Auto-annotation of Pathology Images

Final Report of Capstone

Chao Huang (ch3474), Yihao Li (yl4326),
Yufeng Ma (ym2764), Shuo Yang (sy2886), Xiaoyun Zhu (xz2884)

December 18, 2020

Abstract

To relieve pathologists' burden of manually annotating medical images, we designed and implemented an iterative human-in-the-loop(HITL)-based segmentation pipeline in this capstone project, which can automatically annotate the image and have pathologists manually correct it for further fine-tuning. To deal with the extreme lack of annotated data, we proposed an innovative method of generating synthetic images with minimal real-world labels, then trained U-Net and Mask R-CNN models completely on these synthetic data. These models performed unexpectedly well when tested on the manually labeled data. To further fine-tune these pre-trained models with pathologists' instruction, we implemented a full-stack web app as the pipeline's core component. This app has been deployed onto the AWS cloud so that the pathologists can provide useful feedback for model predictions using a user-friendly GUI.

1 Introduction

1.1 Background

Examination of kidney biopsy slides is essential for deciding whether a kidney is suitable for transplantation. The traditional evaluation is conducted by expert pathologists visually under microscopes. This approach is time-consuming, labor-intensive, and inconsistent across different clinicians. Therefore, tools that can automatically perform renal biopsy analysis quickly and consistently are of significant value for healthcare professionals.

In the past decade, we have witnessed huge progress in artificial neural networks and their infrastructure, and Deep Learning (DL) now enables computers to read digital images and learn relevant features through a network of artificial neurons in an end-to-end manner. Thanks to the proliferation of whole slide scanners and digital pathology, DL techniques are becoming increasingly prevalent in biomedical image processing. With past successful applications, DL-based automatic annotation tools may be leveraged as complements for pathologists to improve their workflow efficiencies as well.

Based on the model and the segmentation masks it generated, a wide array of AI techniques can be further deployed as downstream applications to meet different academic/clinical demands, e.g., automatic diagnosis, model-support augmented notations for labeling, etc.

1.2 Problem Statement

The goal of this project is to construct a HITL platform to operate segmentation tasks on renal pathology whole slide images (WSI). This platform is integrated with some DL models to make inferences on unseen slides on the back-end, and segmentation results can be rendered on the front-end. This platform will enable pathologists to make and save further adjustments on the existing model generated annotations with a back-end database and front-end interactive annotation tools. Those human-corrected annotations will then be used for fine-tuning the DL models.

1.3 Literature Review

Altini et al. proposed SegNet and Deeplab v3+ to address the segmentation and classification of glomeruli. They trained 26 WSIs containing 2344 non-sclerotic glomeruli and 428 sclerotic glomeruli. Their work achieved 0.924 as the best F-score for non-sclerotic glomeruli and 0.730 as the best F-score for sclerotic glomeruli[1]. Jayapandian et al. evaluated the application of U-Net on the segmentation of histologic structures in kidney biopsies with multiple stains. The authors generated 30,048 annotations using 459 WSIs from 38 pathology laboratories. The slides include four kinds of stains: H&E, PAS, TRI, and SIL. Their approach obtained comparable performances across four stains, with the best result on PAS stained WSIs[2]. Besides this group, Hermsen[3], and Bouteldja[4]'s group also deployed U-Net in recent two years. Kawazoe et al. applied Faster R-CNN to detect glomeruli on multi-stained human biopsy slides. They utilized 33,000 annotated glomeruli from 800 WSIs. Their Faster R-CNN method produced the F-score of 0.925 in PAS stain. They also demonstrated that equally high performance could be obtained not only for PAS stain but also for PAM (0.928), MT (0.898), and Azan (0.877) stains[5].

1.4 Project Specific Settings

Different from normal image segmentation problems, this project was formulated under a unique setting:

- Data: The major roadblock we faced was the extreme lack of annotated data, as we only had 10 partially annotated Whole Slide Images (WSI) from our pathologist team at the end of this project. This also explained why this project's primary objective was set as alleviating the burden of manual annotation for pathologists.
- Objectives: Unlike traditional object detection tasks, our pathologist teammates suggested that the importance of capturing correct contours is way more important than correctly classifying them.

In order to combat the data scarcity issue, we came up with several workarounds:

- We proposed a unique and complete pipeline for automated pathology image analysis in Section 1.5.
- We introduced a public dataset from existing work in Section 2.1.
- We proposed a new image synthesizing method in Section 2.2.
- we used a unique offline evaluation setting to match the end-users' perception in Section 4.1.

1.5 Pipeline Design

As depicted in Figure 1, the pipeline we proposed can be split into two parts: an initialization part and an iterative part. In the initialization part, we trained the segmentation engine to detect four histologic structures (namely glomeruli, interstitium, tubules, and vessels, as illustrated in Figure 2) on pre-annotated or newly-synthesized WSIs of kidney biopsy. Once the training is finished, models will be passed into the iterative part. In each iteration, models would first predict annotations on unseen WSIs, which will be rendered on an interactive user interface. The feedback from pathologists, i.e., corrections on auto-generated annotations, can be collected and saved in the backend. Collected feedback will then be utilized to train the next generation of DL models, which will improve the users' experience in turn.

With system scalability in mind, the whole platform is deployed on an AWS EC2 instance. This guarantees that pathologists can efficiently and collaboratively analyze kidney biopsies through a web app without worrying about the storage issues of large biopsy slides or lack of local computing hardware. The detail about this deployment is discussed in section 5.

The structured code repository for our project can be founded on GitHub (see `https://github.com/perottelab`[1]).

[1]Three correlated repositories: Backend, AAPI_code, ASAP.

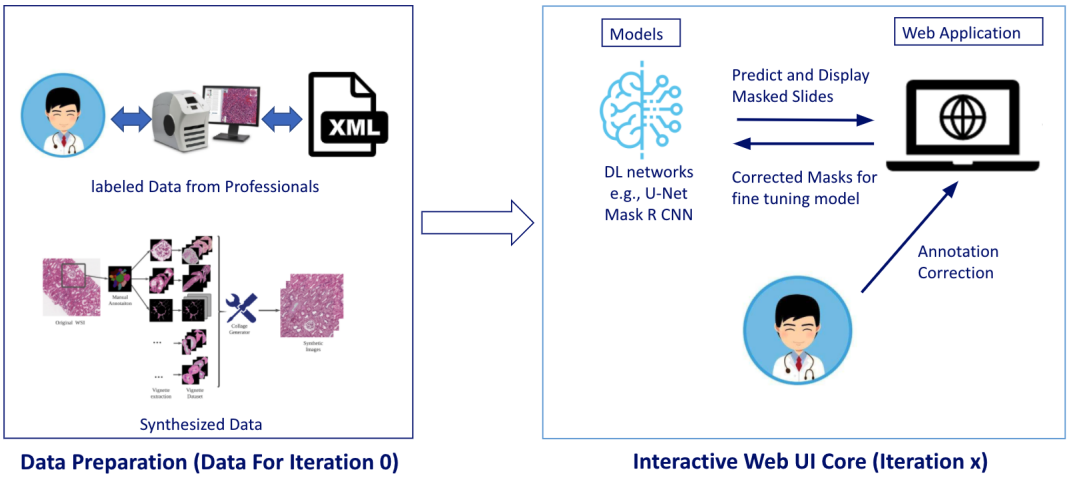
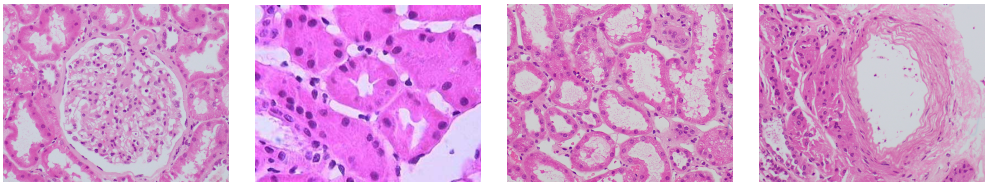


Figure 1: A high level overview of the proposed human-in-the-loop learning pipeline for annotating and analyzing renal pathology images.



(a) Glomeruli (b) Interstitium (c) Tubules (d) Vessels

Figure 2: Histologic structures in kidney biopsy slides.

