

Lib/pipeline.py

```
1 # =====
2 # Pipeline stages
3 # =====
4
5 import gc
6
7 import numpy as np
8 import SimpleITK as sitk
9 import itk
10 from itk import TubeTK
11
12 from scipy.ndimage import gaussian_filter, convolve, uniform_filter, gaussian_filter, binary_dilation, binary_closing,
    label as ndlabel
13 from scipy.spatial import cKDTree
14
15 from skimage.morphology import skeletonize, ball
16 from skimage.graph import route_through_array
17
18
19 def stageRemoveBackground(raw, outsideIJKs, backgroundIJKs, patch_radius=5):
20     """Zero out voxels below the midpoint between the outside and background
21     intensity distributions. Requires both outside and background seeds.
22     Falls back to 95th percentile of outside alone if background seeds are absent.
23     Returns raw unchanged if no outside points are provided.
24     """
25     if not outsideIJKs:
26         return raw
27
28     out_vox = _poolPatches(raw, outsideIJKs, patch_radius)
29
30     if backgroundIJKs:
31         bg_vox = _poolPatches(raw, backgroundIJKs, patch_radius)
32         cutoff = float((np.max(out_vox) + np.min(bg_vox)) / 2.0) #midway
```

```
33     else:
34         cutoff = float(np.percentile(out_vox, 95))
35
36     result = raw.astype(np.float32, copy=True)
37     result[result <= cutoff] = 0.0
38     return result
39
40
41 # SimpleITK based frangi, uses C++, much better than skimage
42 # https://github.com/zenr/simpleitk-doc#sitkobjectnessmeasure
43 def _frangiBlock(block, spacing, scales_mm, structure_strength):
44     """Run multi-scale Frangi on a single block.
45     Gaussian smoothing is applied per scale before ObjectnessMeasure,
46     as sitk.ObjectnessMeasure computes the Hessian via discrete differences
47     and requires pre-smoothing at each scale.
48     """
49     maxV = np.zeros(block.shape, dtype=np.float32)
50     for sigma in scales_mm:
51         img = sitk.GetImageFromArray(block)
52         img.SetSpacing(spacing)
53         smoothed = sitk.RecursiveGaussian(img, sigma=float(sigma))
54         result = sitk.ObjectnessMeasure(
55             smoothed,
56             objectDimension=1,
57             alpha=0.5, beta=0.5, gamma=structure_strength,
58             brightObject=True,
59             scaleObjectnessMeasure=True,
60         )
61         np.maximum(maxV, sitk.GetArrayFromImage(result), out=maxV)
62         del img, smoothed, result
63     return maxV
64
65
66 # NOTE: this is custom rolled from Claude to help alleviate memory overflow issue
67 # A halo is to have some overlap between regions - Frangi is context based
68 def stageFrangi(volume, spacing, vessel_size, vessel_std, structure_strength,
```

```
69         n_scales=12, tiles_per_axis=1):
70     """Multi-scale bright-only Frangi (SimpleITK). Output in [0, 1].
71
72     When ``tiles_per_axis > 1`` the volume is processed in
73     ``tiles_per_axis3`` halo-padded blocks to bound peak memory.
74
75     Halo is sized at 4*sigma_max in mm so per-tile output matches an un-tiled run
76     on the inner (non-halo) region. p99 normalisation is global on the
77     assembled accumulator.
78
79     spacing: float, size of a voxel, 0.006 here
80     vessel_size: int, in voxels
81     vessel_std: standard deviation of size
82
83     """
84     sigma_min_vox = max(0.5, (vessel_size - vessel_std) / 2.0)
85     sigma_max_vox = max(1.0, float(vessel_size + vessel_std) / 2.0)
86
87     meanSpacing = float(np.mean(spacing))
88     scales_mm = (np.linspace(sigma_min_vox, sigma_max_vox, n_scales)
89                  * meanSpacing).tolist()
90
91
92     # Smoothing done across the whole volume
93     volume = gaussian_filter(volume, sigma=3).astype(np.float32, copy=False)
94
95     # Tiling as implemented by Claude - to reduce peak memory use
96     nTiles = max(1, int(tiles_per_axis))
97     if nTiles == 1:
98         maxV = _frangiBlock(volume, spacing, scales_mm, structure_strength)
99     else:
100         # Halo per axis: 4*sigma_max in mm → voxels using each axis' spacing.
