataset to a brain atlas requires an expert to manually do it and is a very slow, laborious, error-prone process.

In this lab I attempt to automatically segment a dataset of histological stained cuts of a rat's brain using the normalized cuts algorithm. The dataset was acquired from Swanson's Brain Maps 2nd Edition Atlas which is available online (more info in the results section)

1.2 Proposed Solution Design and Implementation

1.2.1 User Interface

This lab was created and tested using iPython and so it is recommended to use it for running this lab.

Modules I separated my code as follows:

lab5.py
: Loads the images and performs the normalized cuts algorithm on them.

lib_ncut.py
: Contains an implementation of the normalized cuts algorithm.

1.2.2 Implementation

Preprocessing

Swanson's database is composed of 73 Adobe Illustrator projects. Every project is composed of 2 images: a histological stained section of the brain, and Swanson's manually mapped atlas. However, these are spread out over several layers and not so easy to just export.

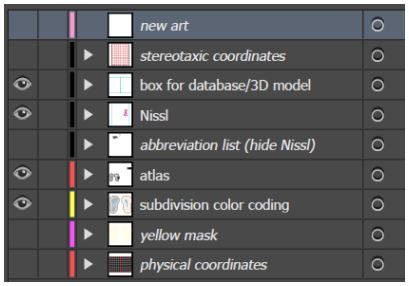

Figure 1.1: Layers of Illustrator. Nissl is the histological stained section

The first thing I did was convert the Adobe Illustrator PDF projects to SVG. For that purpose I used Adobe Illustrator's export feature in batch. This automatically exported the histological stained images as a jpg. That only left Swanson's manually mapped atlas images left to crop out.

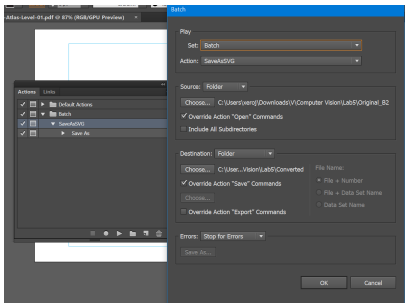

Figure 1.2: Illustrator Batch export. I created an action called SaveAsSVG and then applied it to all the layers

However, I was unable to crop out the ground truth from the Illustrator projects for this lab. So for this progress report I will not be able to compare my results with Swanson's manual mapping.

Normalized Cuts Algorithm

I segmented the histological stained images using normalized cuts. I used an implementation from the book Programming Computer Vision with Python by Jan Erik Solem.

I experimented with different parameters for the weights of the normalized cuts but I still haven't found a good balance. In the end I used the 1 as the weight for the distance and 0.0004 as the weight for similarity.

In order to speed up the process of calculating eigenvectors the images are resized to a smaller size (50x50) when calculating the eigenvectors.

After performing normalized cuts a figure is shown. The figure contains the original image and the segmented image. In the future the atlas image will also be shown here.

1.3 Experimental Results

The computer used to run these experiments is a Dell Inspiron 13 which has an Intel i7 Core and 12GB of memory.

These experiments were run in the Anaconda2 python environment using the Spyder IDE in an iPython console.

The timing (named duration below) of these experiments was taken using the default_timer of Python's timeit library.

The dataset used was L.W. Swanson's Brain Maps 2nd Edition Atlas (http://larrywswanson.com/?page_id=164).

The dataset was split as stated in the preprocessing section.

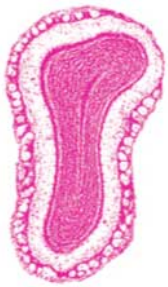

Figure 1.3: Left side of split. Histological section of level 01

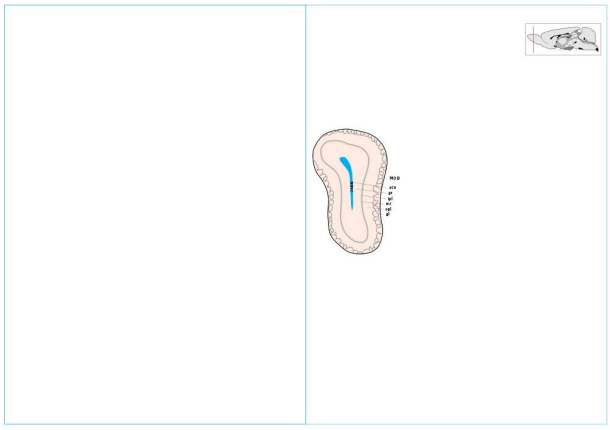

Figure 1.4: Right side of split. Atlas map of level 01. Requires cropping

These were the results of one run of the normalized cut algorithm. The number of clusters used for k-means is stated above each experiment. The total running time for the program was 142.420s

