eda

August 28, 2023

[1]:
```
# this file is a modified version of the original:
# https://www.kaggle.com/code/leonidkulyk/eda-hubmap-hhv-interactive-annotations
```

#

HuBMAP - Hacking the Human Vasculature - Exploratory Data Analysis

Segment instances of microvascular structures from healthy human kidney tissue slides.

#

(_) Overview

Goal: The goal of the competition is to develop a model that can segment instances of microvascular structures, such as capillaries, arterioles, and venules, in 2D PAS-stained histology images from healthy human kidney tissue slides.

Importance: Automating the segmentation of microvasculature structures will help improve researchers' understanding of how blood vessels are arranged in human tissues. This knowledge is crucial for studying the interaction, organization, and specialization of cells in the body.

Context: The competition is aligned with the efforts of the Human BioMolecular Atlas Program (HuBMAP), which aims to map healthy cells in the human body. The Vasculature Common Coordinate Framework (VCCF) is a navigation system that uses blood vasculature to map cellular locations. However, there are gaps in knowledge about microvasculature, and automating segmentation can help fill those gaps and contribute to the development of the VCCF and a Human Reference Atlas (HRA).

Research Focus: HuBMAP researchers are studying the connections between cells throughout the body using advanced molecular and cellular biology technologies. By segmenting microvasculature arrangements, machine learning insights can augment their understanding of how these small vessels are distributed, leading to a better understanding of the relationships between cells and their impact on human health.

#

Table of contents

1.1 Files and Field Descriptions

```
    <ul>
        <li><a href="#1.2" target="_self" rel=" noreferrer nofollow">1.2 Whole Slide Images met
    </ul>
    <ul>
        <li><a href="#1.3" target="_self" rel=" noreferrer nofollow">1.3 Each image metadata</a
    </ul>
    <ul>
        <li><a href="#1.4" target="_self" rel=" noreferrer nofollow">1.4 Tiles visualisation</a
    </ul>
</li>
```

#

0. Import all dependencies

[2]:
```
import os
import json
from PIL import Image
from collections import Counter

import numpy as np
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
import tifffile as tiff
import matplotlib.pyplot as plt
```

[42]:
```
from torch.utils.data import Dataset
from tqdm import tqdm, notebook
import yaml
import cv2
```

[3]:
```
class CFG:
    img_path_template: str = "/kaggle/input/
 ↪hubmap-hacking-the-human-vasculature/train/{}.tif"
```

[4]:
```
def get_cartesian_coords(coords, img_height):
    coords_array = np.array(coords).squeeze()
    xs = coords_array[:, 0]
    ys = -coords_array[:, 1] + img_height

    return xs, ys
```

[27]:
```
SAMPLE_SBMISSION_PATH = "/kaggle/input/hubmap-hacking-the-human-vasculature/
 ↪sample_submission.csv"
TILE_META_PATH = "/kaggle/input/hubmap-hacking-the-human-vasculature/tile_meta.
 ↪csv"
```

```
WSI_META_PATH = "/kaggle/input/hubmap-hacking-the-human-vasculature/wsi_meta.
 ↪csv"
ANNOTATION_PATH = "/kaggle/input/hubmap-hacking-the-human-vasculature/polygons.
 ↪jsonl"
DATASET_CONFIG_PATH = "/kaggle/working/classes_config.csv" # Custom file

# Folders
TRAIN_PATH = "/kaggle/input/hubmap-hacking-the-human-vasculature/train"
TEST_PATH = "/kaggle/input/hubmap-hacking-the-human-vasculature/test"
```