2 Data

2.1 Datasets

The major pain point we faced was the lack of annotated data, as from practical experiences, correctly labeling a WSI is very time-consuming. Therefore, besides annotated data from our pathologist team, we conducted our experiments using data from the following three sources, which can be further divided into two categories:

- **Target Dataset**: Annotated WSI data from the pathologist team led by Dr. Shana Coley (Department of Pathology and Cell Biology, Columbia University). This dataset shares the same staining protocol, resolution, image acquisition tool with the other slides pending annotations.
- **Auxiliary Datasets**: Datasets created outside the scope of this project, i.e., downloaded or created using slides or images from other open-source projects. It consists of the following two components:
  - **MultiStain**: Adapted from a public available dataset[2] containing kidney pathology images with annotations from Jayapandian et al.[2]. For every structure, to increase the number of training data, we merged annotated ROIs with different staining protocols into one class with the assumption that the morphology and spatial structures will not change much under different staining protocols. Based on this assumption, we will refer to this dataset as **MultiStain** in the rest of this report.

[2] Although the full dataset is still in the process of publishing at the time of writing this report, the released small portion of the whole dataset available on GitHub (`https://github.com/ccipd/DL-kidneyhistologicprimitives`) still provides a valuable starting point.

– **Collage**: Synthesized images using Computer Vision techniques (discussed in detail in Section 2.2): we pasted patches of renal structures (also called "vignettes" in this project) onto one simulated slide with geometric augmentation while mimicking the true spatial distribution of different structures. As this procedure is quite similar to collage making, which is primarily used in art creation, we will refer to this dataset as **Collage** in the following discussions.

Using the analogy of transfer learning, the **auxiliary** datasets serve as the source domain, and the target dataset serves as the target domain. Although the distribution may differ across the source and target domain, there are several common techniques utilized in transfer learning to solve the domain shifting problem, e.g., fine-tuning models pre-trained on the source domain by using data from the target domain.

2.2 Synthesized Image Generation - Collage Generator

2.2.1 Motivation and Overview

Considering the efficiency bottleneck of manual patch annotation, we developed a synthesized image generator that can transform several separately annotated vignettes to one fully annotated ROI.

Due to the need to generate an accurate corresponding mask, we could not apply any NN-based techniques like GAN or VAE to generate an ROI-level image and its corresponding mask. Therefore, we started from a naive approach: we manually extracted the individual vignettes from existing annotation, without keeping any spatial context; then, we randomly generated "collage" with its label mask from these vignettes. The workflow is illustrated as in Figure 3. Our pathologist team has approved the quality of synthetic images, and these synthetic images were applied as a large component of the initial training data.

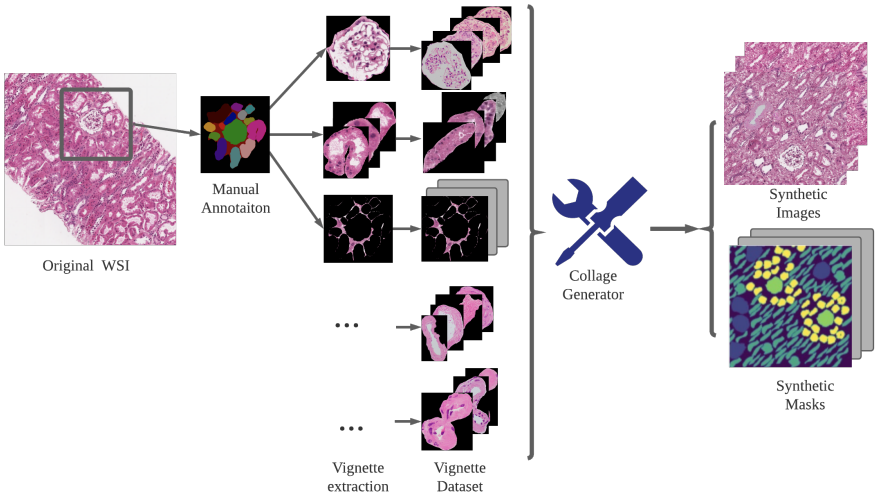


Figure 3: Illustration of the collage generator's working pipeline

This approach shines in several aspects. First, it relieves the burden of annotating the whole slides, which requires special softwares, specific formats of input images, and substantial annotation efforts. Second, all structures appearing on the generated images would be thoroughly annotated with masks. This property is beneficial for training multi-class segmentation engines, e.g., Mask R-CNN, since 1) the model may never get confused on partially annotated data and 2) the model can learn better from the relative positions of different structures. Moreover, it is much easier to get individual renal structures vignettes rather than fully annotated images. With enough vignettes, we can generate an arbitrary number of simulated ROIs for model training. Technically speaking, this approach will inevitably introduce overfitting since, in essence, the segmentation engine is segmenting the same vignette object multiple times. However, as the model

trained from the collage generator is only expected to serve as an initialization of the iterative training, we think it is still a worthwhile approach that maximizes the utilization of the existing data.

2.2.2 Implementation

Figure 4 is an illustration of the collage generator's working mechanism. In short, with the collected vignettes for individual histologic features, we iteratively put them onto a large canvas and then added a synthesized background to the main canvas. More specifically, we generate new collage in the following steps:

- Step 1: Create a canvas larger than the required size. The excess size will be used in step 5.
- Step 2 $\sim$ 4: Add glomeruli with proximal tubules, arteries, arterioles, and distal tubules to the main canvas. Each vignette is geometrically augmented by `Albumentation` [3]. Note that we further separated the "Tubules" class into proximal and distal tubules to distinguish whether the tubules are surrounding glomeruli or not.
- Step 5 $\sim$ 6: Cut the center of the canvas and generate the background to mimic the "part of the structure" that appeared on the edge of actual ROI.
- Step 7 $\sim$ 8: Overlay the canvas with the background and add random Gaussian noise. Then output the collage with a corresponding mask.

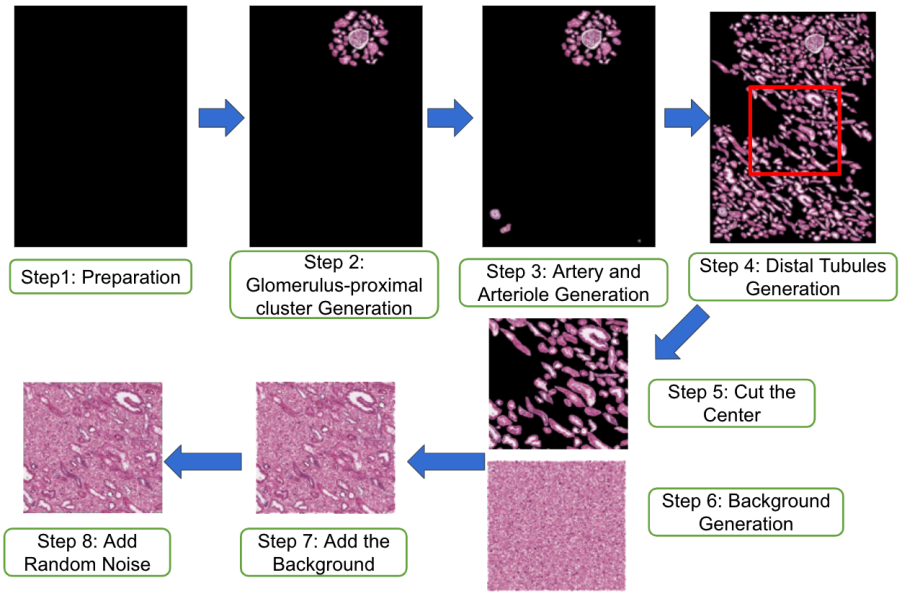


Figure 4: Process of collage generator

Several important features of the collage generator are listed below:

- Following the suggestion from Dr. Perotte and our Pathologist Team, we only kept 40x magnification vignettes in our vignette dataset. As all the vignettes were of the same magnification scale, we used the original size of vignettes instead of the unified size when reading vignettes into the collage generator. This kept the quality of vignette resolution and increased the diversity of vignette sizes on our synthesized images.
- The underlying algorithm was designed to have the vignettes' density controlled by a numerical value called *step_length*. This value is defined as the upper bound of the minimum

[3] `https://github.com/albumentations-team/albumentations`

distance between a vignette and its closet neighbor. In practice, the *step_length* can efficiently control the density of all renal structures presented on an image. The algorithm is illustrated in Algorithm 1.