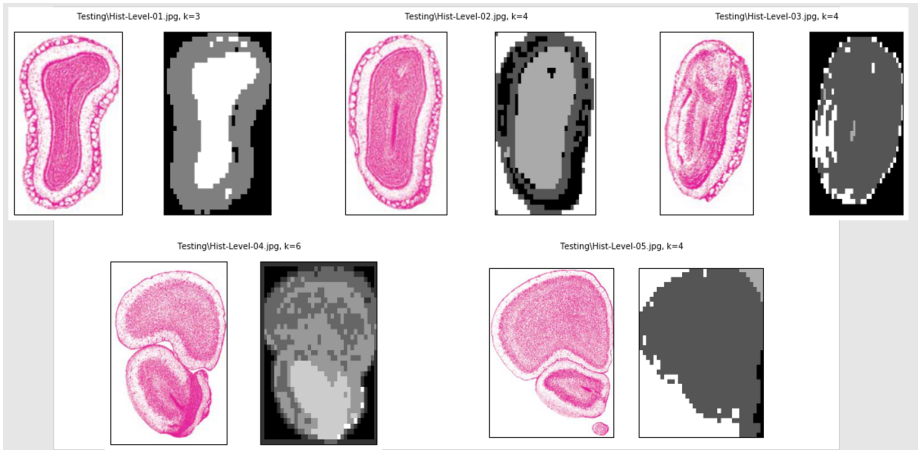

Figure 1.5: Levels 1-5

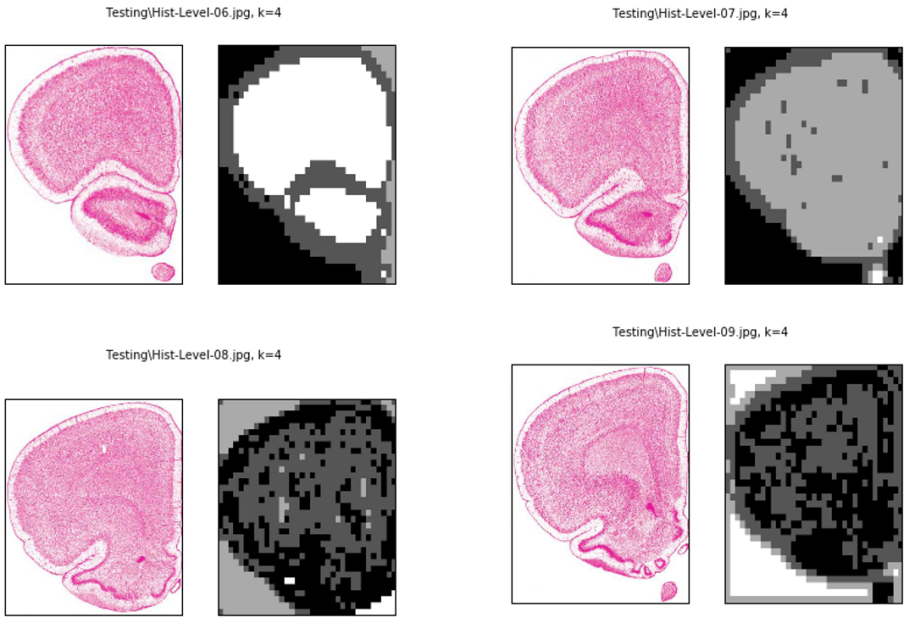

Figure 1.6: Levels 6-9

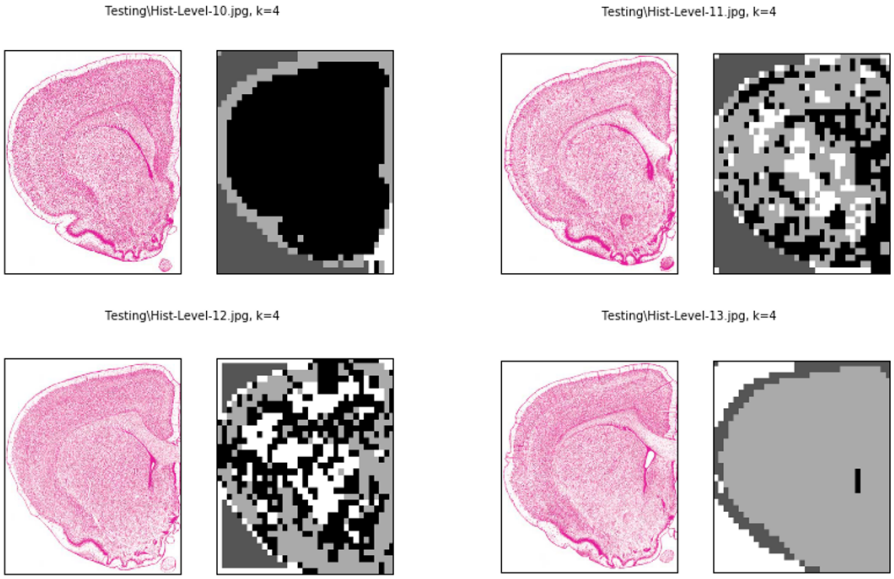

Figure 1.7: Levels 7-13

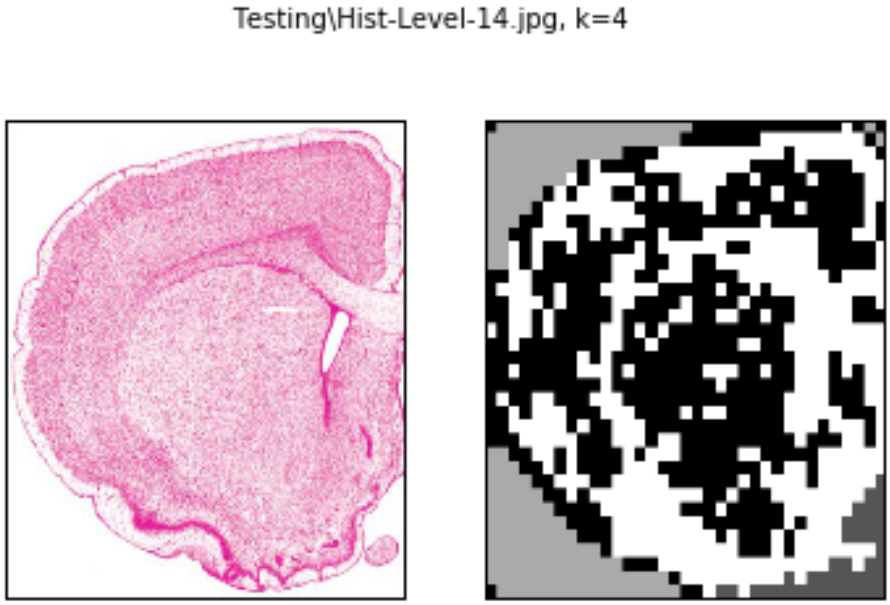

Figure 1.8: Level 14

1.4 Conclusion

Normalized cuts performs poorly in some levels of the atlas as seen in the results. There might be ways to tweak the algorithm for better performance but it might not be the best way to solve this problem.

There are many algorithms for image segmentation which I would like to try on this dataset. It's also hard to manually check the accuracy of the algorithms on all 73 levels of the dataset so I would like to try to figure out a way to find that out automatically.

I manually provided the number of clusters for k-means. There are other clustering algorithm that will try to find the number of clusters automatically which I would like to try.

Density might be a good feature to use for segmentation as well. In the histological sections the human eye can distinguish regions based on how closely packed the pixels in it are.

