HAAG Week 12 Report -Lizard Jaw Segmentation

Shuyu Tian

Time-Log

What did I do this week?

- I provide justification and background information on why specific functions and choices in the code were made, specifically regarding the registration portion of the code as well the non-rigid registration method explored in parallel last week
- I integrated functions used in rigid-registration notebook into the non-rigid registration method in preparation for next week's computational advisor meeting
- As the webpage manager and meeting leader of my group, I updated the webpage of the Stroud group with a subpage for the Lizard Jaw Segmentation group's weekly reports and group meeting recording link.

- What I will do next week
    - I will review with the computational advisor on the registration code provide for improvements and research direction if this process is not done well
    - I will resume collecting jaw segmentation data for training
- Blockers, things I want to flag, problems, etc.
    - If rigid and non-rigid registration methods are not producing usable data, manual segmentation work on 3Dslicer will be needed again and will be a big time-sink

Abstract:

An unresolved issue in contemporary biomedicine is the overwhelming number and diversity of complex images that require annotation, analysis and interpretation. Recent advances in Deep Learning have revolutionized the field of computer vision, creating algorithms that compete with human experts in image segmentation tasks. However, these frameworks require large human-annotated datasets for training and the resulting "black box" models are difficult to interpret. In this study, we introduce *Kartezio*, a modular Cartesian Genetic Programming-based computational strategy that generates fully transparent and easily interpretable image processing pipelines by iteratively assembling and parameterizing computer vision functions. The pipelines thus generated exhibit comparable precision to state-of-the-art Deep Learning approaches on instance segmentation tasks, while requiring drastically smaller training datasets. This Few-Shot Learning method confers tremendous flexibility, speed, and functionality to this approach. We then deploy Kartezio to solve a series of semantic and instance segmentation problems, and demonstrate its utility across diverse images ranging from multiplexed tissue histopathology images to high resolution microscopy images. While the flexibility, robustness and practical utility of Kartezio make this fully explicable evolutionary designer a potential game-changer in the field of biomedical image processing, Kartezio remains complementary and potentially auxiliary to mainstream Deep Learning approaches.

Link: https://www.nature.com/articles/s41467-023-42664-x?

General summary: The paper introduces Kartezio, a computational strategy based on Cartesian Genetic Programming (CGP) that automatically generates transparent and interpretable image processing pipelines by assembling and parameterizing computer vision functions. Kartezio demonstrates precision comparable to state-of-the-art deep learning methods in instance segmentation tasks while requiring significantly smaller training datasets. This few-shot learning capability enhances its flexibility and efficiency across diverse biomedical imaging challenges, including multiplexed tissue histopathology and high-resolution microscopy images. The study positions Kartezio as a complementary tool to mainstream deep learning approaches, offering a robust and practical solution for biomedical image processing.

What did you do and prove it

1. Below are screenshots of updated non-rigid registration code based on work done with rigid-registration:

```
import numpy as np
import nrrd as pynrrd
import open3d as o3d
import copy
import matplotlib.pyplot as plt
import time
```

```
def volume_to_point_cloud(volume, threshold=None):
    min_val, max_val = np.min(volume), np.max(volume)
    if threshold is None or threshold > max_val:
        threshold = max_val * 0.5
    temp = np.asarray(volume > threshold)
    z, y, x = temp.nonzero()
    points = np.vstack((x, y, z)).T
    return points
```

```
def segment_lower_jaw(point_cloud):
    bbox = point_cloud.get_axis_aligned_bounding_box()
    min_bound = bbox.min_bound
    max_bound = bbox.max_bound
    center_z = (min_bound[2] + max_bound[2]) / 1.5
    filtered_points = [p for p in np.asarray(point_cloud.points) if p[2] < center_z]
    segmented_pcd = o3d.geometry.PointCloud()
    segmented_pcd.points = o3d.utility.Vector3dVector(filtered_points)
    return segmented_pcd
```

```python
def prepare_dataset(voxel_size, source_file, target_file):
    source_data, _ = pynrrd.read(source_file)
    target_data, _ = pynrrd.read(target_file)
    source_points = volume_to_point_cloud(source_data)
    target_points = volume_to_point_cloud(target_data)
    source_pcd = o3d.geometry.PointCloud()
    source_pcd.points = o3d.utility.Vector3dVector(source_points)
    target_pcd = o3d.geometry.PointCloud()
    target_pcd.points = o3d.utility.Vector3dVector(target_points)
    source_pcd = segment_lower_jaw(source_pcd)
    target_pcd = segment_lower_jaw(target_pcd)
    source_down, source_fpfh = preprocess_point_cloud(source_pcd, voxel_size)
    target_down, target_fpfh = preprocess_point_cloud(target_pcd, voxel_size)
    return source_data, target_data, source_down, target_down, source_fpfh, target_fpfh
```

```python
def visualize_registration(source, target, transformation=np.eye(4), title='Point Cloud Alignment'):
    source_temp = copy.deepcopy(source)
    target_temp = copy.deepcopy(target)
    source_temp.paint_uniform_color([1, 0, 0])
    target_temp.paint_uniform_color([0, 1, 0])
    source_temp.transform(transformation)
    o3d.visualization.draw_geometries([source_temp, target_temp], window_name=title)
```

[ ]:
```python
import probreg.cpd

def apply_nonrigid_registration(source_down, target_down):
    tf_param, _, _ = probreg.cpd.registration_cpd(source_down, target_down)
    transformed_source = copy.deepcopy(source_down)
    transformed_source.points = tf_param.transform(source_down.points)
    return transformed_source, tf_param
```

[ ]:
```python
# Apply nonrigid registration
transformed_source, tf_param = apply_nonrigid_registration(source_down, target_down)
visualize_registration(transformed_source, target_down, np.eye(4), title='After Nonrigid Registration')
```

Additionally, I updated the Stroud lab webpage with my group's relevant information up to week 11 of this semester (see images below).