Spring 2024

Pathology Slide Segmentation

Mohamed Mohamed
University of Texas at Arlington

Pathology Slide Segmentation

By

Mohamed Mohamed

Masters Thesis for the University of Texas At Arlington

May 2024

Supervising Committee:

Christopher McMurrough

Chris Conly

Shawn Gieser

Acknowledgements

I would like to acknowledge Dr. Luber and his lab for help with this thesis.

Contents

Abstract

The goal of this project was to create an image segmentation model that would extract an image of H&E pathology slides from real life scenarios. This was a part of a project that required the extraction of an image, querying of the image, and displaying the results. The baselines of the model were to be efficient, run on mobile devices, and be able to work with cameras of varying resolutions.

Introduction

Medical imagery is an established, age-old method for analyzing the human body. Many have been established well before current times and continue to improve with use of newer tools coming out today. One of these techniques to prepare tissue samples is H&E staining.

Like most things in the medical field, it has been rigorously tested to ensure its efficiency and reliability. Staining involves dyeing a part of human tissue to view under a microscope. It was first introduced in the 1800s and has been a cornerstone in the field of histopathology since its inception.

The way these medical images are interacted with is the similar to as it was were from back in the day. It involves a doctor sitting down and manually reviewing slides, basing their decisions off experience from seeing slides. But, there are new improvements being made today involving computer technologies. Implementations of image segmentation have been seen on digital pathology slides to diagnose, or label, images of pathology slides. There are similar techniques being introduced in other adjacent fields, such as image segmentation in CT scans to isolate structures, or in ultrasound imaging for detecting inconsistencies in the body. Applying these newer techniques is a fast growing field for all things related to imaging.

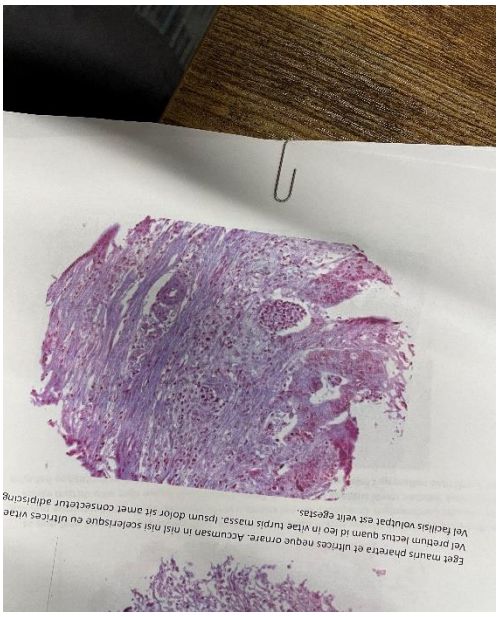
The goal of this project was to design and implement a model to assist in extracting medical imagery for analysis. The model would give physicians an efficient tool to quickly extract images of pathology slides from real life scenarios to improve intractability through the diagnosis processes.

Inspiration

This project was inspired by a doctor's request for a pathology slide search implementation. The objective was to be able to quickly search given pathology slides and display the search results in an interactable manner.

The pathology slide image segmentation model retrieves a picture of the slide and sends the extracted image to a multimodal search engine. The extracted image would be embedded into a latent space and searched across other pathology slides. Finally, the result would be sent over to a computer for imaging.

This paper will focus on the initial stage: The development of the image segmentation model. The baseline for the model was to extract the pathology slides from a camera. A simple example would be a printout picture of a slide lying on a table.



Example taken from data set

The overall goal is to create a seamless interaction for on-the-field physicians to easily search given pathology slides in various real-world settings. Physicians often spend significant time in front of pathology slides when attempting to diagnose patients. Given a consistent and seamless approach to searching and displaying pathology slides, it would aid in comparing patient data with stored references to give a diagnostic in a timely fashion.

Background

H&E Pathology slides:

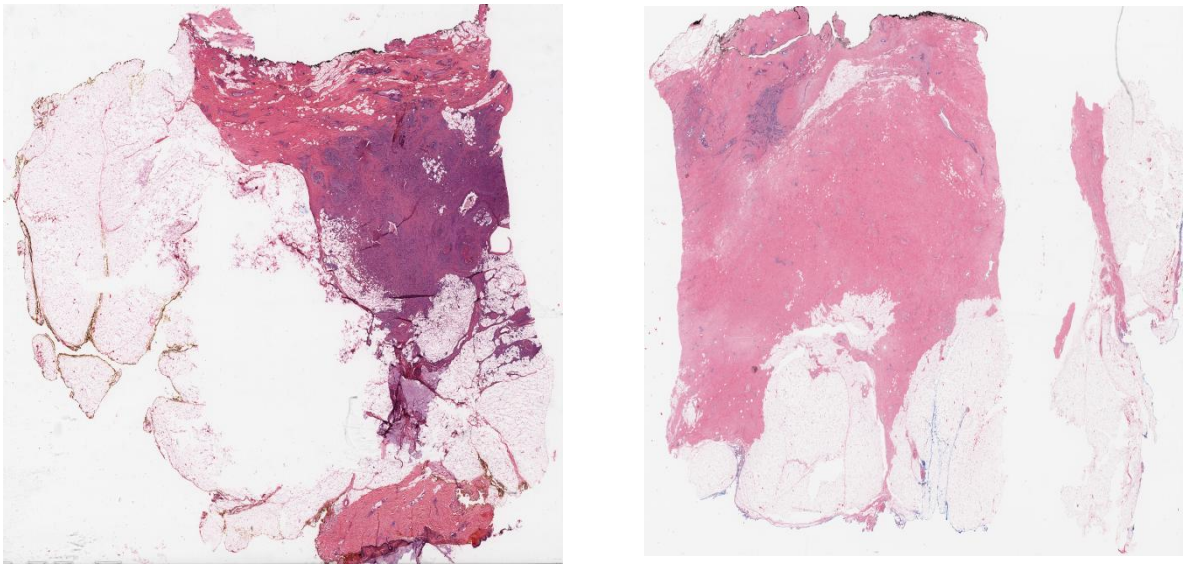
As a medical procedure, doctors can order a patient to undergo a biopsy, where a cut of human tissue is extracted and preserved. This tissue is sent to be examined by a specialist, namely a pathologist. To view these thinly cut tissues, it's not enough to simply place it under a microscope. To mark notable sections of the tissue for pathologists, who are specialists that analyze pathology slides, the tissue undergoes processing before review. These processes vary, but for this specific paper the target is hematoxylin-and-eosin-stained slides, which are also known as H&E slides.

H&E slides are the most common and general kind of pathology slides. They are a baseline for pathology and are able to consistently provide info on the morphology of tissue. The process of creating an H&E slide involves dyeing the tissue with Hematoxylin and Eosin stains to bring color in the morphology of the slide. Simply put, Hematoxylin will dye the cell nuclei blue, and the Eosin will dye the cytoplasm, extracellular matrix, and other structures various shades of pink.

