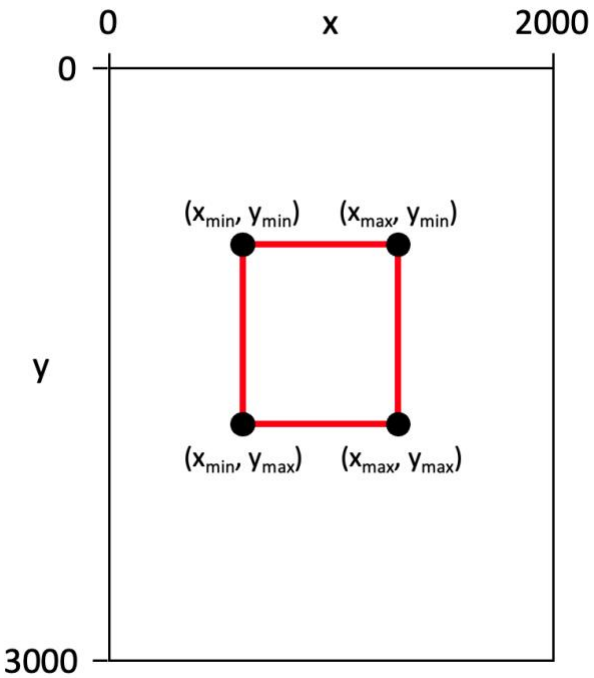
EMBED ROIs

Contents

Overview:

ROIs in EMBED were collected by mapping screensave images (generated when any annotations are made by a radiologist) to their original source images. During this process, ROIs were mapped back to both the original image (**a primary match**) and its corresponding C-View or 2D image (**a secondary match**). Each ROI is given in the form $[y_{min}, x_{min}, y_{max}, x_{max}]$.



Columns:

- **`num_roi: int`**
  num_roi contains an integer indicating the number of ROIs associated with an image (including both primary and secondary matches)

- **`DCM_ROI_coords: str`**
  DCM_ROI_coords contains a nested list of ROI corner coordinates saved as a string. These coordinates are mapped to the orientation of the DICOM images and take the following format:

  "[[[ROI_A1], [[ROI_B1]]]"

  Above shows an **ROI_coords** string with 2 ROIs from 2 different sources.

  Nesting levels:
  - First level: The first level brackets enclose the entire list.
  - Second level: The second level nesting indicates the source of the ROI. If there is only one image it will take the form "[[SOURCE_A]]" or "[[SOURCE_A], [SOURCE_B]]" with two source images. This level can be discarded if you don't need to link it to the original ROI_SSC.
  - Third level: The third level nesting indicates each individual ROI. An image with 1 ROI from 1 source would be encoded as "[[[ROI_A1]]]" and an image with 2 ROIs from 1 source would be encoded as "[[[ROI_A1], [ROI_A2]]]". Each ROI takes the form y_{min}, x_{min}, y_{max}, x_{max}.

- **`PNG_ROI_coords: str`**
  PNG_ROI_coords contains a nested list of ROI corner coordinates saved as a string. These coordinates are mapped to the orientation of the PNG images and follows the same formatting conventions as DCM_ROI_coords.

- **`ROI_match_level: str`**
  ROI_match_level contains a nested list of ROI corner coordinates saved as a string in the following format:

> **“[[[ROI_A1$_{Match_Level}$], [[ROI_B1$_{Match_Level}$]]]”**
>
> Above shows an **ROI_match_level** string with 2 ROIs from 2 different sources. The nesting syntax exactly matches **ROI_coords**. Each match level can be either ‘1’ or ‘2’.

- **`ROI_source: str`**
  This column indicates the path of the ROI_SSC that each ROI was derived from. It can be ignored unless you need to map 2D or C-View ROIs back to the original ROI_SSC image that they were derived from.

- **`SSC_ROI_flipped: str`**
  This column indicates whether the ROI_SSC laterality was flipped to match the orientation of the matched image. It can be ignored unless you need to map 2D or C-View ROIs back to the original ROI_SSC image that they were derived from.

- **`SSC_ROI_dest: str`**
  This column is only populated for ROI_SSC images. It indicates which 2D or C-view images the SSC ROIs were mapped to. It can be ignored unless you need to map 2D or C-View ROIs back to the original ROI_SSC image that they were derived from.