101         # spacing is (x, y, z); numpy axes are (k=z, j=y, i=x).
102         sigma_max_mm = float(max(scales_mm))
103         halo_kji = (
104             max(8, int(np.ceil(4.0 * sigma_max_mm / float(spacing[2])))),
```

```
105         max(8, int(np.ceil(4.0 * sigma_max_mm / float(spacing[1])))),
106         max(8, int(np.ceil(4.0 * sigma_max_mm / float(spacing[0])))),
107     )
108
109     K, J, I = volume.shape
110     k_edges = np.linspace(0, K, nTiles + 1, dtype=int)
111     j_edges = np.linspace(0, J, nTiles + 1, dtype=int)
112     i_edges = np.linspace(0, I, nTiles + 1, dtype=int)
113
114     maxV = np.zeros(volume.shape, dtype=np.float32)
115
116     for kt in range(nTiles):
117         k0, k1 = int(k_edges[kt]), int(k_edges[kt + 1])
118         pk0 = max(0, k0 - halo_kji[0])
119         pk1 = min(K, k1 + halo_kji[0])
120         for jt in range(nTiles):
121             j0, j1 = int(j_edges[jt]), int(j_edges[jt + 1])
122             pj0 = max(0, j0 - halo_kji[1])
123             pj1 = min(J, j1 + halo_kji[1])
124             for it in range(nTiles):
125                 i0, i1 = int(i_edges[it]), int(i_edges[it + 1])
126                 pi0 = max(0, i0 - halo_kji[2])
127                 pi1 = min(I, i1 + halo_kji[2])
128
129                 # Padded read region (own buffer for SimpleITK).
130                 tile = volume[pk0:pk1, pj0:pj1, pi0:pi1].copy()
131                 tileOut = _frangiBlock(tile, spacing, scales_mm,
132                                     structure_strength)
133
134                 del tile
135
136                 # Crop the halo: inner region offset inside the tile.
137                 ck0 = k0 - pk0
138                 cj0 = j0 - pj0
139                 ci0 = i0 - pi0
140                 ck1 = ck0 + (k1 - k0)
141                 cj1 = cj0 + (j1 - j0)
```

```
141         cil = ci0 + (i1 - i0)
142
143         maxV[k0:k1, j0:j1, i0:i1] = \
144             tileOut[ck0:ck1, cj0:cj1, ci0:ci1]
145         del tileOut
146         gc.collect()
147
148     # Global p99 normalisation.
149     nonzero = maxV[maxV > 0]
150     if len(nonzero) > 0:
151         p99 = float(np.percentile(nonzero, 99))
152         if p99 > 0:
153             maxV /= p99
154             np.clip(maxV, 0.0, 1.0, out=maxV)
155     return maxV
156
157
158 def _poolPatches(volume, ijkPoints, radius):
159     shp = volume.shape
160     patches = []
161     for (i, j, k) in ijkPoints:
162         patches.append(volume[
163             max(k - radius, 0):min(k + radius + 1, shp[0]),
164             max(j - radius, 0):min(j + radius + 1, shp[1]),
165             max(i - radius, 0):min(i + radius + 1, shp[2]),
166             ].ravel())
167     if not patches:
168         return np.empty(0, dtype=volume.dtype)
169     return np.concatenate(patches)
170
171
172 def stageGrowFromPMap(probMap, spacing, origin, seeds,
173                        vessel_size, vessel_std,
174                        n_rounds=2, max_new_seeds=150,
175                        ridge_threshold=0.05, ridge_iters=2,
176                        blob_min_prob=0.15):
```

```
177 """Grow vasculature from seeds through probMap. Blob filtering is
178 applied per round to the newly added voxels only, keeping the
179 user-seed pass entirely untouched.