The results of H&E staining produce pathology slides that are dyed and captured in very high resolution. It's common that a single image to be multiple

gigabytes large, resulting in an image with stains that allow pathologists to analyze the morphology of the tissue at a microscopic level.

The results of H&E slides are foundational to the field of pathology, as pathologists study these kinds of slides to identify subtle changes and patterns across tissues. The results of these slides are directly used in diagnosing patients with various medical conditions. The target audience of this project would be physicians who are frequently referencing these pathology slides and would need to reference different slides during diagnosis for results.



Examples of H&E Pathology slide from the Dryad dataset

Image Segmentation:

To enable computers to perceive the real world, computers have had sensors attached to them and are then given the ability to analyze surroundings. Cameras have historically been used as a consistent sensor to stream video input data to computers. This field study, aptly named computer vision (or shorthanded to CV) focuses on creating systems to derive information from visual information. The subsection of this field implemented in this paper is image segmentation.

Image segmentation put plainly is as the name implies, a way to segment an image. While for humans it's natural to perceive objects within a field of view as individual, for machines it's a much harder task to differentiate objects within a view as individual. The goal of image segmentation is to correctly separate groups of pixels within an image, partitioning them into discrete objects.

The goal of this project was to partition an H&E pathology slides from its surroundings. To achieve this, an image segmentation model was created with the focus of segmenting specifically the pathology slide and grouping anything not related as junk data.

The model used in this project was not created from scratch, and a backbone was implemented to accelerate the training process. A backbone is a pretrained foundational architecture that contains pretrained weights and balances. The backbone only serves as a general-purpose structure for models and must be built upon for application in more specific use cases.

The chosen backbone model was ResNet-50, which is a popular Convolution Neural Network architecture. The model choice was decided using the 2022 paper “Backbones-Review: Feature Extraction Networks for Deep Learning and Deep Reinforcement Learning Approaches” by Omar Elharroussa, Younes Akbaria , Noor Almaadeeda and Somaya Al-Maadeeda. They tested a multitude of model backbones and across the board ResNet-50 scored consistently well.

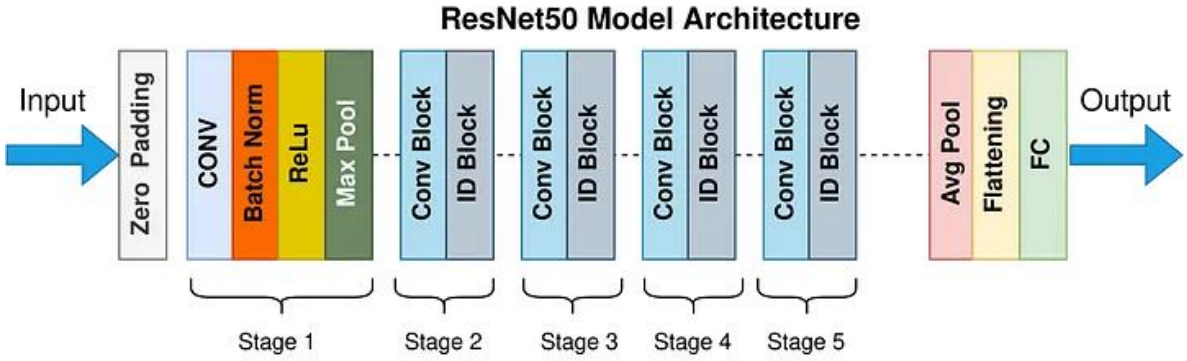
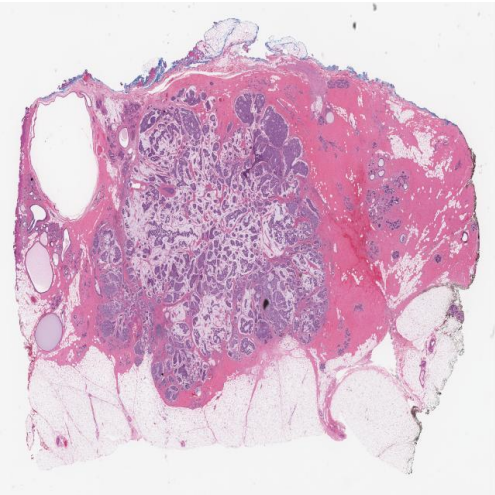
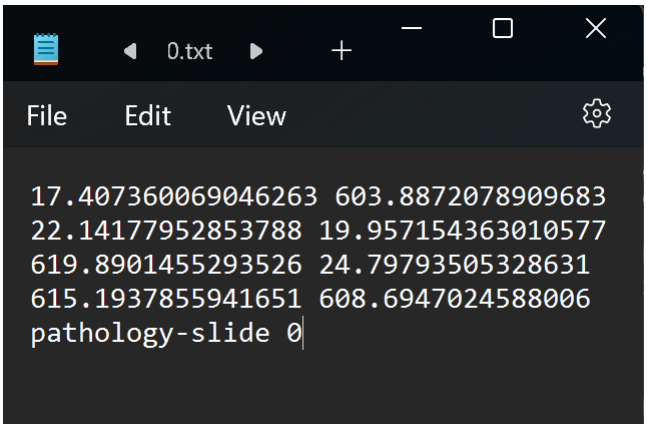


Image of ResNet 50 Model Architecture

For training the image segmentation model, image masks were implemented. An image mask is an overlayed image with labeled sections, marking where an object is. In this case, the image masks were not full sized images but rather pixel value points of where the object, in this case pathology slide, was located inside of the image.

Image and corresponding mask representation





Core ML

For application on mobile devices, there are two target platforms: Apple and Android mobile phones. In todays age it is realistic to say that most working adults, notable here physicians, will have a cellular device. This gives a consistent baseline to work towards for the real-world application, which is to be able to run on modern smartphones.

For usage on Apple smartphones, and all Apple platforms, PyTorch models need to be converted into Core ML models. Core ML is Apples proprietary machine learning framework which is used for efficiently running models on Apple devices. Its meant to serve as a standard for their devices, and supports model formats from popular model libraries such as PyTorch, TensorFlow, etc.

In this project, the model is converted to an Core ML model for usage on Apple devices. This process includes tracing the model and applying it in a Swift framework to run the image segmentation.

Tracing a model, in the context of Core ML, is a 1-to-1 copying of a non-Core ML model into a Core ML model. The output Core ML model is able to be ran on any updated Apple device, meaning support for notably Apple smartphones here. The process is inconsistent in its completeness, and some PyTorch layers are not supported. An example would be PyTorch’s argmax function being supported by Core ML tracing but its argmin function being unsupported.

Related Works

As mentioned in the introduction, there are a lot of related works growing in the field of medical imaging. First and foremost, the most related working is image segmentation of pathology slides. This form of segmentation is applied directly to WSI (whole slide imaging) slides. The example being mentioned in this paper will be from the research article “A generalized deep learning framework for whole-slide image segmentation and analysis” by Mahendra Khened, Avinash Kori, Haran Rajkumar, Ganapathy Krishnamurthi & Balaji Srinivasan.