[55]:
```
def plot_annotated_image(image_dict, scale_factor: int = 1.0) -> None:
    array = tiff.imread(CFG.img_path_template.format(image_dict["id"]))
    img_example = Image.fromarray(array)
    annotations = image_dict["annotations"]

    # create figure
    fig = go.Figure()

    # constants
    img_width = img_example.size[0]
    img_height = img_example.size[1]
    print(f'W =>{img_width}, H => {img_height}')

    # add invisible scatter trace
    fig.add_trace(
        go.Scatter(
            x=[0, img_width],
            y=[0, img_height],
            mode="markers",
            marker_opacity=0
        )
    )

    # configure axes
    fig.update_xaxes(
        visible=False,
        range=[0, img_width]
    )

    fig.update_yaxes(
        visible=False,
        range=[0, img_height],
        # the scaleanchor attribute ensures that the aspect ratio stays constant
        scaleanchor="x"
    )

    # add image
```

```
    fig.add_layout_image(dict(
        x=0,
        sizex=img_width,
        y=img_height,
        sizey=img_height,
        xref="x", yref="y",
        opacity=1.0,
        layer="below",
        sizing="stretch",
        source=img_example
    ))

    # add polygons
    for annotation in annotations:
        name = annotation["type"]
        xs, ys = get_cartesian_coords(annotation["coordinates"], img_height)
        fig.add_trace(go.Scatter(
            x=xs, y=ys, fill="toself",
            name=name,
            hovertemplate="%{name}",
            mode='lines'
        ))

    # configure other layout
    fig.update_layout(
        width=img_width * scale_factor,
        height=img_height * scale_factor,
        margin={"l": 0, "r": 0, "t": 0, "b": 0},
        showlegend=False
    )

    # disable the autosize on double click because it adds unwanted margins␣
 ↪around the image
    # and finally show figure
    fig.show(config={'doubleClick': 'reset'})
```

[ ]:

[47]:
```
# https://www.kaggle.com/code/mersico/hubmap-eda-pycocotools-submission
class HuBMAPDataset(Dataset):
    def __init__(self,
                 annotation_path: str,
                 image_path: str,
                 config_path: str):
        self.__image_path = image_path
        self.__samples = self.parse_jsonl(annotation_path)
        self.__config = self.load_config(config_path)
```

```
    def __len__(self) -> int:
        return len(self.__samples)

    def __getitem__(self, idx: int) -> tuple[np.ndarray, np.ndarray]:
        image, id = self.__get_image(idx)
        mask = self.__get_mask(idx)
        # image = torch.tensor(image, dtype=torch.float32).permute(2, 0, 1)
        # mask = torch.tensor(mask, dtype=torch.float32)
        return image, id, mask

    @staticmethod
    def parse_jsonl(path: str) -> list[dict, ...]:
        with open(path, 'r') as json_file:
            jsonl_labels = [
                json.loads(line)
                for line in notebook.tqdm(
                    json_file, desc="Processing polygons", total=1633
                )
            ]
        return jsonl_labels

    @staticmethod
    def load_config(path: str) -> dict:
        with open(path, mode="r") as f:
            data = yaml.load(stream=f, Loader=yaml.SafeLoader)
        return data

    def __get_image_path(self, id: str) -> str:
        path = os.path.join(
            self.__image_path, f"{id}.tif"
        )
        return path

    def __get_image(self, idx: int) -> np.ndarray:
        id = self.__samples[idx]["id"]
        image_path = self.__get_image_path(id)
        image = Image.open(image_path)
        image = np.asarray(image)
        return image, id

    def __get_mask(self, idx: int) -> np.ndarray:
        mask = np.zeros((512, 512), dtype=np.uint8)
        annotations = self.__samples[idx]["annotations"]

        for vessel in annotations:
            vessel_type = vessel["type"]
```

```
            config = self.__config[vessel_type]

            if config["apply_mask"]:
                coordinates = np.array(vessel["coordinates"])
                mask = cv2.fillPoly(
                    mask, pts=coordinates,
                    color=config["rgb"]
                )
        return mask
```

[38]:
```
# https://www.kaggle.com/code/mersico/hubmap-eda-pycocotools-submission
dataset_config = {
    "background": {
        "apply_mask": None,
        "label": 0,
        "rgb": (0, 0, 0),
        "loss_weight": None
    },
    "blood_vessel": {
        "apply_mask": True,
        "label": 1,
        "rgb": (255, 8, 8),
        "loss_weight": None
    },
    "glomerulus": {
        "apply_mask": True,
        "label": 2,
        "rgb": (8, 12, 255),
        "loss_weight": None
    },
    "unsure": {
        "apply_mask": True,
        "label": 3,
        "rgb": (8, 255, 20),
        "loss_weight": None
    }
}
def write_config(data: dict[dict, ...], path: str) -> None:
    with open(path, mode="w") as f:
        yaml.safe_dump(stream=f, data=data)

write_config(dataset_config, DATASET_CONFIG_PATH)
```

[48]:
```
dataset = HuBMAPDataset(ANNOTATION_PATH, TRAIN_PATH, DATASET_CONFIG_PATH)
```

```
Processing polygons:   0%|          | 0/1633 [00:00<?, ?it/s]
```

#

1. Data overview

Our goal in this competition is to locate microvasculature structures (blood vessels) within human kidney histology slides.

