Biological signal analysis – May 2024: Dana Venkert and Oded Khazon, final project

This project portrays the analysis of 3D data acquired by two-photon microscopy of a vascular network in the mouse brain with the purpose of generating a 3D mask of brain vasculature captured using in vivo intravital microscopy. The data for this project was a 3D stack tiff file.

To perform the required analysis, we applied several pre-processing steps on the data, to attempt to overcome the low signal-to-noise ratio (SNR) of the raw data. Particularly, the data contained depth-dependent noise, as the increase in depth of the image also results in an increase in noise. Therefore, we applied depth-dependent noise-reduction pipeline. Furthermore, additional filtering was required to filter a grid-like patten which appeared in many images, most likely due to the technical features of the data acquisition.

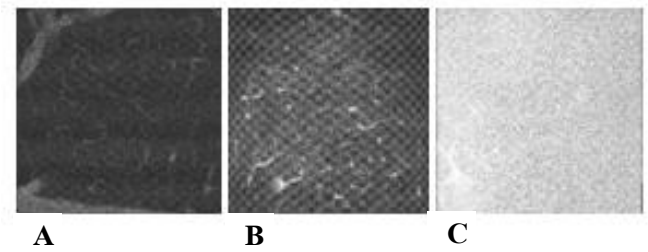


Figure 1: Representative samples of the original data in varying depths. A) First (1/85) image, portraying a horizontal line pattern. B) middle (42/85) image portraying an increase in background noise, as well as a diagonal grid pattern. C) Last image (85/85) portraying both an increase in background noise as well as both grid patterns.

Removal of depth-dependent noise

To construct a noise-removal pipeline, we applied several well-established noise removal techniques such as image smoothing using a Gaussian filter and salt-and-pepper noise reduction using a median kernel. We additionally used frequency-domain based noise reduction to remove the background grid patterns as well as background noise, and interestingly found that applying the frequency-based noise reduction approach was more effective when applied before and not after using the median kernel. We exploited the fact that adjacent images in the stack are related to one another to use a moving average (with the size of 3) to reduce residual noise after the pre-processing.

As the background grid patterns appeared to be a result of the image acquisition, they were indeed constant among all images in the stack. To remove those signals and address the possibility of slight changes in their features between images, we analysed the magnitude spectrum of the images and filtered relevant frequency ranges.

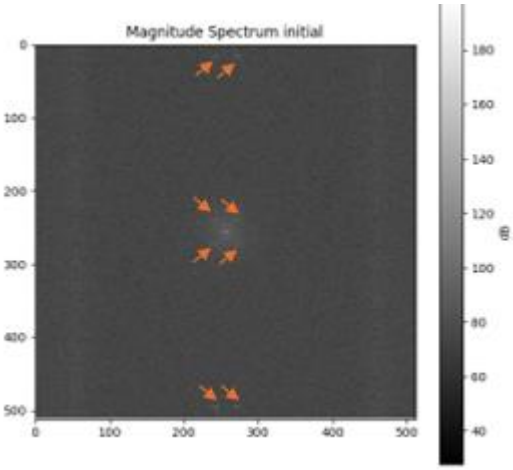


Figure 2: Representative image of the magnitude spectrum of the images in the stack. Peaks corresponding to the background grid noise pattern are marked with orange arrows.

After those initial pre-processing steps, we then used singular-value decomposition (SVD) to remove residual noise without losing information nor fine details. Following the SVD step, we applied an automated threshold (Otsu's threshold) to create a binary mask, from which we removed small holes and objects using functions from the CV.2 library (fig. 3) in a depth dependant manner.

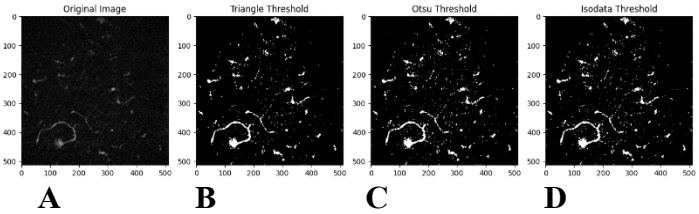


Figure 3: Demonstration of a representative image (42/85) after making and cleaning steps. A) Original image. B) Triangle threshold. C) Otsu's threshold. D) Isodata threshold.