180 """
181 # Pass 1: user seeds – no filtering, these are confirmed
182 mask, _, _ = stageRidgeExtract(
183     probMap, spacing, origin, seeds,
184     vessel_size, vessel_std, ridge_threshold, ridge_iters)
185 soft = mask.astype(np.float32)
186
187 min_spacing = max(2, vessel_size)
188 for _ in range(n_rounds):
189     newSeeds = stageAutoSeed(
190         probMap, probMap, mask, seeds,
191         vessel_size,
192         prob_threshold=ridge_threshold,
193         intensity_sigma=10.0,
194         min_seed_spacing_vox=min_spacing,
195         max_seeds=max_new_seeds)
196     if not newSeeds:
197         break
198
199     add, _, _ = stageRidgeExtract(
200         probMap, spacing, origin, newSeeds,
201         vessel_size, vessel_std, ridge_threshold, ridge_iters)
202
203     # Filter blobs in `add` only – but keep voxels that overlap the
204     # existing mask (these are connections back to confirmed vessels).
205     add_new = add & ~mask
206     add_filt = stageFilterBlobs(add_new, probMap, blob_min_prob)
207     add_final = add_filt | (add & mask)
208
209     mask |= add_final
210     soft += add_final.astype(np.float32)
211
212 total_rounds = n_rounds + 1 # +1 for user seed pass
```

```
213     soft = soft / total_rounds
214     np.clip(soft, 0.0, 1.0, out=soft)
215
216     return mask, soft
217
218
219 def stageExpandCenterlines(centerline_mask, denoised, vessel_seed_intensities,
220                             vessel_size, intensity_sigma=2.0):
221     """Expand centerline voxels radially using intensity as the acceptance criterion.
222     Accepts neighbours within vessel_size//2 voxels whose intensity falls
223     within intensity_sigma standard deviations of the vessel seed distribution.
224     """
225     mu_v = float(np.mean(vessel_seed_intensities))
226     sig_v = float(np.std(vessel_seed_intensities))
227     lo = mu_v - intensity_sigma * sig_v
228     hi = mu_v + intensity_sigma * sig_v
229
230     radius = max(1, vessel_size // 2)
231     struct = ball(radius)
232     dilated = binary_dilation(centerline_mask, structure=struct)
233
234     # Accept dilated voxels only where intensity is vessel-like
235     intensity_gate = (denoised >= lo) & (denoised <= hi)
236     return dilated & intensity_gate & ~centerline_mask | centerline_mask
237
238
239 def stageLocalIntensityProb(roi, vessel_size, vessel_seed_ijks):
240     """Local-relative intensity probability via local z-score.