The competition data comprises tiles extracted from 5 Whole Slide Images (WSI) split into 2 datasets. Tiles from Dataset 1 have annotations that have been expert reviewed. Dataset 2 comprises the remaining tiles from these same WSIs and contain sparse annotations that have not been expert reviewed.

Tiles from Dataset 1 have annotations that have been expert reviewed. Dataset 2 comprises the remaining tiles from these same WSIs and contain sparse annotations that have not been expert reviewed.

- All of the test set tiles are from Dataset 1.
- 1 of the WSIs make up the training set, 2 WSIs make up the public test set, and 1 WSI makes up the private test set.
- The training data includes Dataset 2 tiles from the public test WSI, but not from the private test WSI.

We should expect 14 unique sources of WSIs where 5 have been used for annotations by either a expert or non expert which then get put into dataset 1 and 2 respectively, and the other 9 correspond to WSIs for dataset 3 which have no annotations. Although we should expect 14 we see that there is only thirteen. A keen eye would see that number 5 is missing. This is probably because the 5th WSI belongs to the first dataset and was removed from the dataset to be used as the test set.

##

1.1 Files and Field Descriptions

{train|test}/ Folders containing TIFF images of the tiles. Each tile is 512x512 in size.

polygons.jsonl Polygonal segmentation masks in JSONL format, available for Dataset 1 and Dataset 2. Each line gives JSON annotations for a single image.

wsi_meta.csv Metadata for the Whole Slide Images the tiles were extracted from.

tile_meta.csv Metadata for each image.

##

1.2 Whole Slide Images metadata

Description

wsi_meta.csv Metadata for the Whole Slide Images the tiles were extracted from.

- source_wsi Identifies the WSI.
- age, sex, race, height, weight, and bmi demographic information about the tissue donor.

Let's load the wsi_meta.csv file and display it.

[6]:
```
wsi_meta_df = pd.read_csv("/kaggle/input/hubmap-hacking-the-human-vasculature/
↪wsi_meta.csv")
```

[7]:
```
wsi_meta_df
```

[7]:

	source_wsi	age	sex	race	height	weight	bmi
0	1	58	F	W	160.0	59.0	23.0
1	2	56	F	W	175.2	139.6	45.5
2	3	73	F	W	162.3	87.5	33.2
3	4	53	M	B	166.0	73.0	26.5

As can be seen above, only 4 wsi are present in the available data out of the 5 mentioned.

##

1.3 Each image metadata

Description

tile_meta.csv Metadata for each image.

- source_wsi Identifies the WSI this tile was extracted from.
- {i|j} The location of the upper-left corner within the WSI where the tile was extracted.
- dataset The dataset this tile belongs to, as described above.

Let's load the tile_meta.csv file and see what it looks like.

[8]:
```
tile_meta_df = pd.read_csv("/kaggle/input/hubmap-hacking-the-human-vasculature/
↪tile_meta.csv")
```

[9]:
```
tile_meta_df.head()
```

[9]:

	id	source_wsi	dataset	i	j
0	0006ff2aa7cd	2	2	16896	16420
1	000e79e206b7	6	3	10240	29184
2	00168d1b7522	2	2	14848	14884
3	00176a88fdb0	7	3	14848	25088
4	0033bbc76b6b	1	1	10240	43008

[10]:
```
tile_meta_df.describe()
```

[10]:

	source_wsi	dataset	i	j
count	7033.000000	7033.000000	7033.000000	7033.000000
mean	8.205744	2.707806	14296.542585	23227.014930
std	4.004251	0.571724	5946.054578	10644.032931
min	1.000000	1.000000	1536.000000	2560.000000
25%	6.000000	3.000000	9728.000000	15265.000000
50%	9.000000	3.000000	13824.000000	22528.000000
75%	12.000000	3.000000	18944.000000	29656.000000

```
max       14.000000     3.000000  30208.000000  52753.000000
```

Let's see how many WSI there are.