To being their training process, an approximate mask is created for the tissue sections of the WSI images. The goal of this is to remove unnecessary, non-tissue area from the image before being fed to the model. By removing the background, it allows for higher accuracy results with lower computational costs. Training on just the tissue sample area will avoid confusing the model with unrelated white space. The computational costs are lowered alongside this since there is physical less area to input. This is significant because the WSI images are massive, and the white space surrounding tissue can count up to millions of pixels.

These WSI images are large in size, with the paper mentioning that their typical WSI image is 80,000 x 60,000 pixels. This is the nature of most WSI images, as the resolution needs to be microscopic for physicians’ use. The full resolution allows them to deeply understand the nuclei and cell structure of each patient’s tissue sample.

To deal with these massive images, the researchers take the approach of patching the WSI images. The patches area is decided using a lower resolution image, based on the area of the image masks, and then patched from the full resolution image. Combining the masking with the patching allows for the patches to extract only the most relevant areas of the tissue, and completely bypassing any whitespace. This approach yields to consistent and high information patches. The patches are split into cancerous and non-cancerous patches depending on whether there is any cancerous pixels within a patch. The threshold for being labeled cancerous is a single cancerous pixel.

Preprocessing the patches with image augmentation is also introduced. The images are applied with augmentations such as Gaussian Blur and flips along an axis. Color augmentation is also utilized, with example applications being changes to brightness, contrast, and hue.

Tumorous regions were found to be a very small section of section of each WSI image. This is because WSI images are massive as explained before, a single label cannot properly classify it when being used for training data. A human tissue sample can is labeled as cancerous if any amount of cancer is found, even if sections of the tissue sample are completely cancer free. This approach is fine in the real world since that implication is understood by physicians, but corrections need to be made when using the images as training data, especially so when data is being patched from slides into cancerous and non-cancerous patches. The approach taken to avoid class imbalance of cancerous and non-cancerous data

was to use a hybrid loss function. The hybrid loss function utilizes cross-entropy loss and “a loss function based on the Dice overlap coefficient.” Loss is a way to represent the amount of error, and using functions to minimize loss leads to accurate models. The first loss used here is cross-entropy, which is a logarithmic based loss function. The formula for cross-entropy is:

$$H(P,G) = -\sum p(x) \log(g(x))$$

where P is the target probability distribution of x, G is the models outputted distribution of x, and x is the object. In this case, x would be the tissue patch extracted from a WSI image. Cross-entropy is a well-established loss function for classification tasks, and is a good pick here since the model is attempting to classify sections of WSI images as cancerous and non-cancerous The other mentioned loss function utilizes the “Sørensen–Dice coefficient”. The formula for it is:

$$2 * |A \cap B| / (|A| + |B|)$$

where A and B are two sets. A ∩ B, or A union B, is a representation of the overlap between the two sets. The |A ∩ B| is the count of elements that overlap between A and B. |A| is the count of elements inside set A, and |B| is similarly the count of elements inside set B. |A| + |B| is the combined count of elements inside set A and B. |A ∩ B| is multiplied by two because each overlap is counted once, but the Sørensen–Dice coefficient is finding the overlap percentage between the combined sets. By multiplying |A ∩ B| by two, it properly represents the percentage of overlap between two sets instead of one. The exact cross-entropy equation used in this

situation is:

$$CL = \sum_{i}^{N} (g_i log(p_i) + (1 - g_i) log(1 - p_i))$$

And the dice loss equation used is:

$$DL = 1 - \frac{2\sum_{i}^{N} p_i g_i}{\sum_{i}^{N} p_i^2 + \sum_{i}^{N} g_i^2}$$

Finally, the combined equation for both losses is:

$$Loss = \alpha * CL + \beta * DL_{BG} + \gamma * DL_{FG}$$

where CL is cross-entropy loss and DL is dice loss. DL_{FG} represents the dice loss of the foreground pixels that relate to tumorous pixels and DL_{BG} is of the background pixels that relate to non-tumorous pixels. The result is a formula capable of capturing mistakes made by the overall classification, the classification of tumorous pixels, and non-tumorous pixels respectively. This also solves the imbalance issue by separating the loss values of tumorous and non-tumorous pixels, ensuring that one won't imbalance the other.

The approach taken for creating the inference model involves using three different backbones. There are three combined models used to produce the output. U-Net with DenseNet-121 as the backbone encoder, U-Net with Inception-ResNet-V2 as the backbone encoder, and DeeplabV3Plus with Xception network as the backbone. Their results showed that "using an ensemble of three different networks provided superior segmentation performance compared to using the

networks individually," meaning that the combined outputs of the three models was better than the output of one individual model.

For inference, the preprocessing steps are different than the ones for training. During patching, there is a set overlap on each patch for melding consecutive patches. It was found that 50% between neighboring patches was a balanced amount for "accuracy and computational efficiency." The patch size was also increased by a factor of 4 for inference compared to training for greater accuracy.

This ties into the work in this paper because it's a form of image segmentation performed onto a pathology slide. Their goal was to label sections of pathology slides as cancerous or non-cancerous. The WSI images of pathology slide were patched, and then inference was performed on the patches to generate heatmaps of the tissue. The inference heatmap patches had 50% overlap with their neighboring patches and were stitched together to create one overall image. The end result is a heatmap with certain sections identified as cancerous tissue. This uses a lot of similar fundamental concepts compared to the project in this paper, but is still notably different.

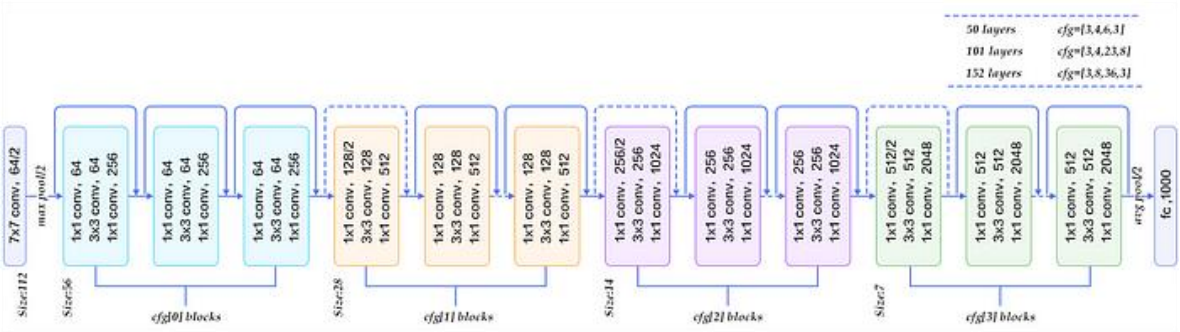
There is a difference between what is being segmented. In the described work, the goal is to segment the cancerous tissue in a WSI image of a pathology slide. This is an application leaning more towards clinical testing, it infers what is cancerous and gives this suggestion to the user. On the other hand, the image

segmentation in this thesis paper does not involve labeling sections of a slide or labeling the slide at all with a diagnosis. The objective is to segment an image of a pathology slide itself and not apply any analysis to the data within the slide. The form of segmentation is significantly different, and the objectives are as well.