Algorithm 1: vignette_insertion(vignette, patience, step_length): The insertion operation in the Collage Generator

```
Start with a random position on the canvas;
if Inserting the vignette here causes an overlap then
    for patience times do
        move the insertion point by step_length in both x and y, until we find a position
          that will no longer overlap any inserted vignettes;
    end
    Use this position as the add_point;
else
    for patience times do
        move the insertion point by step_length in both x and y, until we find a position
          that has an overlap with any inserted vignettes;
    end
    Use the last position before overlap as the add_point;
end
Insert Vignette at add_point;
```

- An additional `format_converter` was implemented so that the output can be easily parsed into any standard Common Object in COntext(COCO)[6] or COCO-like format, which is used by most of the mainstream state-of-the-art image segmentation models[4]. It is used in our Mask R-CNN pipeline (which will be discussed in Section 3.2) and can be used with other compatible relevant models in the future.

2.3 Pre-processing

We implemented two parallel preprocessing pipelines for U-Net and Mask R-CNN, respectively, due to the differences in both models' input layer. The Mask R-CNN pipeline only takes in the **Collage** dataset generated from the collage generator discussed above, but it does not require any preprocessing except data augmentation. The U-Net model can take in both **Collage** and **MultiStain** datasets, but it needs a special preprocessing procedure, which is discussed in the following section 2.3.1 and 2.3.2.

2.3.1 Annotation File Parsing

To save the storage space of annotations, most of the WSI analysis platforms like HALO[5]—which is also the tool currently used by our pathologist team—stores annotations using a markup language like XML rather than pixel-level masks. As Figure 5 shows below, every region of interest (ROI) is denoted as an XML element node, containing a series of coordinates marking the exterior edge of annotation polygons. To transform annotations stored as XML to binary or multinomial masks, which are more friendly to CNN architectures, we implemented a small parser to parse XML files. By using `shapely.Polygon` [6] objects, every annotation will be transformed into a polygon. Then by utilizing the `cv2.fillPoly()` function, we obtained the images' mask, where the value for every pixel stands for the corresponding target class we want to classify and segment.

[4] `https://cocodataset.org/#home`
[5] `https://indicalab.com/halo/`
[6] `https://github.com/Toblerity/Shapely`

```
−<Annotation Name="Glomerulus_9" Type="Polygon" PartOfGroup="Glomerulus" Color="#ff0000">
 −<Coordinates>
    <Coordinate Order="0" X="12005.0" Y="53910.0"/>
    <Coordinate Order="1" X="12004.0" Y="53911.0"/>
    <Coordinate Order="2" X="12002.0" Y="53911.0"/>
    <Coordinate Order="3" X="12001.0" Y="53911.0"/>
    <Coordinate Order="4" X="11998.0" Y="53912.0"/>
    <Coordinate Order="5" X="11988.0" Y="53914.0"/>
    <Coordinate Order="6" X="11987.0" Y="53915.0"/>
    <Coordinate Order="7" X="11985.0" Y="53915.0"/>
    <Coordinate Order="8" X="11984.0" Y="53917.0"/>
    <Coordinate Order="9" X="11982.0" Y="53918.0"/>
    <Coordinate Order="10" X="11979.0" Y="53920.0"/>
    <Coordinate Order="11" X="11976.0" Y="53922.0"/>
```

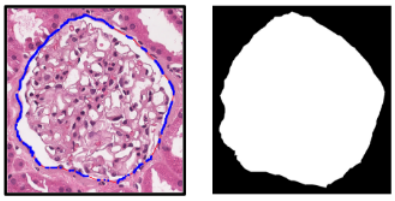



Figure 5: Illustration of annotation XML file, with corresponding image and generated mask.

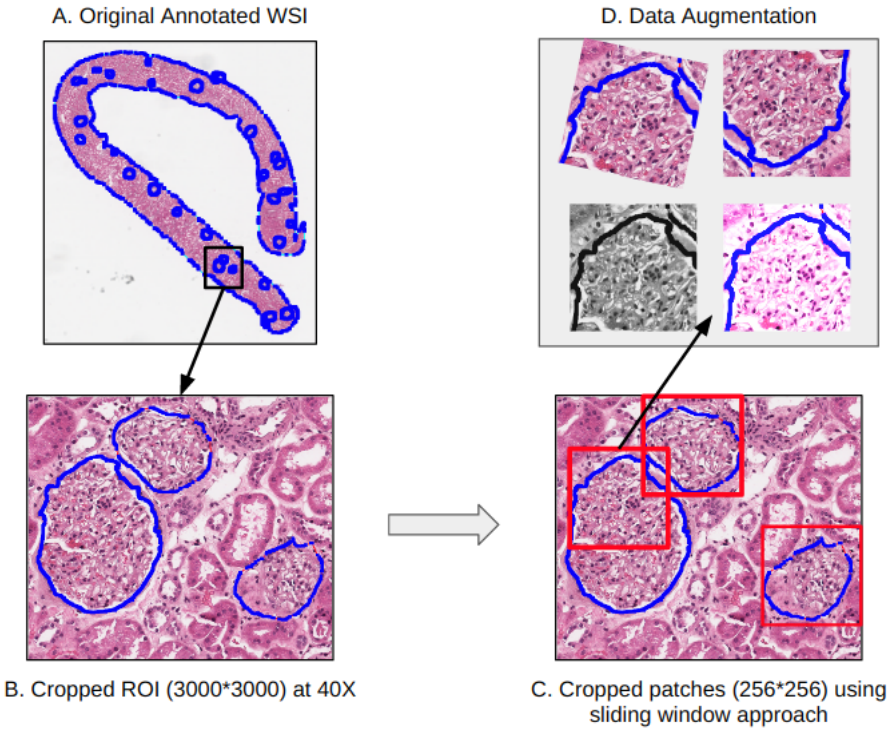


Figure 6: Illustration of data preparation pipeline. Note that current patch size $(256 * 256)$ is suitable for semantic segmentation and may be altered to a larger value for other tasks like panoptic segmentation.

2.3.2 Patches Extraction

As WSI biopsies are always of super-high-resolution($\sim 50000 * 50000$ pixels) and stored in a particular format with multi-resolution layers, it is computationally inefficient and hardware-infeasible to feed the whole slide into a neural network at once. Therefore, a sliding window approach is used to crop small patches $(256 * 256$ pixels) from the WSI and the corresponding mask. These small patches of images and masks are used as inputs and ground truth labels of U-Net. The full pipeline of data preparation is illustrated in Figure 6.

To minimize the number of uninformative patches that only contain backgrounds, we first determined a series of $3000 * 3000$ regions of interest (ROI) covering adjacent annotations on every slide. The center of each ROI was determined in the following way. Hierarchical clustering was first applied to the centroid's coordinates of every annotation polygon to group adjacent annotations. Then we computed the centroid of each cluster and used them as the centers of ROIs. Therefore, every annotated polygon can not only be covered by an ROI but also be placed in the center of

the ROI as often as possible.

Note that for specific structures, some of them may only appear in a small part of the whole ROI due to different sizes and appearance frequency, which will lead to an unbalanced number of positive (containing structures that we are interested in) and negative patches. Therefore, we applied down-sampling techniques on negative patches using the following rule: 1) eliminated patches whose mean value of R channels is greater than 220 (which is an indicator that this patch only contains the blank background of high probability); 2) sampled negative patches by a Bernoulli distribution with $p = 0.6$, so only 60% of them are kept in the final dataset.

Before feeding into neural network models, it is also necessary to apply augmentation techniques on patches to make the model more robust. To maintain the one-to-one correspondence between image and label, identical spatial transformations, e.g., rotation and flipping, must be applied on masks and images simultaneously. In contrast, color augmentations like hue and saturation values are only applied to images to avoid mistakenly manipulating the ground truth labels in the masks.

After going through the pipeline, every input data sample would be formulated into an (image, mask) pair. Both the image and the mask have the size of $256 * 256$ pixels.

3 Methodology

With synthetic data and other published datasets, we selected U-Net[7], which is the state-of-the-art practice in the pathology annotation field, and Mask R-CNN[8], which is the frontier of common object detection, as our underlying DL models. These two pipelines are parallel to each other in the designation.