Finally, to improve the mask and address the problem of continuity in some sections of the mask,

we used a sequence of erosion and dilation steps ("closing" and "opening" steps). This mask was then used to remove residual background noise by its construction with each image, to achieve the cleanest possible image for segmentation and skeletonization of the stack (Fig. 4).

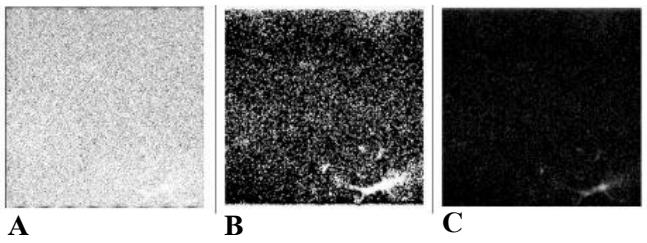


Figure 4: Image 85/85 of the stack. A) original image. B) Image after pre-processing and before addition of the mask. C) Image after the construction with the processed mask.

Code snippets from the main cleaning pipeline:

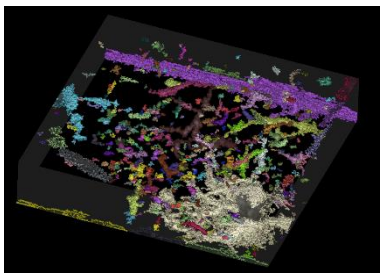
```
def noise_reduction_pipeline(zstack, rank=100):
    noise_reduced_zstack =
np.zeros(shape=zstack.shape)

    for i in range(zstack.shape[0]):
        cur_img = zstack[i, :, :]

        Processed_image = smooth_image(cur_img)

        central_line_filter(Processed_image)
        f_image =
diagonal_line_filter(np.fft.ifft2(np.fft.ifftshift(
f_image)))

        Processed_image =
np.fft.ifft2(np.fft.ifftshift(f_image))
        Processed_image = np.abs(Processed_image)

        U, s, V = linalg.svd(Processed_image)

        cur_sigmas = s.copy()
        cur_sigmas[rank:] = 0
        S = linalg.diagsvd(cur_sigmas, U.shape[1],
V.shape[1])
        Processed_image = U @ S @ V
Processed_image = low_pass_filter(Processed_image,
255).
        Processed image =
cv2.convertScaleAbs(Processed_image, alpha=(255.0 /
Processed_image.max()))

        Processed_image =
cv2.medianBlur(Processed_image, 3)

        ret_iso, isodata_threshold =
cv2.threshold(Processed_image, 0, 255,
cv2.THRESH_BINARY + cv2.THRESH_OTSU)

        isodata_threshold =
morphology.binary_closing(isodata_threshold)
        isodata_threshold =
morphology.binary_opening(isodata_threshold)
        removed_holes =
remove_small_holes(isodata_threshold,
area_threshold=50)

        if 0 <= i < 30:
            removed_objects =
remove_small_objects(removed_holes, min_size=30)
        elif 30 <= i < 60:
            removed_objects =
remove_small_objects(removed_holes, min_size=50)
        elif 60 <= 60 <= 85:
            removed_objects =
remove_small_objects(removed_holes, min_size=70)

        final_image = Processed_image *
removed_objects

        noise_reduced_zstack[i, :, :] = final_image

    return noise_reduced_zstack
```

Segmentation of the 3D stack

After achieving a satisfactory level of data cleaning given the low SNR of the original data, we proceeded to using watershed segmentation and adjusting its parameter to the images.

Watershed Algorithm is an image segmentation technique which is based on watershed transformation. The segmentation process will take the similarity with adjacent pixels of the image as an important reference to connect pixels with similar spatial positions and values. We later used connectivity-based labelling (based on binary Otsu's mask) to set the different vessels apart (Fig. 5).

Figure 5: 3D stack after using watershed segmentation labelled with connectivity-based labelling.