Model Overview

The overall program can be broken down into a few key steps. First, is initialization of the model. The model used a ResNet-50 architecture as a backbone with a segmentation head. To properly utilize ResNet-50, the last layers were removed. This is because the last two layers of ResNet-50 are part of its fully connected layer, which will not be needed here for a single object image segmentation.

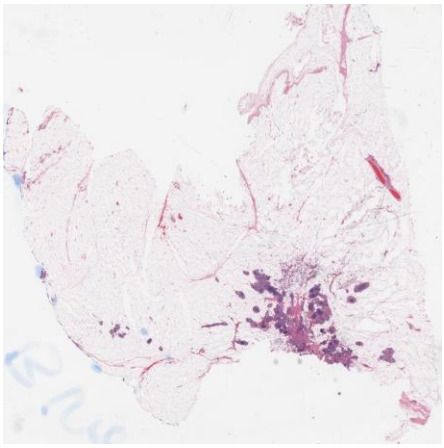
More detailed image of ResNet-50 Architecture



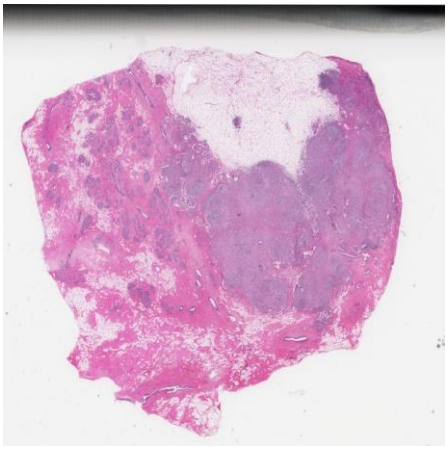
The next step is to preprocess is to split the dataset into training and validation images. The dataset of this project comprised of a public data set of whole slide pathology slide images, photos of the same images printed onto paper, and photos of a textbook with pathology slides on them. The photos were taken using and iPhone 12 camera. For splitting the dataset, a 70-30 split of the first two data sources, and the pictures of pathology slides on paper were saved for final testing. All of these images had an attached image mask. For the images from the

Dryad dataset, entire image was the mask bounding area. This is because they dataset consisted of only whole slide imaging (also known as WSI) of pathology slides. This meant the entire image was truly a pathology slide. It should be noted that small selection of the images did have inconsistencies in their scanning but were kept in to be true to real life scenarios.

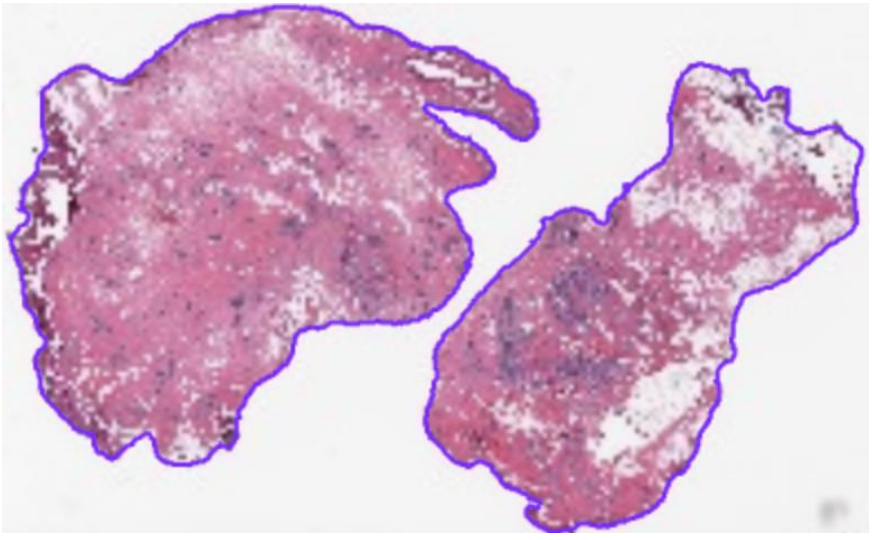
Writing on the bottom left



Scanning mistake on top right



The rest of the dataset was manually annotated with image masks.



Example of annotated masks, the area in the purple box represents the H&E slide

In total, the training and validation dataset consisted of 584 images of traditional H&E WSI and 53 images taken from a phone camera and the test dataset consisted of 20 images of WSI on paper.

After splitting the dataset, the next step would be training the dataset. All images are preprocessed before being trained to improve accuracy. The images are normalized, and have PyTorch's random flip, invert, and rotation applied to them. A Gaussian blur and color jitter and also applied.

The model is set to accept input and output of images at the resolution of 640x640. This was based on the second section of this program, which was the image search of the pathology slide. The input resolution of the search was 640x640, so a larger resolution would end up as lost information, and a smaller resolution would hinder the search.

The training function utilized masks for measuring accuracy. Everything inside of the image masks bounding area was labeled as a pathology slide and everything outside of it was grouped together. 100 epochs were used for training, with the epochs containing the final and best results being saved.

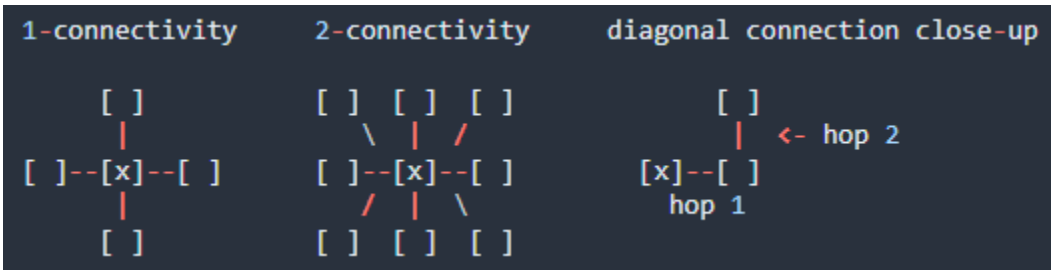
Following training is the function implementing the model. The segmentation function consists of a few parts. The steps were preprocessing, running the model itself, identifying possible noise, and returning the final images.

The first step of the preprocessing is adding padding to the input image. This is done here because it was found the mode was disproportionate towards white bounding areas found within input. The theory behind this is that because the WSI dataset consistently had a bounding white space on each image, it made the model flag false positives. By adding a bounding area and later removing it, the model performed much better on real world scenarios. The image is then resized to 640,640 resolution with the padding to be inputted into the model.

The next step was removing possible noise received from the model. The received output is converted into a heatmap, and sections of certain density are grouped together.