3.1 Segmentation Models - U-Net

3.1.1 Model Architecture

U-Net[7] model is widely used in the segmentation of kidney histopathology fields. Its encoder-decoder based architecture is illustrated in Figure 7. Following the practice from Jayapandian et al., we applied an additional weight (a constant of 1.2) as the misclassification penalty (which is the cross-entropy loss in our model) for pixels lying on the boundaries of target regions. In this way, the model can better learn the exact contour of target structures.

Although the output channel of U-Net can be arbitrary, which means that it can also handle multi-class segmentation problems in one model, all the previous approach like Jayapandian et al. is still working in a "One v.s. Rest" binary fashion. We followed this practice training a series of models segmenting one particular type of structure at a time. We generated output masks for every patch of the input during prediction, then concatenated them as one full slide.

3.1.2 Output Post-processing

The output of U-Net models would be heatmaps generated from the input series of patches, where for every pixel, the value represented the probability of being predicted as part of one renal component that the model was trained for detecting. In order to transform the heatmaps of patches into one final segmentation result at the ROI level, two additional steps were adapted,

1. **Overlapping Heatmap Aggregation**

   As consecutive patches might overlap each other as the stride size (64) was smaller than the patch size (256), an aggregation function was needed for deciding the prediction values for pixels in the overlapping regions. After experiments, we utilized the `max` function to keep all positive predictions in overlapping regions.

2. **Multi-class Segmentation Aggregation**

   As each U-Net model is currently designed to segment one of the three target structures under a binary classification setting, an ensemble step should be performed to merge predictions for all classes into one multi-class heatmap. Therefore, we applied the "one-vs.-rest" (OvR)

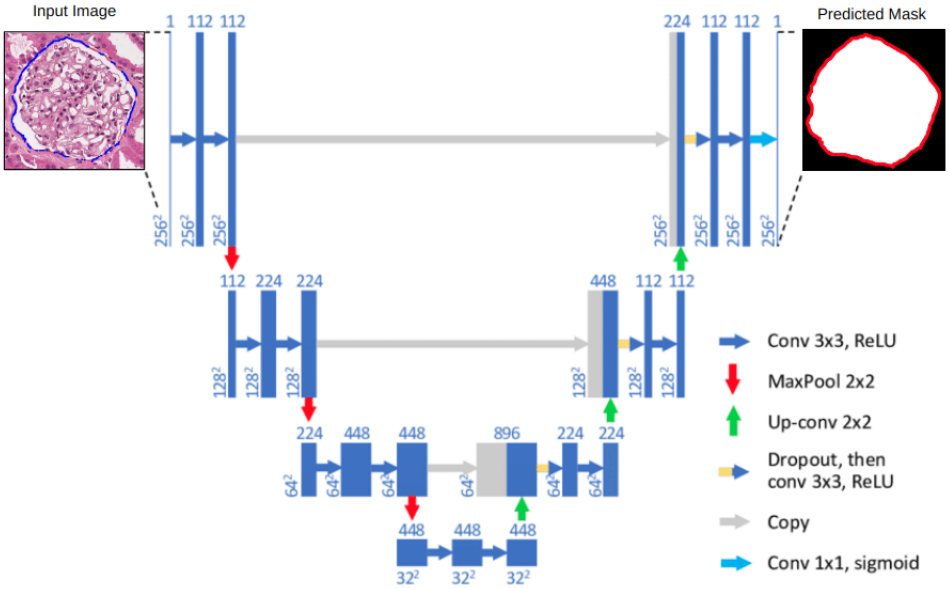


Figure 7: Illustration of U-Net Architecture, modified from the official illustration[7] based on our project-specific hyperparameter setting. Also, a higher weight (1.2) was applied on the boundary of target area in the loss function, indicated as the red contour on the mask.

settings commonly used in multi-class classification problems. In other words, if a pixel was predicted as positive by more than one binary U-Net models, the one with the highest probability would be used for deciding the final class of that pixel.

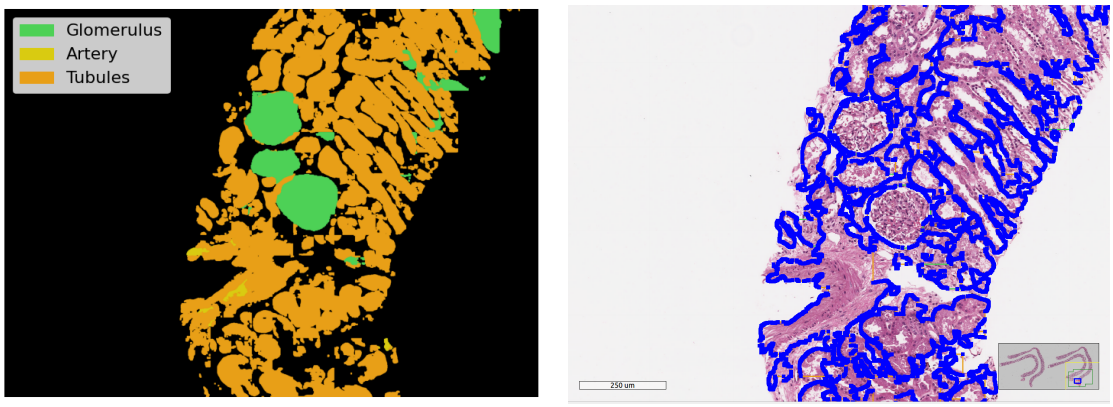
After the unified heatmap aggregates predictions for all structures was generated for one ROI, we used morphological transformations to reduce small noisy regions so that uninformative small proposals could be removed from the final output. Specifically speaking, we utilized the "opening" operation in mathematical morphology to remove small false positive objects from the foreground. Moreover, we first estimated the average size of every target structure by using statistics from the training set. Then, we set 10% of the estimated size as the threshold to filter out contours with too small areas for every structure. Upon conducting these two postprocessing steps, many false-positive predictions can be eliminated to improve the model accuracy further.

After postprocessing, we utilized the `findContours` function from OpenCV to extract contours from heatmaps and encoded them as XML files using ASAP format so that the predictions could be rendered properly as annotations and displayed on ASAP. The predicted mask and annotations are illustrated in Figure 8.

3.2 Segmentation Models - Mask R-CNN

Built on a series of famous successes including ResNet[9], Feature Pyramid Networks(FPN)[10], Faster R-CNN[5], Mask R-CNN[8] is considered as the state-of-the-art approaches in common-object detection area: it replaces the feature map generator of Faster R-CNN with a ResNet-based FPN, and generate 3 different output: a label prediction from softmax, a boundary box regressor from fully connected layer, and a new mask layer from $1 * 1$ kernel convolution. An illustration of this structure is in figure 9.

For the implementation of our Mask R-CNN pipeline, we utilized the official PyTorch implementation by Facebook AI Research, namely Detectron2[11]. Improved from its predecessor implemented in Caffe, Detectron2 claims that it is faster than any other existing PyTorch-based implementations and has full compatibility to any PyTorch extensions. With a well-tested default hyper-parameter configuration, we observed very convincing results (illustrated in figure 10) without any further tuning except the iteration number. Benefited from the ResNet-FPN, Detectron2 can take images of any size as inputs, and the size of each image can even vary inside the training



(a) Generated segmentation mask

(b) Rendered annotations

Figure 8: Demo of ROI prediction results.

