Deep Learning workflow for computational pathology

In this example, we show how to use MATLAB to manipulate very large images in the context of computational pathology (CPATH).

It is structured in two parts:

1. How to handle very large images such as whole slide images (WSIs)
2. How to pre- and post-process histology images.

Table of Contents

Part 1: Handling gigapixel-sized WSIs

The first part of this example shows how to read, display, explore, and organize WSIs (and their patches) in MATLAB. Thanks to the recently introduced `blockedImage` object, it is now possible to handle very large images without running out of memory. A `blockedImage` is an image made from discrete blocks (patches), which can be organized and managed using a `blockedImageDatastore` object and displayed using `bigimageshow`.

Create `blockedImageDatastore` at Specific level and Block Size

Create a `blockedImage`.

```
bim = blockedImage('tumor_091R.tif'); % This is a built-in image in MATLAB
```

Inspect resolution levels

```
tumorImage = bim;
levelSizeInfo = table((1:length(tumorImage.Size))', ...
    tumorImage.Size(:,1), ...
```

```
    tumorImage.Size(:,2), ...
    tumorImage.Size(:,1)./tumorImage.Size(:,2), ...
    'VariableNames',["Resolution Level" "Image Width" "Image Height" "Aspect Ratio"])
```

levelSizeInfo = 3×4 table

	Resolution Level	Image Width	Image Height	Aspect Ratio
1	1	5000	5358	0.9332
2	2	1250	1340	0.9328
3	3	625	670	0.9328

Create a `blockedImageDatastore`, specifying the resolution level and the blocksize.

```
bls = selectBlockLocations(bim,"Levels",2,"BlockSize",[512, 512]);
bimds = blockedImageDatastore(bim, "BlockLocationSet", bls);
```

Read all the blocks in the datastore.

```
b = readall(bimds)
```

Display the blocked image

```
montage(b)
```

Display the big image

```
h = bigimageshow(bim);
```

Display the big image with grid lines indicating blocks

```
h = bigimageshow(bim,...
      'GridVisible','on', 'GridLevel', 1,...
      'GridLineWidth', 2, 'GridColor','k','GridAlpha',0.3);
```

Part 2: Useful pre- and post-processing operations on WSIs in MATLAB

Since the goal of using deep learning techniques in CPATH is to produce solutions that are clinically translatable, i.e., capable of working across large patient populations, it is advisable to deal with some of the most likely WSI artifacts upfront, thereby increasing the abilities of the resulting model to generalize over image artifacts found in other test sets.

The second part of this example shows examples of preprocessing operations to handle commonly found artifacts in histopathology images as well as postprocessing morphological operations for improving the quality of the results at pixel level. Essentially, this example should help the medical image analysis community to create an image analysis pipeline for WSIs (and, as bonus, the ability to reproduce the code and examples described in a recent paper on this topic) using MATLAB.

It highlights the usefulness of MATLAB (and Image Processing Toolbox) functions such as:

- Image thresholding and filtering: **imbinarize**,**bwareafilt**, and **imlincomb**
- Morphological image processing operations: **imclose**, **imopen**, **imdilate**, **imerode**, **imfill**, and **strel**

- Feature extraction: **bwlabel** and **regionprops**
- Visualization: **montage**, **imoverlay**, **plot** and **rectangle**

Tissue Identification

In this section we show how to separate tissue from slide background in order to feed images of the tissue in to deep learning or machine learning models.

We start by reading and displaying the input image:

```
img1 = imread("download.png");
imshow(img1)
```

Next, we set up a mask for identifying tissue in the image (and distinguishing tissue pixels from background pixels) using an appropriately chosen threshold and overlay the mask on the original image:

```
mask = img1;
mask(:,:,1) = 0;
mask(:,:,3) = 0;
mask(:,:,2) = 255 * (img1(:,:,3) < 230);

img2 = img1;
img2 = imlincomb (0.3, mask, 0.7, img2);
imshow(img2)
```

Finally, we perform thresholding and morphological operations to produce a connected region of tissue pixels without any holes in it, and overlay the resulting region (in green) on the original image:

```
mask2 = rgb2gray(img1);
T = 200; % threshold (in a [0..255] range)
mask2_test = imbinarize(mask2, T/255.0);
% figure, imshow(mask2_test)
mask2_test_neg = imcomplement(mask2_test);
% figure, imshow(mask2_test_neg)

radius = 50; % radius of disk strel
se_oval = strel('disk', radius);
mask2_neg_test_close = imclose(mask2_test_neg, se_oval);
% figure, imshow(mask2_neg_test_close)
mask2_neg_test_close_hull = imfill(mask2_neg_test_close, 'holes');
% figure, imshow(mask2_neg_test_close_hull)

mask3 = uint8(zeros(size(img1)));
mask3(:,:,2) = 255 * mask2_neg_test_close_hull;
% figure, imshow(mask3)

img3 = img1;
img3 = imlincomb(0.3, mask3, 0.7, img3);

figure, imshow(img3)
```

The figure below shows:

- (left) initial image;
- (center) result of thresholding operation to separate tissue pixels from glass pixels;
- (right) result of applying hull filling to capture the full shape and size of the tissue and remove the slide background from further analysis down the pipeline.

```
figure, montage({img1, img2, img3}, 'Size', [1 3])
```

Tile Sampling

In this section we show how to create *tiles* or *patches* from large images (which are then fed into deep learning algorithms). We show three different sampling methods: regular grid, overlapping patches, and random patches.