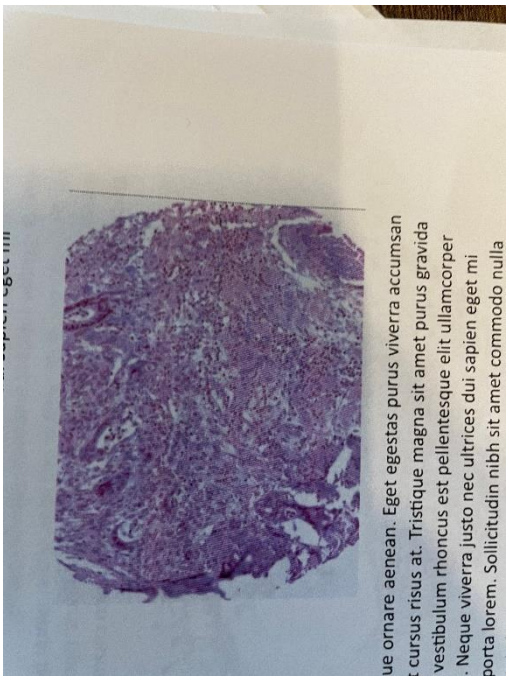
For extracting dense areas, the heatmap is used to filter out pixels below a certain threshold. The skimage package function “measure.label” is used. The function checks to see if any pixels are “neighbors,” meaning they are touching pixels, and groups neighboring pixels together based on that. It returns coordinates for a bounding box, using the extrema’s of the pixel grouping to define the box points.

Image example of connectivity from Skimage

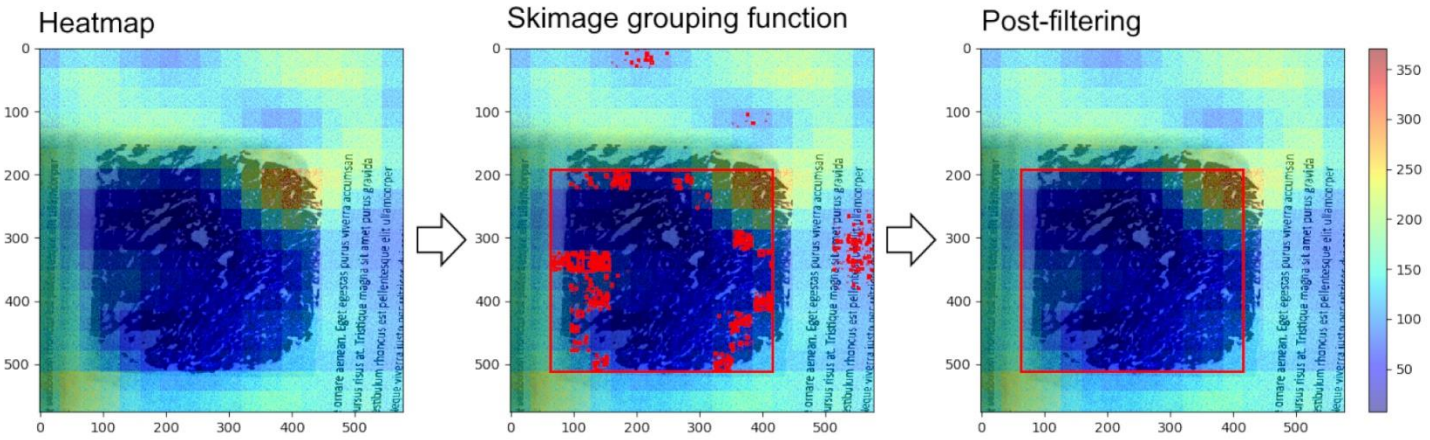


Any grouping of pixels smaller than 3000 pixels was filtered out. For reference, the total pixel count of a 640 by 640 image is 409,600 pixels. For this to be accurate, the assumption is made that grouping of pixels less than 0.75% of the image are safe to ignore. The filtered bounding boxes from measure.label are returned for extraction of the pathology slide. Below is a visual representation of the process:

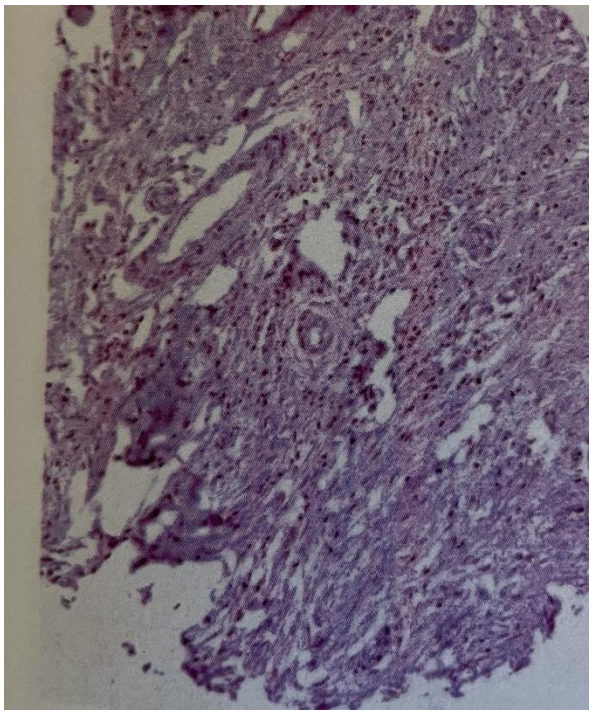
This is an example of an input image:



Here is the shown image being processed:



Output resulted crop:



Code

Below is the most relevant sections of code from the overview:

Training Image transformations:

```
transform = transforms.Compose([
    transforms.Resize((640, 640)),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomInvert(p=.5),
    transforms.RandomRotation(degrees=(-15, 15)),
    transforms.GaussianBlur(15),
    transforms.ColorJitter(),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225]),
])

```

Preprocess transformations:

```
    preprocess = transforms.Compose([
        # Resize to match the model's input size
        transforms.Resize((640, 640)),
        transforms.Pad(30),
        transforms.Resize((640, 640)),
        transforms.ToTensor(),           # Convert to tensor
        transforms.Normalize(mean=[0.485, 0.456, 0.406],std=[0.229, 0.224, 0.225]),

    ])

```

Training loop:

```
num_epochs = 13
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model = model.to(device)
best = 1000000000000000000
# Training loop
for epoch in range(num_epochs):
    model.train()
    total_loss = 0.0  # Accumulate the loss across batches
    for images, masks in train_loader:
        images, masks = images.to(device), masks.to(device)
        masks = masks.long()
        # Zero the gradients
        optimizer.zero_grad()

        # Forward pass
        outputs = model(images)

        # Compute the loss
        loss = criterion(outputs, masks)

        # Backpropagation
        loss.backward()

        # Update weights
        optimizer.step()

        # Accumulate the loss for this batch
        total_loss += loss.item()

    # Calculate the average loss for the epoch
    avg_loss = total_loss / len(train_loader)

    # Validation loop
    model.eval()
    total_val_loss = 0.0
    with torch.no_grad():
        for images, masks in valid_loader:
            images, masks = images.to(device), masks.to(device)
            masks = masks.long()
            # Forward pass
            outputs = model(images)

            # Compute the validation loss
            val_loss = criterion(outputs, masks)
            total_val_loss += val_loss.item()

    # Calculate the average validation loss over all validation batches
    avg_val_loss = total_val_loss / len(valid_loader)