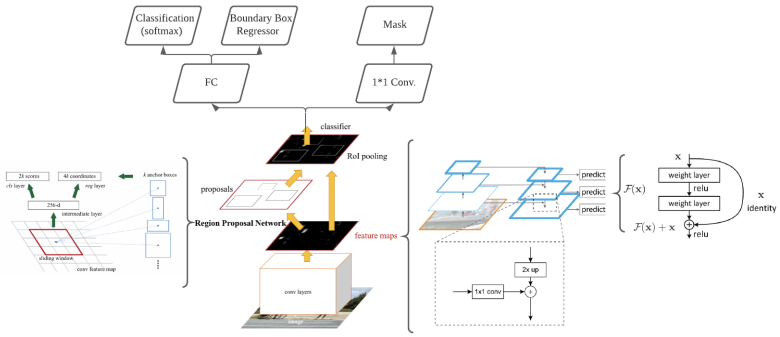


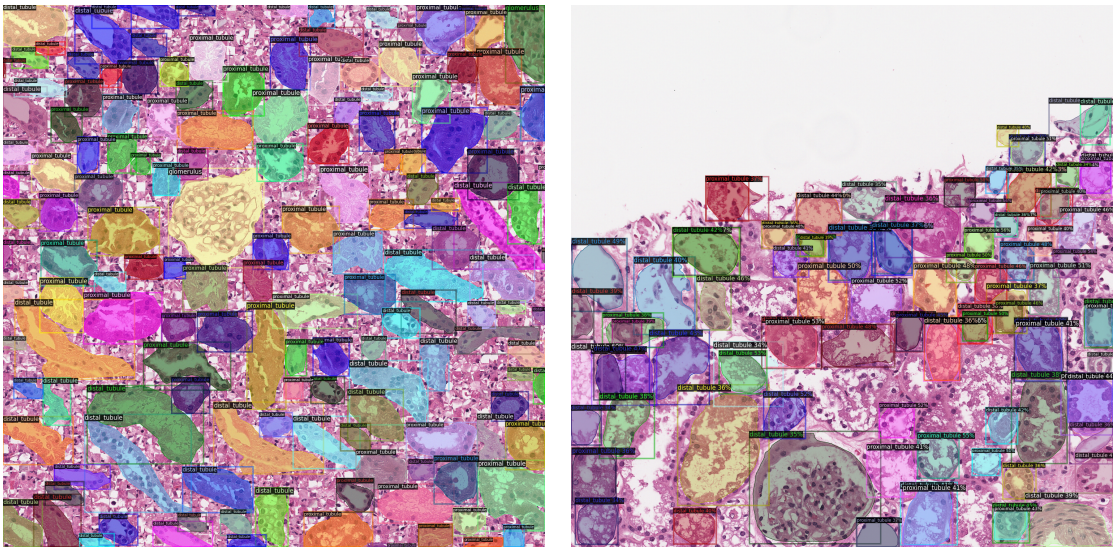
Figure 9: Illustration of Mask R-CNN structure

set or the test set. Therefore, unlike training U-Net models, where we had to first crop every ROI into a series of small patches using a sliding window approach, for training Mask R-CNN models, we directly passed the whole $3000 * 3000$ ROIs into Detectron2. In this way, we kept more context information for the FPN structure and kept more complete renal structures in one input image.

The main challenge of using Detectron2 is that it requires completely-annotated input using the Common Object in COntext (COCO) format[6], which is almost impossible to obtain in the real-world application. However, our collage generator elegantly solved this problem from its mechanism: when we add each vignette into the canvas, we can calculate its contour and append it into the COCO annotation file on-the-fly. We implemented an additional output module to the collage generator to make the training dataset compatible with the input of Detectron2.

The collage generator is helpful to the training of Mask R-CNN from a unique angle, as it can manually define the content inside the training set. The underlying classifier structure inside Mask R-CNN suffers from – even being very fragile in practice to – the imbalanced distribution of target objects. Unfortunately, this is the nature of the renal structure distribution in real kidney cross-section cuts. To combat this issue, when setting the collage generator, we manually defined the fraction of each structure class to upsample objects of the minority classes.

For any possible future extension or replacement of the model, we also implemented a class called `format_converter` to help transfer the input images to standard COCO-Object Detection format.



(a) Training data from collage generator

(b) Prediction output on actual WSI

Figure 10: Visualization of training data and prediction output of Detectron2

4 Experiments & Results

Because of the unique settings of our project discussed in section 1.4, namely the extreme lack of fully annotated data and the special criterion on evaluating our segmentation, our evaluation pipeline is defined in a special way, which is discussed in section 4.1. In order to fairly compare different model architectures, we used a carefully selected metric in 4.2 and provided a complete quantitative evaluation in section 4.3.

4.1 Experiment Settings

The choice of validation and test dataset can be tricky in our task with minimal annotated data.

All the 10 annotated WSIs (the **Target** dataset, provided by our pathologist team) are first split in a 6/2/2 manner for training/validation/testing set, respectively. For the **MultiStain** dataset, ROIs are collected from these WSIs and merged with the auxiliary data to construct the final training and validation dataset. For the **Collage** dataset, vignettes are extracted from these WSIs and fed into the collage generator as materials. In this way, the mixture of the target dataset and the auxiliary dataset can help solve the domain shifting issue and improve transfer learning performance.

Note that the ground truth test dataset **only** comes from the **Target** dataset since it is most similar to the pathologists' perception and shares the same settings as all the other slides utilized in the future iterations of models. By reporting the evaluation metric on the test set, we can know whether the models suffer from the potential domain shifting issue, e.g., differences in resolution (pixel size of the histologic structure), staining protocol (hue of the image), etc.

Meanwhile, from the pathologists' perspective, capturing the correct contour is way more important than classifying it correctly. However, it is generally harder to define a quantitative metric measuring "how close two contours are." Thus, in actual discussions, our pathologist team also used their expertise to evaluate the results qualitatively.

4.2 Evaluation Metrics

In order to have a fair comparison between U-Net and Mask R-CNN models, we decided to compare the performance of semantic segmentation on every ROI. In other words, postprocessing steps mentioned in Section 3.1.2 were applied on U-Net models to aggregate patch level outputs into ROI level outputs. And detected objects (also known as proposals) were grouped under every

semantic class (i.e., every renal structure) for Mask R-CNN models. Then, we constructed a pixel-level confusion matrices for each class, and counted the false positives (FP), false negatives (FN), true positives (TP) and true negatives (TN). The pixel-wise average F_1-score (in some literature it is also known as Dice Coefficient) is finally reported as the primary metric, which is defined as,

$$F_1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall},$$
$$\text{where } Precision = \frac{TP}{TP + FP},$$
$$Recall = \frac{TP}{TP + FN}.$$

Note that, due to the inefficiency of annotating all tubules on one renal biopsy (as they take up almost 80% areas of a biopsy), currently, all available annotations for this class were partially or incompletely annotated in the test set. Therefore, besides the primary metric F_1-score, we also provided two other commonly used metrics, Recall (also known as the positive predictive value, TPR) and Precision (also known as the true positive rate, PPV) for comparison. However, these results cannot be considered as the perfect criterion due to the mislabeling issue.

4.3 Experiment Results

	U-Net @ MultiStain			U-Net @ Collage			Mask R-CNN @ Collage		
Class	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall
Glomeruli	0.8295	0.7378	0.9883	0.7750	0.6770	0.9343	0.5569	0.4167	0.9724
Vessels	0.5757	0.4471	0.8842	0.3842	0.3622	0.4964	0.1760	0.1800	0.1750
Tubules	0.9093	0.9476	0.8745	0.7727	0.9441	0.6541	0.5138	0.9961	0.3511