A. Source Code

lab5.py

```
# -*- coding: utf-8 -*-
"""
Course: CS 4365/5354 [Computer Vision]
Author: Jose Perez [ID: 80473954]
Assignment: Lab 5
Instructor: Olac Fuentes
"""
from timeit import default_timer as timer
from PIL import Image
from scipy.misc import imresize
from lib_ncut import ncut_graph_matrix, cluster

import numpy as np
import pylab as plt
import os

# Number of images
IMAGES = 15

# Image folder
DIR = "Testing"

# Prefix for histological cut images
PREFIX = "Hist-Level-"

# Prefix for atlas images
ATLAS_PREFIX = "Complete-Atlas-Level-"

# Image extension
EXT = ".jpg"

# Number of clusters for each level starting at level 1
############## 0, 1, 2, 3, 4, 5, 6, 7, 8,
   9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24
K = np.array([-1, 3, 4, 4, 6, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
   4, 4, 4, 4, 4])

# Size of images for eigenvector calculation
# Resize images for fast computation of eigenvectors
```

```
wid = 35

# Set the decimals for the timer results
DECIMALS = '{:.3f}'

program_start = timer()
for index in range(1, IMAGES):
    start_time = timer()

    # ZFill(2) adds a leading zero in front
    path = os.path.join(DIR, PREFIX + str(index).zfill(2) + EXT)
    #atlas_path = os.path.join(DIR, ATLAS_PREFIX + str(index).zfill(2) +
       EXT)
    print "Processing: ", path

    # Load images
    im = np.array(Image.open(path))
    #im_atlas = np.array(Image.open(atlas_path))
    (w, h, c) = im.shape

    # Resize image to (wid,wid) for fast eigenvector calculation
    rim = imresize(im,(wid,wid),interp='bilinear')
    rim = np.array(rim,'f')

    # Create normalized cut matrix
    A = ncut_graph_matrix(rim,sigma_d=1,sigma_g=.0004)

    # Cluster
    code, V = cluster(A,k=K[index],ndim=3)

    # Reshape to original image size
    codeim = imresize(code.reshape(wid,wid),(w,h),interp='nearest')

    # Plot original and segmented side by side
    figure = plt.figure(index)
    figure.suptitle(path + ", k=" + str(K[index]))

    # add_subplot(rows, column, index)
    # Plot the original image
    subplot = figure.add_subplot(1, 2, 1)
    subplot.set_xticks(())
    subplot.set_yticks(())
    subplot.imshow(im)

    # Plot the segmented image
    subplot = figure.add_subplot(1, 2, 2)
    subplot.set_xticks(())
    subplot.set_yticks(())
    subplot.imshow(codeim)

    # Plot the atlas image
```

```
    #subplot = figure.add_subplot(1, 3, 3)
    #subplot.set_xticks(())
    #subplot.set_yticks(())
    #subplot.imshow(im_atlas)

    print "Processing took", DECIMALS.format(timer() - start_time), "s"

print "Program ran for ",DECIMALS.format(timer() - program_start), "s"
```

lib_ncut.py

```
# -*- coding: utf-8 -*-
"""
Course: CS 4365/5354 [Computer Vision]
Author: Jose Perez [ID: 80473954]
Assignment: Lab 5
Instructor: Olac Fuentes
"""
import numpy as np
def ncut_graph_matrix(im,sigma_d=1e2,sigma_g=1e-2):
    """ Create matrix for normalized cut. The parameters are
    the weights for pixel distance and pixel similarity. """
    m,n = im.shape[:2]
    N = m*n
    # normalize and create feature vector of RGB or grayscale
    if len(im.shape)==3:
        for i in range(3):
            im[:,:,i] = im[:,:,i] / im[:,:,i].max()
        vim = im.reshape((-1,3))
    else:
        im = im / im.max()
        vim = im.flatten()

    # x,y coordinates for distance computation
    xx,yy = np.meshgrid(range(n),range(m))
    x,y = xx.flatten(),yy.flatten()
    # create matrix with edge weights
    W = np.zeros((N,N),'f')
    for i in range(N):
        for j in range(i,N):
            d = (x[i]-x[j])**2 + (y[i]-y[j])**2
            W[i,j] = W[j,i] = np.exp(-1.0*sum((vim[i]-vim[j])**2)/sigma_g)
               * np.exp(-d/sigma_d)
    return W

from scipy.cluster.vq import whiten, vq, kmeans
def cluster(S,k,ndim):
    """ Spectral clustering from a similarity matrix."""
    # check for symmetry
    if np.sum(np.abs(S-S.T)) > 1e-10:
        print 'not symmetric'
```

```
    # create Laplacian matrix
    rowsum = np.sum(np.abs(S),axis=0)
    D = np.diag(1 / np.sqrt(rowsum + 1e-6))
    L = np.dot(D, np.dot(S,D))
    # compute eigenvectors of L
    U,sigma,V = np.linalg.svd(L)
    # create feature vector from ndim first eigenvectors
    # by stacking eigenvectors as columns
    features = np.array(V[:ndim]).T
    # k-means
    features = whiten(features)
    centroids,distortion = kmeans(features,k)
    code,distance = vq(features,centroids)
    return code,V
```

B. Academic Honesty Certification

I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class.

Jose Perez

Date