[11]:
```
print("Unique source WSIs --", list(np.unique(tile_meta_df.source_wsi)))
print("Number of unique source WSIs --", len(np.unique(tile_meta_df.
  ↪source_wsi)))
```

```
Unique source WSIs -- [1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14]
Number of unique source WSIs -- 13
```

As you can see from the output above, the 5th wsi is missing.

Now let's see how many datasets exist.

[12]:
```
print("Unique datasets --", list(np.unique(tile_meta_df.dataset)))
print("Number of unique datasets --", len(np.unique(tile_meta_df.dataset)))
```

```
Unique datasets -- [1, 2, 3]
Number of unique datasets -- 3
```

[13]:
```
swsi_count = Counter(tile_meta_df.source_wsi)

fig = px.bar(
    x=list(swsi_count.values()), y=list(swsi_count.keys()),
    color_discrete_sequence=['darkslateblue'],
    orientation='h', height=700
)
fig.update_layout(
    title={
        'text': "Source WSI Composition",
        'y':0.95,
        'x':0.5,
        'xanchor': 'center',
        'yanchor': 'top'
    },
    xaxis_title="Frequency", yaxis_title="Source WSI"
)
fig.show()
```

Most WSIs consists of 600 tiles, which is very, very much :).

##

1.4 Tiles visualisation

Each Whole Slide Image (WSI) is divided into tiles (as seen above, basically 600 pieces). For some of these tiles, polygonal segmentation masks are given in the polygons.jsonl file.

polygons.jsonl description – Polygonal segmentation masks in JSONL format, available for Dataset 1 and Dataset 2. Each line gives JSON annotations for a single image with:

- id Identifies the corresponding image in train/
- annotations A list of mask annotations with:
- type Identifies the type of structure annotated:
  - blood_vessel The target structure. Our goal in this competition is to predict these kinds of masks on the test set.
  - glomerulus A capillary ball structure in the kidney. These parts of the images were excluded from blood vessel annotation. You should ensure none of your test set predictions occur within glomerulus structures as they will be counted as false positives. Annotations are provided for test set tiles.
  - unsure A structure the expert annotators cannot confidently distinguish as a blood vessel.
- coordinates A list of polygon coordinates defining the segmentation mask.

Let's load these annotations and see what they look like in an example.

[14]:
```
with open('/kaggle/input/hubmap-hacking-the-human-vasculature/polygons.jsonl',␣
↪'r') as json_file:
    json_list = list(json_file)

tiles_dicts = []
for json_str in json_list:
    tiles_dicts.append(json.loads(json_str))
```

[15]:
```
print("Count of annotated tiles --", len(tiles_dicts))
```

```
Count of annotated tiles -- 1633
```

As you can see from the output above, the number of annotated tiles is over 4 times smaller for the total number of tiles (7033 tiles / 1633 annotated tiles).

Now, let's go directly to the visualization. Take for example a tile with id 9b9349a10d8d. In the polygons.jsonl list, this tile has an index of 1000.

[16]:
```
tiles_dicts[1000]["id"]
```

[16]:
```
'9b9349a10d8d'
```

[17]:
```
tile_meta_df[tile_meta_df.id == "9b9349a10d8d"]
```

[17]:

	id	source_wsi	dataset	i	j
4299	9b9349a10d8d	1	2	7680	5120

[ ]:
```

```

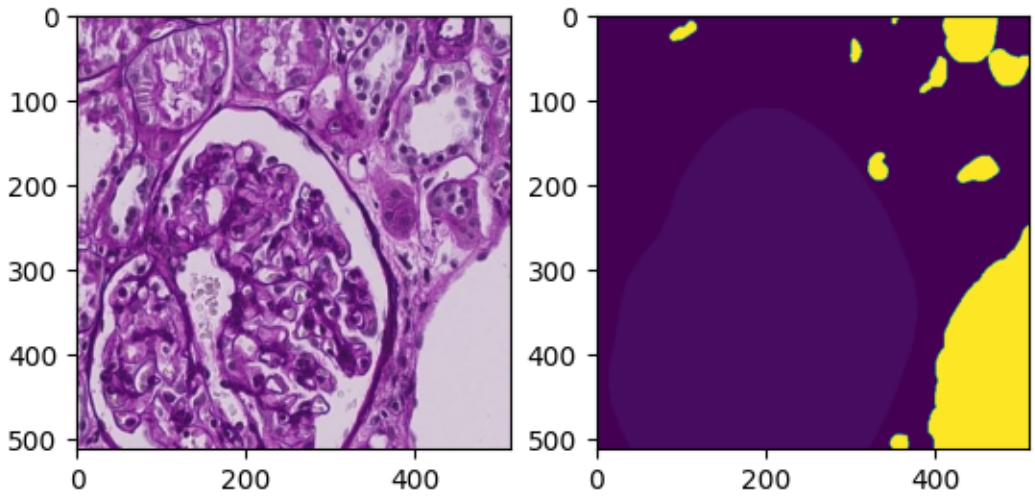
[45]:
```
len(dataset[0])
```

[45]:
```
2
```

[53]:
```
image, id, mask = dataset[1000]
print(id)
fig, (ax1, ax2) = plt.subplots(1, 2)

ax1.imshow(image)
ax2.imshow(mask)
plt.show()
```

```
9b9349a10d8d
```



[57]:
```
plot_annotated_image(tiles_dicts[1000])
```

```
W =>512, H => 512
```

Take for example another tile with id 0870e4f9d580. In the polygons.jsonl list, this tile has an index of 50.

[19]:
```
tiles_dicts[50]["id"]
```

[19]:
```
'0870e4f9d580'
```

[20]:
```
tile_meta_df[tile_meta_df.id == "0870e4f9d580"]
```

[20]:

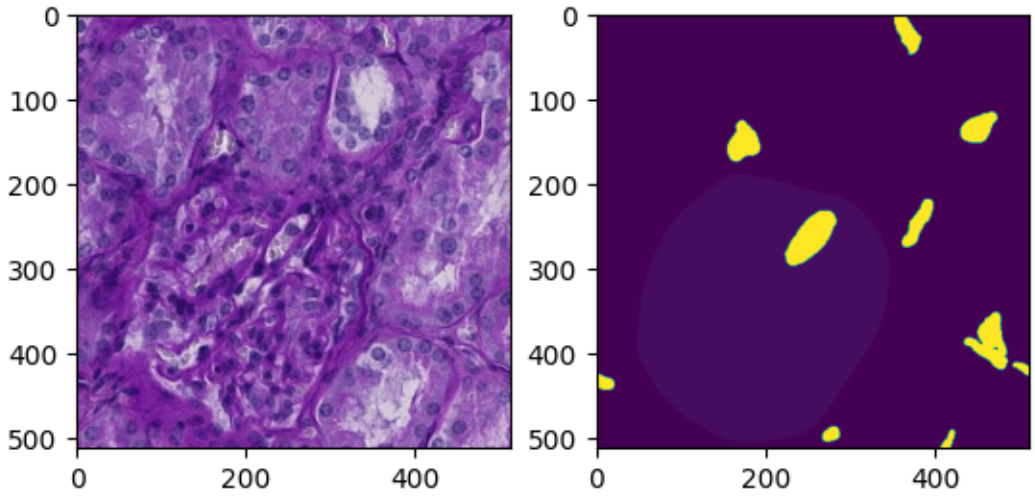
	id	source_wsi	dataset	i	j
253	0870e4f9d580	2	2	24576	23040

[54]:
```
image, id, mask = dataset[50]
print(id)
fig, (ax1, ax2) = plt.subplots(1, 2)

ax1.imshow(image)
```

```
ax2.imshow(mask)
plt.show()
```

0870e4f9d580



[56]:
```
plot_annotated_image(tiles_dicts[50])
```

W =>512, H => 512

[22]:
```
# (   ) WORK STILL IN PROGRESS
```

#

Thank You!

Thank you for taking the time to read through my notebook. I hope you found it interesting and informative. If you have any feedback or suggestions for improvement, please don't hesitate to let me know in the comments. If you liked this notebook, please consider upvoting it so that others can discover it too. Your support means a lot to me, and it helps to motivate me to create more content in the future. Once again, thank you for your support, and I hope to see you again soon!