Table 1: Quantitative evaluation results

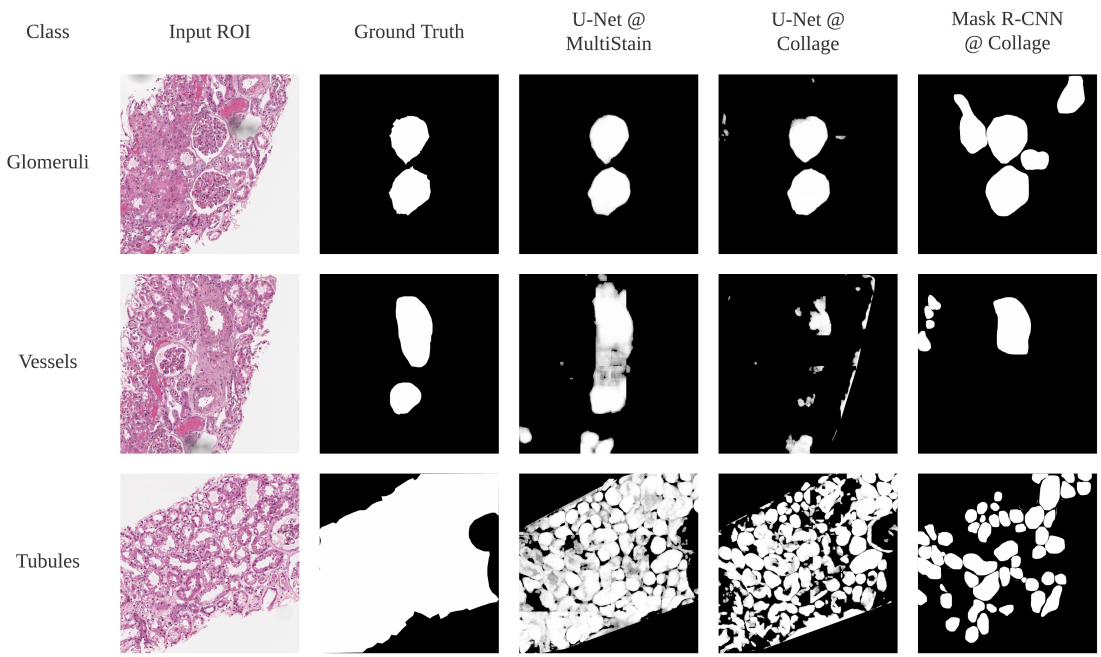
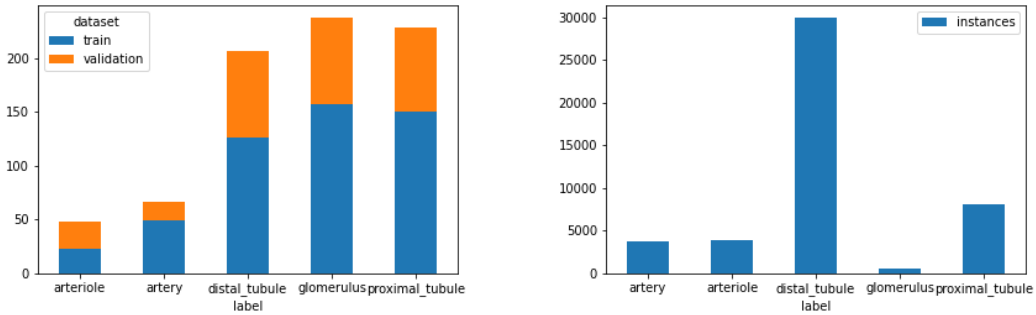


Figure 11: Visualization of segmentation results along with the input image and ground truth

Based on the quantitative evaluations shown in Table 1 and segmentation visualizations displayed in Figure 11, we can arrive at the following observations by comparing the performances of different models in their initialization step:

- **Promising performance can be achieved even with limited real-world data.**
  - For all three target structures, U-Net trained on the **MultiStain** dataset performed the best among all three models, which confirmed the assumption that morphological features can still be learned under different staining protocols.
  - Out of our expectation, even for models trained from purely synthetic **Collage** images, the transfer learning results are still acceptable for glomeruli and tubules. This encourages deploying pre-trained models on synthetic data into our iterative learning pipeline in order to minimize the effort of human annotation.
  - Though under qualitative metrics, Mask R-CNN models are a little inferior to U-Net models. The drop on F_1-score mainly comes from misclassification, e.g., as shown in Figure 11, the Mask R-CNN model mistakenly labeled several tubules as glomeruli. However, on the other hand, for Mask R-CNN models, the output individual target objects are more clearly annotated with sharp contours. According to our pathologist team (see Section 1.4), it is much easier to correct the predicted class label than correcting an inaccurate contour. Therefore, Mask R-CNN models still have their own highlights and potentials.
- **Areas for improvements**
  - Vessels, including arteries and arterioles, are the most difficult structure to be recognized because of their dynamic shape and low appearing frequency.
  - The classifier inside Mask R-CNN is suffering from the imbalanced dataset and lack of data (as shown in Figure 12). This will be solved in the 1st iteration (we are currently at iteration 0, i.e., initializing the loop).



(a) Number of vignettes in dataset

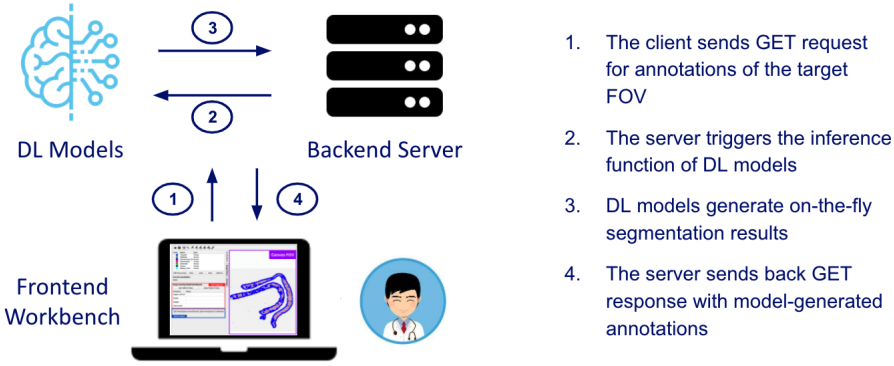
(b) Number of individual instances in training, counted by Detectron2 API

Figure 12: Distribution of available vignettes and instances for different renal structures.

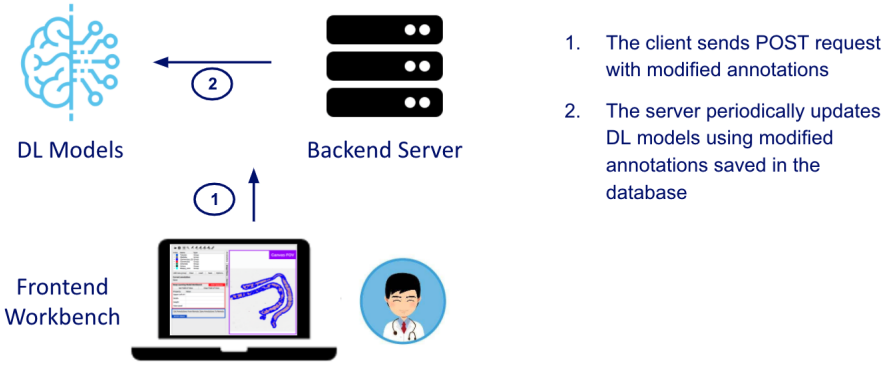
5 Human-in-the-loop Platform Development

We have successfully developed a proof-of-concept version of the human-in-the-loop platform as a core component of the iterative model training schedule. A complete demo of using this platform can be found on YouTube (see `https://youtu.be/arvTPajFP1o`).

The system could be decoupled into a backend server and a frontend client. The backend server has two major components, a DL model (or model groups) backend and a database for saving annotations and slides. The frontend client can render both WSI saved in local file systems and annotations received from the backend server. Moreover, the frontend client comes with an annotation toolkit, allowing pathologists to create and modify annotations interactively. The corrections provided by end-users will then be posted to the backend database and used for training the next generation of DL segmentation models. The major two workflows are shown in Figure 13.



(a) Retrieve model predictions



(b) Save user modification

Figure 13: Web application workflows

5.1 Back-end Development

5.1.1 Local File System and Database Design

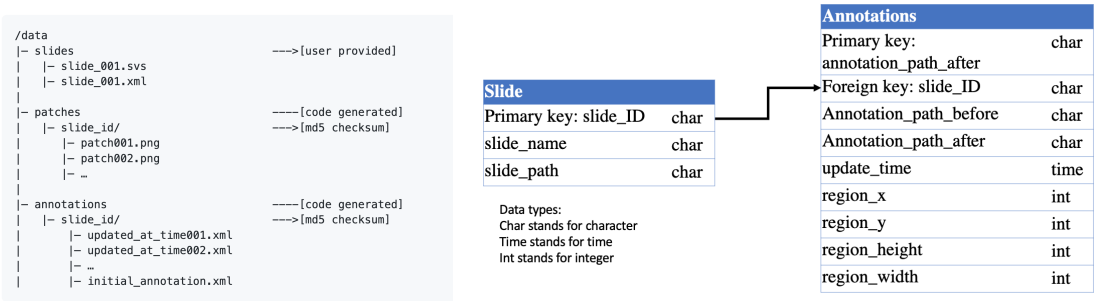
Considering that the slides (WSIs) are usually quite large ($\sim$ 300MB), the slides are saved in the local file system instead of the database and are avoided to be sent over network requests. The client-side and the server-side must have identical copies of the same slide, and a unique identifier (MD5 checksum) is used between the server and the client to enforce the constraint.