241     Voxels significantly brighter than their local neighbourhood are vessel candidates.
242     """
243
244     window = max(5, vessel_size * 10) # local region
245
246     # Get the vessel mean and std
247     v_vox = _poolPatches(roi, vessel_seed_ijks, 5)
248     mu_v = float(np.mean(v_vox))
```

```
249     sig_v = float(np.std(v_vox))
250
251     local_mean = uniform_filter(roi, size=window, mode='reflect')
252     local_var = uniform_filter(roi * roi, size=window, mode='reflect') - local_mean ** 2
253     local_std = np.sqrt(np.maximum(local_var, 1.0)) # floor avoids div0
254
255     z = (roi - local_mean) / local_std
256
257     # Calibrate against vessel seed z-scores: scale so seed mean → 0.8
258     v_z = _poolPatches(z, vessel_seed_ijks, max(1, vessel_size // 2))
259     target = 0.8
260     scale = target / max(float(np.mean(v_z)), 0.5)
261     prob = np.clip(z * scale, 0.0, 1.0).astype(np.float32)
262
263     return prob, {
264         "mu_v": mu_v, "sig_v": sig_v
265     }
266
267
268 # Ridge following
269 def stageRidgeExtract(volume, spacing, origin, seedIJKs,
270                       vessel_size, vessel_std, ridge_threshold,
271                       n_iterations):
272
273     mask = np.zeros(volume.shape, dtype=bool)
274     if not seedIJKs:
275         return mask, 0, 0
276
277     meanSp = float(np.mean(spacing))
278     scale_min = max(0.001, max(0.5, (vessel_size - vessel_std) / 2.0) * meanSp)
279     scale_max = max(scale_min + 1e-3, max(1.0, float(vessel_size + vessel_std)) * meanSp)
280
281     itkImage = itk.GetImageFromArray(volume.astype(np.float32, copy=False))
282     itkImage.SetSpacing([float(spacing[0]), float(spacing[1]), float(spacing[2])])
283     itkImage.SetOrigin([float(origin[0]), float(origin[1]), float(origin[2])])
284
```

```
285     seg = _newSegmentTubes(itkImage, scale_min, scale_max, ridge_threshold)
286
287     nTubes = 0
288     nPoints = 0
289     nextId = 0
290
291     # Iteration 0: extract from user/auto seeds (IJK -> physical)
292     pendingPhys = []
293     for (i, j, k) in seedIJKs:
294         ci = itk.ContinuousIndex[itk.D, 3]()
295         ci[0] = float(i); ci[1] = float(j); ci[2] = float(k)
296         phys = itkImage.TransformContinuousIndexToPhysicalPoint(ci)
297         tube = _extractOneTube(seg, phys, nextId)
298         nextId += 1
299         if tube is not None:
300             _rasterizeTube(tube, mask, origin, spacing, scale_max)
301             nTubes += 1
302             nPoints += tube.GetNumberOfPoints()
303             pendingPhys.extend(_tubeEndpointsPhys(tube))
304
305     # Iterations 1..N: re-seed from previous tubes' endpoints
306     for _ in range(max(0, n_iterations)):
307         if not pendingPhys:
308             break
309         newEndpoints = []
310         for ep in pendingPhys:
311             phys = itk.Point[itk.D, 3]()
312             phys[0] = float(ep[0]); phys[1] = float(ep[1]); phys[2] = float(ep[2])
313             tube = _extractOneTube(seg, phys, nextId)
314             nextId += 1
315             if tube is not None:
316                 _rasterizeTube(tube, mask, origin, spacing, scale_max)
317                 nTubes += 1
318                 nPoints += tube.GetNumberOfPoints()
319                 newEndpoints.extend(_tubeEndpointsPhys(tube))
320     pendingPhys = newEndpoints
```

```
321
322     return mask, nTubes, nPoints
323
324
325 # TubeTK ridge extraction
326 # https://github.com/InsightSoftwareConsortium/ITKTubeTK/
327 def _newSegmentTubes(itkImage, scale_min, scale_max, ridge_threshold):
328     SegFilterType = TubeTK.SegmentTubes[
329         itk.Image[itk.F, 3]
330     ]