Regular grid

```
img1 = imread("download.png");

figure
imshow(img1)

hold on
for i = 258:240:1699
    plot([i,i], [24,2184], 'Color','k', 'LineStyle','-');
end

for i = 24:240:2185
    plot([258,1698], [i,i], 'Color','k', 'LineStyle','-');
end

hold off
```

Overlapping patches

```
figure
imshow(img1)

hold on
for i = 258:288:1699
    plot([i, i], [24, 2184], 'Color','k', 'LineStyle','-');
    plot([i+80, i+80], [24, 2184], 'Color','k', 'LineStyle','-');
end

for i = 24:288:2185
    plot([258, 1698], [i, i], 'Color','k', 'LineStyle','-');
    plot([258, 1698], [i+80, i+80], 'Color','k', 'LineStyle','-');
end

hold off
```

Random patches

```
figure
```

```
imshow(img1)

hold on

seed = 42;
rng(uint16(seed)); % To ensure reproducibility
num_boxes = 54;

for i=1:num_boxes
    x = randi([258 1458], 1);
    y = randi([24 1944], 1);
    rectangle('Position',[x y 240 240], 'EdgeColor','k')
end

hold off
```

Prediction Cleaning

In this section we show how to use morphological operations to post-process the result of a deep learning image segmentation model and clean up spurious pixels that might be present. Our code also shows the differences between pixel-wise prediction and tile-wise prediction.

```
img1 = imread("download.png");

%% Working with the prediction
pred1 = imread("download.png");
pred1 = im2gray(pred1);

mask1 = uint8(zeros(size(img1)));
mask1(:,:,2) = pred1;

img4 = imlincomb (0.3, mask1, 0.7, img1);

figure, imshow(img4)
```

```
labeledImage = bwlabel(pred1);
measurements = regionprops(pred1, 'Area');
[num_areas, ~] = size(measurements);

mask2 = uint8(zeros(size(img1)));
num_pixels = numel(pred1);

mask2a = bwareafilt(imbinarize(pred1),[2000 num_pixels]);
% figure, montage({pred1, mask2a})

mask2a_hull = imfill(mask2a, 'holes');

mask2b = uint8(zeros(size(img1)));
mask2b(:,:,2) = 255*uint8(mask2a_hull);

img5 = imlincomb(0.3, mask2b, 0.7, img1);
```

```
figure, imshow(img5)
```

```
% Note: The block below runs fine by itself (e.g., as a separate script or
% on the command window), but NOT as part of a Live Script. I've sent a
% note to MathWorks about this.

squares_x = [4 4 4 5 4 5];
squares_y = [2 3 6 6 7 7] - 2;

figure
imshow(img1)


hold on
nx = numel(squares_x);
ny = numel(squares_y);

for i = 258:240:1699
    plot([i, i], [24, 2184], 'Color','k', 'LineStyle','-');
end

for i = 24:240:2185
    plot([258, 1698], [i, i], 'Color','k', 'LineStyle','-');
end

for i = 1:nx
    x = squares_x(i)*240+24;
    y = squares_y(i)*240+258;
    rectangle('Position',[x y 240 240], 'EdgeColor','g', 'FaceColor', [0, 1, 0, 0.3])
end
```

Additional morphological operations for image post-processing

In this section we show how to use MATLAB to perform additional morphological post-processing operations and prepare useful plots.

```
img1 = imread("download.png");
img2 = rgb2gray(img1);
T = 200; % threshold (in a [0..255] range)
mask1_test = imbinarize(img2, T/255.0);
mask1_test_neg = imcomplement(mask1_test);
kernel = strel('disk', 11);

mask2 = imdilate(mask1_test_neg, kernel);
mask2_hull = imfill(mask2, 'holes');
mask3 = imerode(mask1_test_neg, kernel);
mask3_hull = imfill(mask3, 'holes');
mask4 = imopen(mask1_test_neg, kernel);
mask4_hull = imfill(mask4, 'holes');
mask5a = imdilate(mask1_test_neg, kernel);
mask5a_hull = imfill(mask5a, 'holes');

mask5b = imclose(mask1_test_neg, kernel);
```

```
mask5b_hull = imfill(mask5b, 'holes');

mask6 = mask2_hull - mask5b_hull;
mask6_fig = uint8(zeros(size(img1)));
mask6_fig(:,:,1) = 255*uint8(mask6);
mask6_fig(:,:,2) = 0;
mask6_fig(:,:,3) = 0;
fig7a_aux = uint8(zeros(size(img1)));
fig7a_aux(:,:,2) = 255* mask2_hull;
fig7a = img1;
fig7a = imlincomb(0.3, fig7a_aux, 0.7, fig7a);
fig7b_aux = uint8(zeros(size(img1)));
fig7b_aux(:,:,2) = 255* mask3_hull;
fig7b = img1;
fig7b = imlincomb(0.3, fig7b_aux, 0.7, fig7b);
fig7c_aux = uint8(zeros(size(img1)));
fig7c_aux(:,:,2) = 255* mask4_hull;
fig7c = img1;
fig7c = imlincomb(0.3, fig7c_aux, 0.7, fig7c);
fig7d_aux = uint8(zeros(size(img1)));
fig7d_aux(:,:,2) = 255* mask5b_hull;
fig7d = img1;
fig7d = imlincomb(0.3, fig7d_aux, 0.7, fig7d);
fig7d_overlay = imoverlay(fig7d, mask6, 'red');
figure
montage({fig7a, fig7b, fig7c, fig7d_overlay}, 'Size', [2 2]) % Fig 7 in the paper
```