- **`SSC_match_level: str`**
  This column indicates whether the ROI_SSC laterality was flipped to match the orientation of the matched image. It can be ignored unless you need to map 2D or C-View ROIs back to the original ROI_SSC image that they were derived from.

- **`PNG_flipped: str`**
  This column is currently only populated for images with an ROI. It indicates whether the PNG was horizontally flipped during the DICOM to PNG conversion.

Example:

The ROI columns were designed to be parsed back into lists before use.

Example Functions:

```python
def parse_roi(roi_coords: str):
    """
    function to convert our ROI_coords string into a nested list.
    since we don't care about mapping back to the original ROI_SSC image
    we can discard one level of nesting to make them easier to use
    """
    # remove "][)(," symbols from the string
    roi_coords = roi_coords.translate({ord(c): None for c in "][)(,"})

    # split the list on whitespace, map each value to an
    # integer, and send it to a list
    # we now have a list like this
    # [ymin0, xmin0, ymax0, xmax0, ymin1, ...]
    flat_roi_list = list(map(int, roi_coords.split()))

    # we need to reformat it to
    # [[ymin0, xmin0, ymax0, xmax0], [ymin1, ...]]
    # get an empty list
    out_roi_list = []

    # iterate over the number of rois (number of coords mod 4)
    for i in range(len(flat_roi_list) // 4):
        # for each, append the relevant 4 flat_roi_list
        # indices to the out_roi_list
        out_roi_list.append(flat_roi_list[4*i:4*i+4])

    return out_roi_list

def parse_match_level(match_level: str):
    """
    function to convert our ROI_match_level string into a nested list.
    since we don't care about mapping back to the original ROI_SSC image
    we can discard one level of nesting to make it easier to use
    """
    # remove "][' " symbols from the string
    match_level = match_level.translate({ord(c): None for c in "]['"})

    # split the list on commas, map each value to an integer
    # and sent it to a list
    match_level_list = list(map(int, match_level.split()))

    return match_level_list
```

```python
def plot_rois(ax, roi_list: list, match_level_list: list):
    """
    function to plot the rois on the given matplotlib axes
    """
    # set our dicts to color primary/secondary ROIs differently
    color_dict = {1: 'xkcd:bright green', 2: 'xkcd:bright red'}
    label_dict = {1: 'Primary ROI', 2: 'Secondary ROI'}

    # zip the rois and their match levels together
    for roi, match_level in zip(roi_list, match_level_list):
        # unpack our ROI values
        ymin, xmin, ymax, xmax = roi

        # format the roi into a patch
        roi_patch = pat.Rectangle(
            (xmax, ymax),
            xmin - xmax,
            ymin - ymax,
            edgecolor=color_dict[match_level],
            fc='None',
            label=label_dict[match_level]
        )

        # add the patch to the axes
        ax.add_patch(roi_patch)



def plot_image(image, roi_list: list, match_level_list: list):
    """
    function to take a cv2 image and a list of rois and match levels
    and plot them
    """
    # get a figure and axis
    fig, ax = plt.subplots(1, 1, dpi=150)

    # plot our image on the axes
    ax.imshow(image)

    # plot our rois on the axes
    plot_rois(ax, roi_list, match_level_list)

    # get a legend
    ax.legend()

    # show the image
    fig.show()
```

Example Usage:

```
# load the metadata CSV
meta_path = "/PATH/TO/metadata.csv"
meta_df = pd.read_csv(meta_path)

# get a boolean mask to filter out images with no ROIs
meta_mask = (meta_df.num_roi > 0) & (meta_df.FinalImageType == "2D")

# apply the mask to the dataframe and randomly sample 10 cases
meta_sample_df = meta_df[meta_mask].sample(10)

# iterate over our sample dataframe
for i, data in meta_sample_df.iterrows():
    # load our image from the png path
    # alternately load DICOMs with pydicom and extract their pixel_arrays
    image = cv.imread(data.png_path)

    # parse our roi string into a list (use DCM_ROI_coords if loading DCMs)
    roi_list = parse_roi(data.PNG_ROI_coords)

    # parse our match level string into a list
    match_level_list = parse_match_level(data.ROI_match_level)

    # plot our image using the helper function
    plot_image(image, roi_list, match_level_list)
```

This code produces images like the one below.