In the backend's file system, a constraint for the directory structure is set as shown in Figure 14(a). All slides and the corresponding annotations are saved and loaded in this structure in the file system. Besides, to perform on-the-fly prediction on any arbitrary region, small patches ($256 * 256$ pixels) cropped from slides are also cached in the file system. Therefore, only the region's meta-information (e.g., bounding box size, coordinates) is needed in future network requests rather than the actual image. These patches will then be used to reconstruct the intended regions upon the client's request.

Meanwhile, a database is designed to save the meta-data of slides and records and track the updated history of annotation. As shown in Figure 14(b), two tables are created to store the slides and annotation records.

The first table "Slide" includes the name, the path, and a unique id of every slide available before starting the server. As mentioned earlier, every slide's unique id is the WSI checksum generated using the MD5 hashing algorithm to prevent the inconsistency of slide names between the client and the server.

The second table "Annotation" stores information about annotations. The "slide_ID" column is the foreign key referring to the "Slide" table's primary key. The "Annotation_path_after" column points to one annotation file's path, and it is used as the primary key for the annotation table. The "update_time" column saves the timestamp when this annotation record is inserted



(a) Server side file system structure

(b) Database schema

Figure 14: Local file system and database design

into the database. The other columns are optional. It indicates that this annotation has been updated based on an older version if all columns are provided. For example, if an update is made on a specific region, that region's information will be saved as a tuple of (region_x, region_y, region_height, region_width) in the database. The "Annotation_path_before" column serves as a back pointer to back-trace the annotations that have been updated to generate this new one. With these columns, the users are provided with the capability to rollback to certain previous annotations whenever they are unsatisfied with the updated version.

5.1.2 API Endpoint Design

As the back-end server mainly support 2 functionalities, 1) retrieving annotations for slides or regions and 2) updating annotations from received data, two corresponding API endpoints are designed as follows:

```
GET/POST http://127.0.0.1:5000/annotation/<slide_id>
```

with four optional query parameters supported:

```
<API_Endpoint>?x=<x>&y=<y>&width=<width>&height=<height>
```

The "GET" request is used to retrieve annotations. If all four query parameters are provided, the server will then call the pre-trained DL model's inference function to provide on-the-fly segmentation for the region specified by the query parameters. Otherwise, pre-computed full slide annotations will be retrieved from the database using the slide_id provided in the request. The response body will be the content of XML-encoded annotation files, either freshly generated by the model or cached in the local file system.

The "POST" request is used to update existing annotations according to newly received annotations sent within the request's data body. If all four query parameters are provided, the update will only be applied within the specified region. In other words, all annotated polygons lying in that region from the latest (but also outdated) version will be replaced by those in the same region from the newly received annotations. If incomplete query parameters are provided, the updates will be applied to the whole slide instead of a specific region. After updating, the new annotations will be saved as an XML file in the local file system, and a new record describing this update will be inserted in the "Annotations" table of the database.

5.1.3 Periodic Model Updates

Besides the functionalities of CRUD in the database for the annotation management and calling DL models, the backend server is also responsible for updating DL models using newly corrected annotations, illustrated as step 2 in Figure 13(b). By using `apscheduler`[7], the updating task would be scheduled in a separate background process and triggered every 20 minutes. In this

[7] https://github.com/agronholm/apscheduler

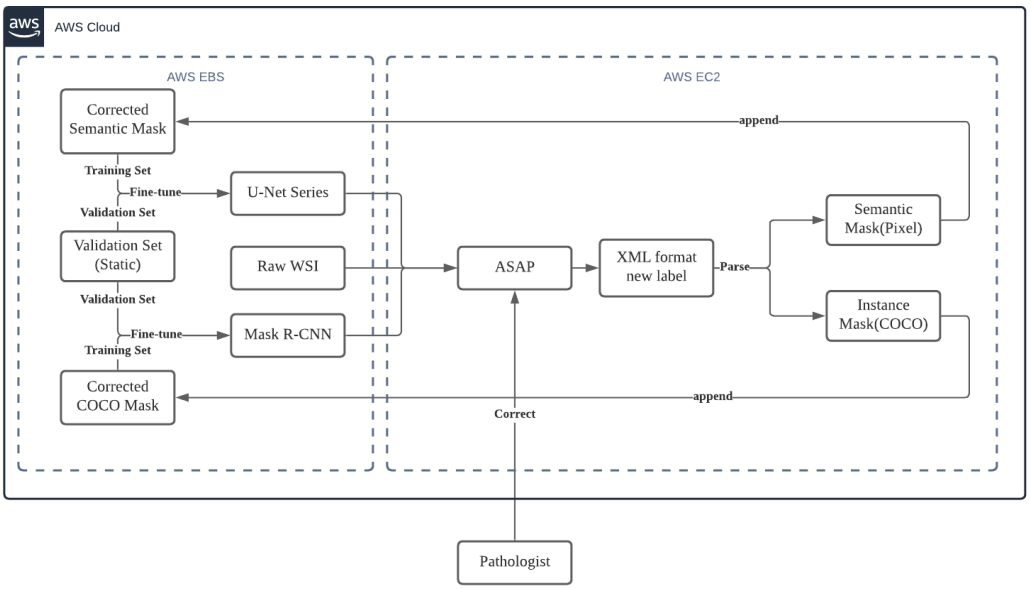


Figure 15: Periodic model updating pipeline. Note that the Mask R-CNN branch is still under construction.

way, the updating process will not affect the communication between the server and the client, and corrected annotations will be utilized iteratively. According to the experiment results shown in Section 4.3, U-Net models trained on the MultiStain dataset perform the best, and they are selected as the base models for future iterations.

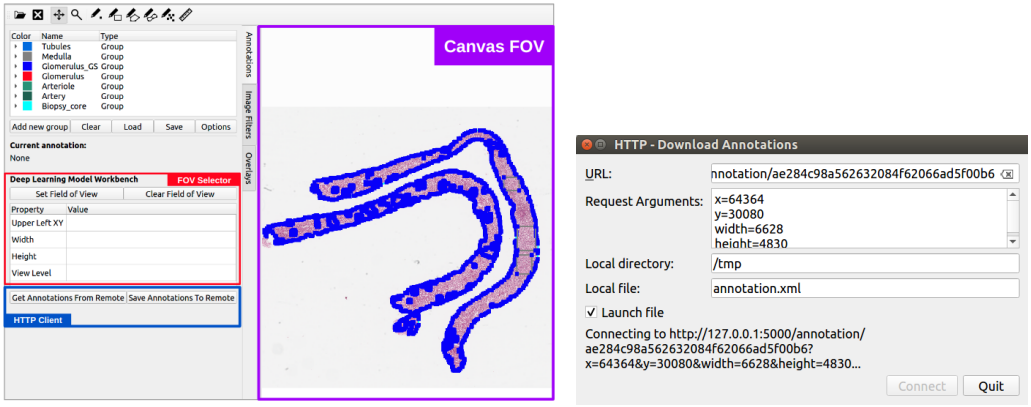
By balancing the trade-off between implementation difficulty and theoretical correctness, the current model update pipeline is implemented, as shown in Figure 15. Three aspects are specially considered when designing the updating strategy,

- **Choice of Validation Set**: in order to have a fair and consistent comparison between the updated models and the base models, the validation set, which was constructed on the **MultiStain** dataset and used for tuning the U-Net models, is also adopted here for model selection. In other words, the validation set is static and will not change across different iterations of models.
- **Constructing New Training Sets**: as timestamps are collected when corrected annotations are received from the frontend client, the models can utilize the latest version of annotations for every slide. As the models will be updated solely based on user-corrected annotations, we do not apply any weights or filtering to slides with corrected annotations so that all of them can be fully leveraged. Same preprocessing steps showed in Section 2.3.1 and Section 2.3.2 are applied to the new training set before feeding them into the DL models. Every new model will be trained on the new training set for 5 epochs before evaluation.
- **Model Selection Criterion**: the F_1 score on the validation set is still used as the evaluation metric, and the performance of the latest model will be used as the benchmark: If the newly updated model can achieve at least 95% of the benchmark performance measured by F_1 score, it will replace the old ones and be used for future iterations. In this way, we can prevent the models from overfitting the new training set, which can be relatively small, especially in the first few iterations.