Interestingly, histogram equalization (which was attempted to use in the cleaning process) resulted in poorer outcomes regarding segmentation of the output stack, and therefore was omitted from the pre-processing for segmentation.

Skeletonization of the 3D segmented stack

For the skeletonization of the segmented stack, we used the skimage skeletonize function, modified to the needs of our data. Histogram equalization was used to overcome the poor contrast in the segmented stack. Additionally, due to the nature of the data and the result of the image cleaning process; some data

was lost – resulting in severed segments of the blood vessels, lacking continuousness. This output, when serves as an input to the skeletonization function, resulted in short and divided lines rather than a clear skeleton of the blood vessel (Fig.6).

Therefore, we used another pre-processing designated for the skeletonization, in which we dilated the segmented stack to connect noncontinuous parts of some of the segmented blood vessels and enable the skeletonization to create a continuous line. This technique appeared to improve the quality of the skeletonization yet enhances the residual background noise, as shown in here:

```
def Sekeltonization_pipline_stack_seg(zstack,
min_label = 3, show_net = False):
    equalized_stack = stack_hist_eq(zstack)

    masked_stack =
zstack_mask_otsu(equalized_stack)

    skel_processed_stack =
remove_small_holes(masked_stack, area_threshold=80)
    small_blobs_removed_skel =
remove_small_blobs(skel_processed_stack, 80)

    pre_skeletonization_segmentation =
zstack_dialation(np.uint8(small_blobs_removed_skel)
, kernel_size=(9, 9))

    small_blobs_removed_skel_stack =
np.uint8(pre_skeletonization_segmentation)
    skel_segmented_zstack =
stack_seg(small_blobs_removed_skel_stack)
    if show_net == False:
        segmented_binary_mask =
skel_segmented_zstack >= min_label

    elif show_net == True:
        segmented_binary_mask =
np.select([skel_segmented_zstack==0,
skel_segmented_zstack == 1, skel_segmented_zstack
==2], [np.zeros_like(skel_segmented_zstack),
np.ones_like(skel_segmented_zstack),
2*np.ones_like(skel_segmented_zstack)])

    else:
        skel_segmented_zstack =
small_blobs_removed_skel_stack

    skeletonize_zstack =
skeleton_stack(segmented_binary_mask)
    Skeletonization_labels =
stack_seg(skeletonize_zstack)

    return [skeletonize_zstack,
Skeletonization_labels]
```

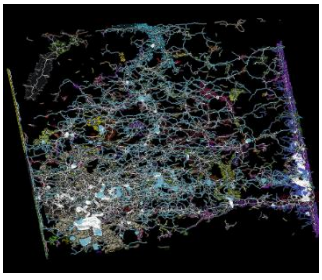


Figure 6: 3D skeleton based on watershed technique. Skeletonization was performed after dilation pre-processing steps to increase skeleton connectivity.

Summary and reflection

In this project, we encountered several limitations that impacted our ability to effectively process the data. One significant challenge was the SNR, which complicated the process of data cleaning and necessitated careful consideration during the removal of noisy data. This issue particularly affected the quality of skeletonization, resulting in non-continuous lines that hindered our ability to undertake advanced tasks such as identifying branches within the skeleton. Despite our efforts, achieving a satisfactory level of skeletonization suitable for branching analysis remained elusive.

To address these challenges and explore alternative methods, we considered different approaches for thresholding. Moving away from global Otsu's thresholding, we experimented with local thresholding techniques. However, this strategy proved problematic given the persistently low SNR, which undermined the effectiveness of local thresholding in accurately segmenting the desired features from noisy background.

Future analysis could potentially utilize manual skeletonization methods such as the medial axis transform, thinning of the vessels' outline via erosion or the h_minima function from the skimage library. Looking ahead, one promising direction for enhancing our workflow involves developing automated processes to replace manual parameter determination. Automation and advanced ML tools could streamline our methodology and potentially yield more consistent and reliable results.

In summary, this project has highlighted critical areas for improvement in image analysis methodologies, particularly when working with data of low SNR. Future analyses could benefit from refining skeletonization techniques, implementing automated parameter determination, and exploring novel approaches to overcoming noise-related challenges in image processing.