331     seg = SegFilterType.New()
332     seg.SetInput(itkImage)
333     attrs = [
334         ('SetRadiusInObjectSpace', scale_min),
335         ('SetRadiusStart', scale_min),
336         ('SetRadiusMin', scale_min),
337         ('SetRadiusMax', scale_max),
338         ('SetMinRidgeness', ridge_threshold),
339         ('SetMinRidgenessStart', ridge_threshold),
340         ('SetMinRoundness', 0.0),
341         ('SetMinRoundnessStart', 0.0),
342         ('SetMinCurvature', 0.0),
343         ('SetMinCurvatureStart', 0.0),
344         ('SetMinLevelness', 0.0),
345         ('SetMinLevelnessStart', 0.0),
346         ('SetMinLength', 1),
347         ('SetBorderInIndexSpace', 0),
348         ('SetOptimizeRadius', True),
349         ('SetDynamicScale', True),
350     ]
351     for name, val in attrs:
352         if hasattr(seg, name):
353             try:
354                 getattr(seg, name)(val)
355             except Exception:
356                 print("Exception in _newSegmentTubes - ridge extraction")
```

```
357     return seg
358
359
360 def _extractOneTube(seg, physPt, tubeId):
361     try:
362         tube = seg.ExtractTubeInObjectSpace(physPt, tubeId)
363         if tube is not None and tube.GetNumberOfPoints() > 0:
364             return tube
365     except Exception as e:
366         print(f"Exception {e} in singlar tubular extraction for ridge")
367     return None
368
369
370 def _tubeEndpointsPhys(tube):
371     n = tube.GetNumberOfPoints()
372     # A tube has at least 2 pts
373     if n < 2:
374         return []
375     pts = tube.GetPoints()
376     out = []
377     for ep in (pts[0], pts[-1]):
378         pos = ep.GetPositionInObjectSpace()
379         out.append([float(pos[0]), float(pos[1]), float(pos[2])])
380     return out
381
382
383 def _rasterizeTube(tube, mask, origin, spacing, scale_max):
384     sx, sy, sz = float(spacing[0]), float(spacing[1]), float(spacing[2])
385     ox, oy, oz = float(origin[0]), float(origin[1]), float(origin[2])
386     K, J, I = mask.shape
387     for pt in tube.GetPoints():
388         pos = pt.GetPositionInObjectSpace()
389         radius = scale_max
390         if hasattr(pt, 'GetRadiusInObjectSpace'):
391             r = pt.GetRadiusInObjectSpace()
392             if r > 0:
```

```
393         radius = min(float(r), scale_max)
394         ri = int(round((float(pos[0]) - ox) / sx))
395         rj = int(round((float(pos[1]) - oy) / sy))
396         rk = int(round((float(pos[2]) - oz) / sz))
397         radVox = (max(1, int(round(radius / sx))),
398                  max(1, int(round(radius / sy))),
399                  max(1, int(round(radius / sz))))
400         k_lo = max(rk - radVox[2], 0); k_hi = min(rk + radVox[2] + 1, K)
401         j_lo = max(rj - radVox[1], 0); j_hi = min(rj + radVox[1] + 1, J)
402         i_lo = max(ri - radVox[0], 0); i_hi = min(ri + radVox[0] + 1, I)
403         if k_hi <= k_lo or j_hi <= j_lo or i_hi <= i_lo:
404             continue
405         kk, jj, ii = np.ogrid[k_lo:k_hi, j_lo:j_hi, i_lo:i_hi]
406         dist2 = (((kk - rk) * sz) ** 2 +
407                 ((jj - rj) * sy) ** 2 +
408                 ((ii - ri) * sx) ** 2)
409         mask[k_lo:k_hi, j_lo:j_hi, i_lo:i_hi] |= (dist2 <= radius * radius)
410
411
412 # Auto-seed discovery
413 def stageAutoSeed(volume, probMap, currentMask, vesselSeedIJKs,
414                   vessel_size, prob_threshold=0.10,
415                   intensity_sigma=2.0, min_seed_spacing_vox=4,
416                   max_seeds=200):
417     """Pick high-probability voxels in the seed-intensity bracket without existing
418     segmentation. Returns IJK list (greedy spaced).