```

Model class:

```
class SegmentationModel(nn.Module):
    def __init__(self, num_classes):
        super(SegmentationModel, self).__init__()
        # Load a pre-trained ResNet-50 model
        self.resnet50 = models.resnet50(pretrained=True)

        # Remove the classification layer (fully connected layer)
        self.backbone = nn.Sequential(*list(self.resnet50.children())[:-2])

        # Define the segmentation head
        self.segmentation_head = nn.Sequential(
            nn.Conv2d(2048, 512, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.ConvTranspose2d(512, num_classes, kernel_size=32, stride=32)  # Upsample to
640x640
        )

    def forward(self, x):
        # Forward pass through the backbone
        x = self.backbone(x)

        # Forward pass through the segmentation head
        x = self.segmentation_head(x)

        return x

```

Heatmap filtering and bounding:

```
    # Apply thresholding to identify high-certainty pixels
    high_certainty_pixels = (heatmap <= threshold)

    # Use connected component analysis to find clusters of high-certainty pixels
    labels, num_features = measure.label(
        high_certainty_pixels, connectivity=2, return_num=True)

    # Calculate bounding boxes for each dense section
    bounding_boxes = []
    for i in range(1, num_features + 1):
        dense_section_indices = np.argwhere(labels == i)
        min_row, min_col = np.min(dense_section_indices, axis=0)
        max_row, max_col = np.max(dense_section_indices, axis=0)
        if (max_row - min_row) * (max_col - min_col) >= 2500:
            bounding_box = (min_row, min_col, max_row, max_col)
            bounding_boxes.append(bounding_box)

    return bounding_boxes

```

Cropping:

```
   1. resolution_ratio_x = (640 / 576) * (int(frame.size[0]) / 640)
 2.     resolution_ratio_y = (640 / 576) * (int(frame.size[1]) / 640)
 3.     for box in boxes:
 4.         section_image = frame.crop(
 5.             (
 6.                 int(box[1] * resolution_ratio_x),
 7.                 int(box[0] * resolution_ratio_y),
 8.                 int(box[3] * resolution_ratio_x),
 9.                 int(box[2] * resolution_ratio_y),
10.             )
11.         )
12.         output_image.paste(
13.             section_image,
14.             (
15.                 int(box[1] * resolution_ratio_x),
16.                 int(box[0] * resolution_ratio_y),
17.                 int(box[3] * resolution_ratio_x),
18.                 int(box[2] * resolution_ratio_y),
19.             )
20.         )
21.     print("output image")
22.     output_image.save("./results/cropped/cropped_" + str(count) + ".png")
23.
```

Masking images:

```
 1. def get_masks(folder_path):
 2.     text_file_names = []
 3.     # Ensure the folder path exists
 4.     if not os.path.exists(folder_path):
 5.         print("doesnt exist")
 6.         return text_file_names
 7.
 8.     # Iterate through each file in the folder
 9.     for filename in os.listdir(folder_path):
10.         file_path = os.path.join(folder_path, filename)
11.         if os.path.isfile(file_path) and filename.endswith('.txt'):
12.             text_file_names.append(filename)
13.
14.     image_width = 640
15.     image_height = 640
16.     all_masks = {}
17.     for text_file in text_file_names:
18.         f=open(folder_path+'/'+text_file, "r")
19.         x = f.read().splitlines()
20.
21.         mask = np.zeros((image_height, image_width), dtype=np.uint8)
22.
23.         all_masks[text_file[:-3]+"jpg"] = mask
24.     return all_masks
25.
26.
```

Applying model to images:

```
model = SegmentationModel(2)
model.load_state_dict(torch.load('best.pth'))

folder_path = './dataset/test/images3'
i = 0
# Iterate through each file in the folder
for filename in os.listdir(folder_path):
    file_path = os.path.join(folder_path, filename)
    if os.path.isfile(file_path) and filename.endswith('.jpg'):
        # Load and preprocess the input image (replace 'input_image.jpg' with the actual path)
        input_image_path = folder_path+'/'+filename
        input_image = Image.open(input_image_path)

        infer(model, input_image, i)
        i+=1
        print(datetime.now()-now)
        now = datetime.now()

```

Turning code into CoreML Model:

```
# Load the PyTorch model
model = SegmentationModel(2)
model.load_state_dict(torch.load("best.pth", map_location=torch.device("cpu")))

# Dummy input - adjust according to your model input shape
input_batch = torch.randn(1, 3, 640, 640)

preprocess = transforms.Compose(
    [
        # Resize to match the model's input size
        transforms.Resize((640, 640)),
        transforms.Pad(30),
        transforms.Resize((640, 640)),
        transforms.ToTensor(),  # Convert to tensor
        # transforms.Normalize(mean=[0.485, 0.456, 0.406],std=[0.229, 0.224, 0.225]),
    ]
)

# Trace the model
trace = torch.jit.trace(model, input_batch)



mlmodel = ct.convert(
    trace,
    inputs=[
        ct.ImageType(name="image",
            shape=(1, 3, 640, 640)
        )
    ],
    minimum_deployment_target=ct.target.macOS13,
)

mlmodel.save("SegmentationModel_no_metadata.mlpackage")

mlmodel = ct.models.MLModel("SegmentationModel_no_metadata.mlpackage")

labels_json = {"labels": ["pathology_slide", "null"]}

mlmodel.user_defined_metadata["com.apple.coreml.model.preview.type"] = "imageSegmenter"
mlmodel.user_defined_metadata["com.apple.coreml.model.preview.params"] = json.dumps(
    labels_json
)

mlmodel.save("SegmentationModel_with_metadata.mlpackage")

```

Model class tuned for CoreML:

```
class SegmentationModel(nn.Module):
    def __init__(self, num_classes):
        super(SegmentationModel, self).__init__()
        # Load a pre-trained ResNet-50 model
        self.resnet50 = models.resnet50(pretrained=True)

        # Remove the classification layer (fully connected layer)
        self.backbone = nn.Sequential(*list(self.resnet50.children())[:-2])

        # Define the segmentation head
        self.segmentation_head = nn.Sequential(
            nn.Conv2d(2048, 512, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.ConvTranspose2d(512, num_classes, kernel_size=32, stride=32)  # Upsample to
640x640
        )

    def forward(self, x):
        # Forward pass through the backbone
        x = self.backbone(x)

        # Forward pass through the segmentation head
        x = self.segmentation_head(x)

        return x

```

Experimentation

On the path of the making a final product, the model had undergone experimentation and changes. The first large change was switching from YoloV7 to ResNet-50 backbone. The first iteration had used YoloV7 and was able to segment images, but it was found that the model architecture had a hard time adapting to real world pictures. Compared to ResNet-50 backbone, it picked up a significant amount of noise from slight lighting changes and had a hard time being trained.