5.2 Front-end Development

Although hosted together with the backend server on a remote machine, the frontend client can be accessed via X11 forwarding. This design will provide a security guarantee over SSH, and a system admin can control user privileges on the OS-level.

An HTTP client interface was added onto the existing ASAP[8] platform to receive and update annotations from the backend Flask server. An overview of the updated version of GUI is shown in Figure 16. These newly added functionalities can be divided into two parts, 1) a Field-Of-View (FOV) selector and 2) an HTTP client.



(a) Overview of updated ASAP GUI

(b) HTTP client UI

Figure 16: Demo of graphical user interface

For the first part, the selector will be updated using the current canvas parameters at the right-hand side after the "Set Field of View" button is clicked. All parameters, like the width, height, and the upper-left coordinates of the current region, will be saved in the table on the left-hand side, which will be used later as query parameters when sending requests to the backend server. As conducting segmentations on the entire WSI is usually time-consuming (which may take around several minutes for each structure in current settings), and it is hard for users to check and modify all predicted annotations on the slides all at once, the FOV selector helps users focus on a specific region and modify the model-generated annotations of the current slide in an incremental and computationally efficient way.

For the second part, the HTTP client can construct both the GET and POST requests sent to the backend server. As mentioned above, the cached parameters saved in the table will be used as query parameters when sending requests to the server. For the GET request, if those parameters are not set, in other words, the table is cleared by the user, the intended functionality is to fetch a pre-computed annotation file for the entire slide. Otherwise, the latest models will generate annotations on-the-fly for the region specified by the parameters saved in the table. The data received from the backend will be ByteArray containing XML-encoded annotations for the whole slide or the focused region. The client will then save the received data in a temporary file and call ASAP's internal methods to reload annotations from that file. For the POST request, the only difference is that the client is now trying to push something new to the backend database. The parameters saved in the table will then be used in the backend server to merge the previous and updated version of annotations. By first saving the currently rendered annotations as a temporary file in the local file system, the client will then send the file's content through a POST request to trigger the "merge annotation" functionality in the backend.

The original version of ASAP provides interactive annotation toolkits for operations like locating and modifying annotation contours, adding or deleting an entire annotation or a group of annotations, etc. As our modification onto ASAP can be viewed as adding an extension for constructing and sending HTTP requests, all these previous functionalities can still be used smoothly.

[8] `https://github.com/computationalpathologygroup/ASAP`

6 Discussion

6.1 Conclusion

In summary, we developed an interactive human-in-the-loop platform to help pathologists relieve the burden of manual annotations. As the platform requires a minimal amount of labeled data to achieve acceptable model performance, we believe this tool can improve clinicians' efficiency in the long run. Our major contributions are summarized as follows:

1. Proposed an innovative algorithm to generate synthesized pathology images and the corresponding annotation masks;
2. Proposed two parallel pipelines for two different genres of image segmentation models and achieved promising performances on the task of semantic segmentation by using only the synthetic data;
3. Proposed a human-in-the-loop working pipeline for further fine-tuning of the pre-trained models mentioned above.
4. Designed and implemented a user-friendly full-stack web app to support the human-in-the-loop pipeline, and deployed the whole system on AWS.

6.2 Future Work

So far, promising first-stage results have already been achieved on both the DL model training and the human-in-the-loop platform construction. Some thoughts for the future work to refine the existing models are shared below.

6.2.1 Mask R-CNN

The iterative pipeline needs to generate enough real-world annotations for Mask R-CNN to address the imbalanced dataset issue in the Detectron2 experiment. It is also promising to try calibrating the model to find a proper threshold. We believe that Detectron2, the state-of-the-art segmentation engine, has a high potential in the later-stage performance. Moreover, the Mask R-CNN model needs to be integrated into the human-in-the-loop platform along with the existing integrated U-Net models.

6.2.2 Better UX

To improve user experiences, a logging system is needed for checking the status of periodic model updates. Critical information, like model improvements and training duration, can also be sent to the user's reference front-end. Moreover, as latency is the main issue to use X11 forwarding, it is better to rewrite the front-end client as a cross-platform Qt App. For deployment, other AWS resources like AWS CloudWatch(Logging recording), DynamoDB(Database), Cognito(login system), and even Cloudformation(DevOps) may be helpful for scaling up this platform.

6.2.3 Iterative Pipeline

As discussed in section 6.2.1, Mask R-CNN needs a relatively more extensive training set. In our expectation, the U-Net model will be used in the early stages to help pathologists label enough slides, and Mask R-CNN will be fine-tuned in the later iterations. This iteration would also require precious feedback from our Pathologist Team later.

6.3 Ethical Considerations

While doing the project, we always paid attention to data ethics. The human-in-the-loop training and annotation platform was designed to save all biopsy slides in the server-side local file system and prevent sharing slides over the open network. Furthermore, OS-level user privileges will be granted by the system admin. In other words, only users with SSH credentials can connect to

the remote server, and the admin controls permissions to the file system. Some users would only modify or download annotations and would have no access to original slides. We believe saving files remotely and designing various access levels can protect data privacy and prevent a data breach.

7 Contribution

- Chao Huang: Major contributor for the U-Net training pipeline; Major Contributor for the postprocessing, annotation generation and online model updating pipeline[9]; Major contributor for developing the HTTP client extension and modifying GUI of ASAP[10]; Helped design and implement the backend server[11].
- Yihao Li: Team Leader. Major Designer of the main pipeline and all sub-pipelines except the U-Net one. Major contributor to the design and implementation of `collage_generator` and **Collage** dataset. Major Contributor to `format_converter` and Detectron2 pipeline's implementation[9].
- Yufeng Ma: Developed the backend server of the human-in-the-loop platform[11], including designing the local file system, database and endpoint APIs; Code documentation including collage generator.
- Shuo Yang: In charge of data management, cooperated with our pathologist team to prepare annotations, collected original vignettes for collage generator, and managed the collage dataset.
- Xiaoyun Zhu: The main contributor to the searching and reading of existing literature and researches. Including the converter among various COCO formats and the input and output format for Detectron2 and related tutorials.
- Also, we appreciate the work of Dr. Perotte and our pathologist team led by Dr. Coley.

[9]Sub-repository of DL model pipelines and the collage generator:`https://github.com/perottelab/AAPI_code`
[10]Sub-repository of front-end client: `https://github.com/perottelab/ASAP`
[11]Sub-repository of back-end server: `https://github.com/perottelab/Backend`

References

[1] N. Altini, G. D. Cascarano, A. Brunetti, F. Marino, M. T. Rocchetti, S. Matino, U. Venere, M. Rossini, F. Pesce, L. Gesualdo *et al.*, "Semantic segmentation framework for glomeruli detection and classification in kidney histological sections," *Electronics*, vol. 9, no. 3, p. 503, 2020.

[2] C. P. Jayapandian, Y. Chen, A. R. Janowczyk, M. B. Palmer, C. A. Cassol, M. Sekulic, J. B. Hodgin, J. Zee, S. M. Hewitt, J. O'Toole *et al.*, "Development and evaluation of deep learning-based segmentation of histologic structures in the kidney cortex with multiple histologic stains." *Kidney International*, 2020.

[3] M. Hermsen, T. de Bel, M. Den Boer, E. J. Steenbergen, J. Kers, S. Florquin, J. J. Roelofs, M. D. Stegall, M. P. Alexander, B. H. Smith *et al.*, "Deep learning–based histopathologic assessment of kidney tissue," *Journal of the American Society of Nephrology*, vol. 30, no. 10, pp. 1968–1979, 2019.

[4] N. Bouteldja, B. M. Klinkhammer, R. D. Bülow, P. Droste, S. W. Otten, S. F. von Stillfried, J. Moellmann, S. M. Sheehan, R. Korstanje, S. Menzel *et al.*, "Deep learning–based segmentation and quantification in experimental kidney histopathology," *Journal of the American Society of Nephrology*, 2020.

[5] Y. Kawazoe, K. Shimamoto, R. Yamaguchi, Y. Shintani-Domoto, H. Uozaki, M. Fukayama, and K. Ohe, "Faster r-cnn-based glomerular detection in multistained human whole slide images," *Journal of Imaging*, vol. 4, no. 7, p. 91, 2018.

[6] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[7] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[8] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[10] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2117–2125.

[11] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, "Detectron2," https://github.com/facebookresearch/detectron2, 2019.