419
420     TODO: this could make use of frangi to suppress some FP
421     """
422     if not vesselSeedIJKs:
423         return []
424     rad = max(1, vessel_size // 2)
425     pooled = _poolPatches(volume, vesselSeedIJKs, rad)
426     if pooled.size == 0:
427         return []
428     mu = float(np.mean(pooled))
```

```
429     sd = max(float(np.std(pooled)), 0.5)
430     lo = mu - intensity_sigma * sd
431     hi = mu + intensity_sigma * sd
432
433     candidate = (probMap >= prob_threshold) & (volume >= lo) & (volume <= hi)
434     candidate &= ~currentMask
435     if not candidate.any():
436         return []
437
438     # (N, 3) KJI
439     coords = np.argwhere(candidate)
440     scores = probMap[candidate]
441     order = np.argsort(-scores)
442     coords = coords[order]
443     del scores
444
445     # Greedy minimum-spacing subsample
446     selected = []
447     spacing_sq = float(min_seed_spacing_vox) ** 2
448     for kji in coords:
449         if len(selected) >= max_seeds:
450             break
451         ok = True
452         for s in selected:
453             d = kji - s
454             if int(d[0]*d[0] + d[1]*d[1] + d[2]*d[2]) < spacing_sq:
455                 ok = False
456                 break
457         if ok:
458             selected.append(kji)
459     return [(int(c[2]), int(c[1]), int(c[0])) for c in selected]
460
461
462 # Component-mean filtering to remove blobs or filled areas using
463 # the probMap previously generated (frangi and intensity)
464 def stageFilterBlobs(mask, probMap, min_mean_prob):
```

```
465     """Drop connected components whose mean prob is below threshold."""
466     if min_mean_prob <= 0 or not mask.any():
467         return mask
468     labeled, n = ndlabel(mask)
469     if n == 0:
470         return np.zeros_like(mask)
471
472     # flatten arrays - more efficient, avoids looping
473     flat_lbl = labeled.ravel()
474     flat_prb = probMap.ravel()
475     sums = np.bincount(flat_lbl, weights=flat_prb, minlength=n + 1)
476     counts = np.bincount(flat_lbl, minlength=n + 1).astype(np.float64)
477     means = np.zeros_like(sums)
478     nonzero = counts > 0
479     means[nonzero] = sums[nonzero] / counts[nonzero]
480     means[0] = 0.0 # background label
481     keep_labels = np.where(means >= min_mean_prob)[0]
482     keep_labels = keep_labels[keep_labels > 0]
483     if keep_labels.size == 0:
484         return np.zeros_like(mask)
485     return np.isin(labeled, keep_labels)
486
487
488
489 def stageClosing(mask, iters = 2):
490     """Basic binary closing"""
491     return binary_closing(mask, iterations=iters)
492
493
494 def stageReconnect(mask, probMap, vessel_size,
495                    max_gap_voxels=None, min_path_prob=0.05):
496     """Bridge nearby endpoints using probMap as supporting evidence.
497     For each pair of disconnected vessel endpoints within max_gap_voxels,
498     find a minimum-cost path through (1 - probMap) and accept it if the
499     average cost is below 1 - min_path_prob.
500     """
```

```
501 max_gap = max_gap_voxels if max_gap_voxels else vessel_size * 3
502
503 skel      = skeletonize(mask.astype(np.uint8)).astype(bool)
504 kernel    = np.ones((3, 3, 3), dtype=np.int32); kernel[1, 1, 1] = 0
505 nc        = convolve(skel.astype(np.int32), kernel, mode='constant', cval=0)
506 endpoints = np.argwhere(skel & (nc == 1))
507 if len(endpoints) < 2:
508     return mask
509
510 comp_labels, _ = ndlabel(mask)
511 ep_labels      = comp_labels[endpoints[:, 0], endpoints[:, 1], endpoints[:, 2]]
512
513 tree = cKDTree(endpoints)
514 pairs = tree.query_pairs(r=max_gap)
515 cost  = (1.0 - probMap).astype(np.float32)
516 np.clip(cost, 0.01, 1.0, out=cost)
517
518 out = mask.copy()
519 bridged = 0
520 for ia, ib in pairs:
521     if ep_labels[ia] == ep_labels[ib]:
522         continue # already connected
523     pA, pB = endpoints[ia], endpoints[ib]
524     try:
525         path, total = route_through_array(
526             cost, tuple(pA), tuple(pB), fully_connected=True)
527     except Exception:
528         continue
529     if total / len(path) > (1.0 - min_path_prob):
530         continue
531     for z, y, x in path:
532         out[z, y, x] = True
533     bridged += 1
534 return out
```