For deciding on the transformations of the training image preprocessing, various testing was conducted. At the beginning, only geometric transformations were applied, but this was quickly found to be incorrect, and more transformations were implemented. A normalization transformation was introduced, and the values for it were borrowed from ImageNet's standard. The Blur, Jitter, and Invert transformations were slowly introduced by testing a small number of epochs using varying image transformations from PyTorch's library.

Deciding on the patching process was the most rigorous. Before properly patching the image, the first approach was to truncate the image given a threshold for the heatmap. This was found to be too simple and unable to produce notable results at all. The second attempt was implementing edge detection on the area above a certain threshold to extract the pathology slide. While this approach could extract the slide, it was inconsistent with its results since some slides could be

cleanly split in between, and offered no robustness in filtering out noisy results from the model. The next approach was checking each pixel to find nearest neighbors. This gave mixed results and was computationally expensive. Its concept is very similar to the final application using skimage, but the simply written approach was significantly less consistent and more resource demanding.

The model consistently finds excess white space to be part of an extracted pathology slide, and most troublingly around the border of the entire image. Padding was introduced here to give a buffer around the image, and its removed after being fed to the model. The primary concern here was a loss of resolution, but for the final cropped output the original photo is referenced to ensure minimal loss of data during extraction.

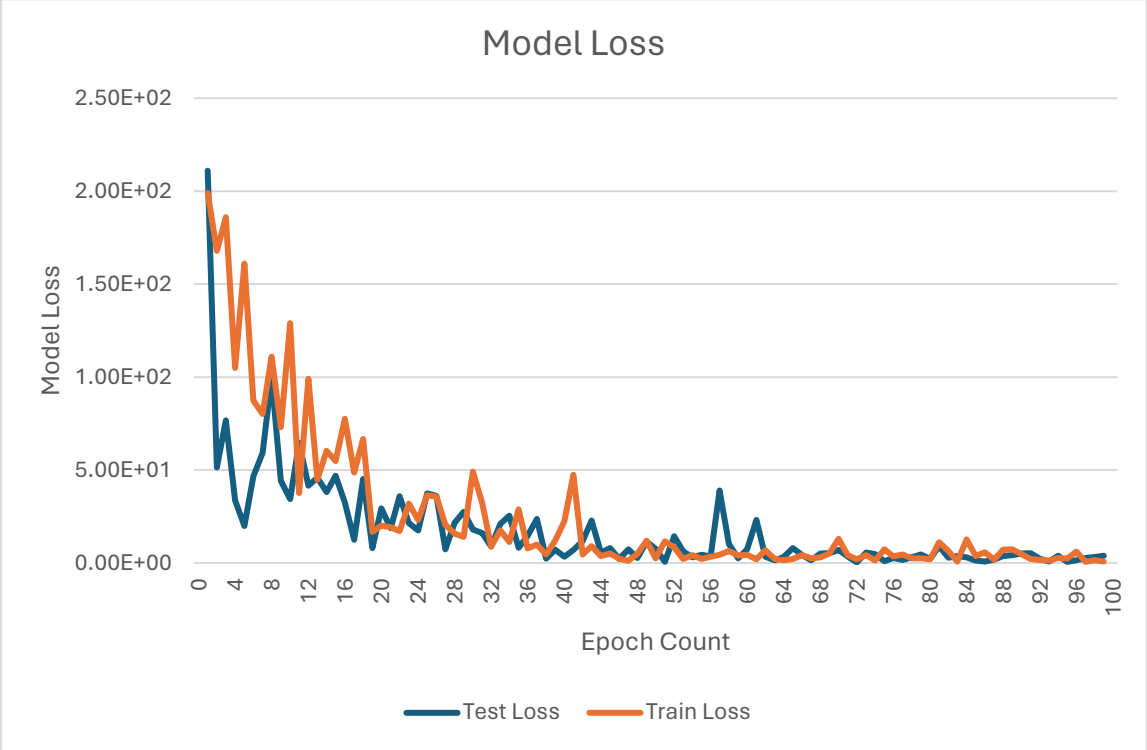
Applying a thresholding value to the heatmap was introduced after using normalization for preprocessing, as ensured the heatmap would stick within a more consistent range. This approach, while rudimentary, is effective enough to filter out a majority of the image before applying the patching techniques.

An attempt was made to identify sections based on how pink and blue they might be. H&E pathology slides are consistent in their coloring, with the Hematoxylin and Eosin always dyeing in their same shades. This did not go very far though, as even though H&E slides themselves are very consistent, the other factors in a photo, such as lighting and contrast, vary significantly enough to make

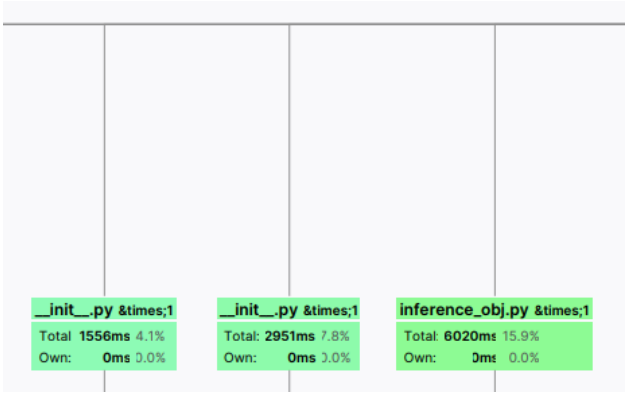
it difficult to extract information purely based on color. The goal was the normalize the image, and then convert it to a CIELAB color space, where each pixel is given LAB values. L values represent lighting, A values represent red/green hues, and B represent the yellow/blue hues. A and B channels were used to define a pink hue range found in H&E slides, but the results were inconsistent.

Results

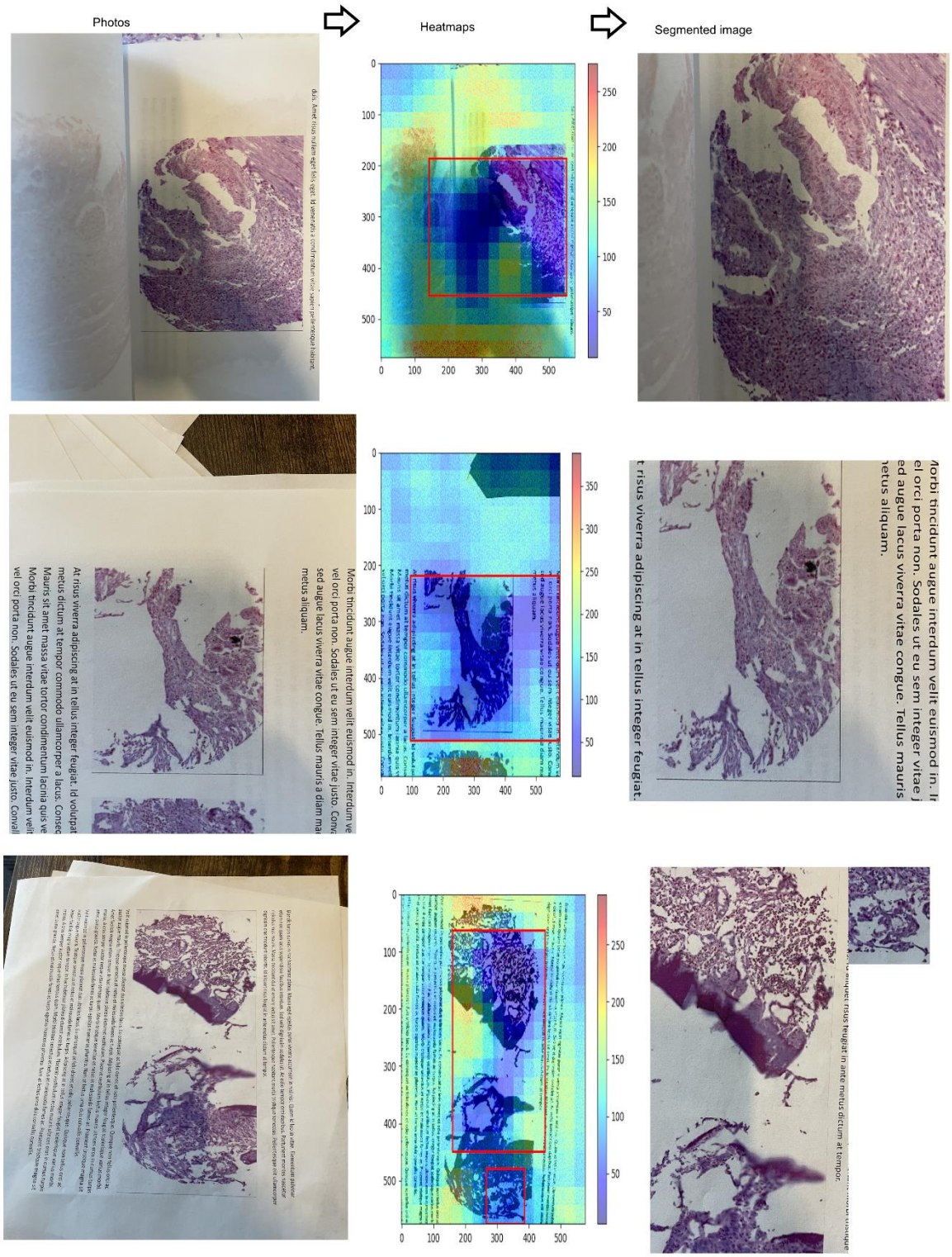
Below is the graphed loss from training:



Below is the time analysis of the program. In total, it takes around 11 seconds to apply the model. Around 6 of those seconds is loading the weights, and with the model already loaded, it takes 4-5 seconds to apply patching and save the image.



Here are examples of results from the final test set:



Conclusion

The goal was to create a model capable of segmenting pathology slides from photos. The created model can consistently identify a pathology slide given a reasonable setting and extract it. It's efficient enough to be applied on multiple platforms with reasonable run-time.

There are a few cases where improvements are needed. When there is more than one pathology slide within the view, if they are not distinctly separated by a non-white space, the model will combine them into one. This is a more novel case as it's not often that two slides would sit next to each other, but it is a case to keep in mind.

The second case is when the slide is surrounded by bodies of text. Small sections of nearby text can be confused as part of a slide. A dark shade of a slide is similar to the shade of normal text, and in order to eliminate this more thorough bounding must be considered.

Lastly, testing on more mobile devices in required. In this project, only iPhone 12 was tested to run the model. Using Apples CoreML package, the model was ported to work on Apple iPhones. In theory the ported model should be able to function on all Apple platforms. Android was not tested though, and the platform should be considered. The concept of tracing a model for Core ML and PyTorch

Mobile for Android is the same, implying that the functionality would carry over to Android, but more proper testing should be done.

Future Works

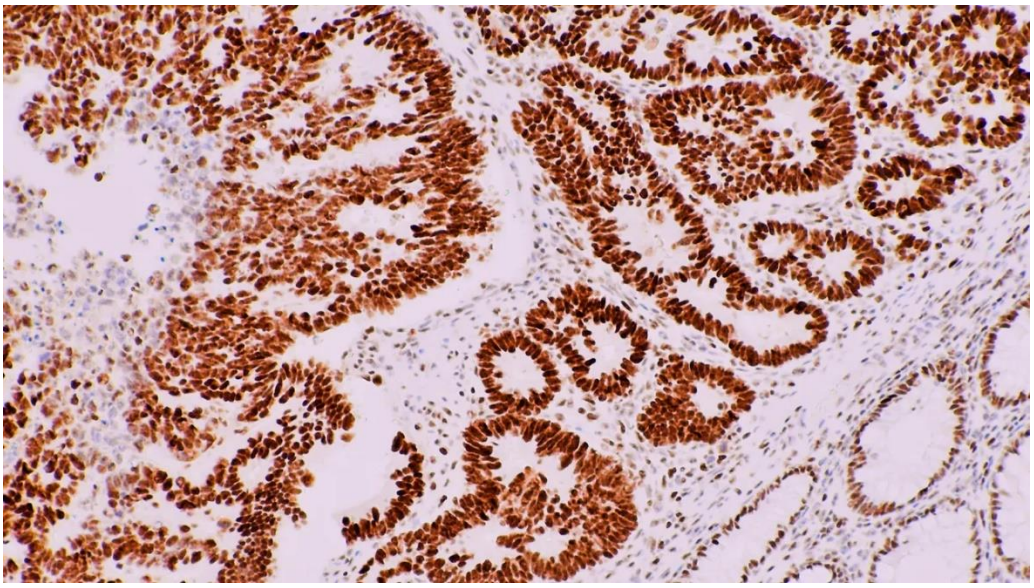
For future improvements, there should be more rigorous bounding, testing of various other backbones, and better frameworks for other platforms. For the bounding, the current implementation does not guarantee that there will be no overlap from nearby objects. An alternative approach to this would be to add more vertices to the bounding. More vertices would allow for much more accurate bounding, by ensuring that the object is more properly outlined.

There are a multitude of other backbones that could be used for testing. While ResNet-50 was able to produce acceptable results, it is nearing a decade old and newer models may be able to produce better results.

Implementations were tested on a Raspberry Pi, Jetson Nano, and iPhone. While they were able to run across these platforms, it was done in the same fashion as a desktop, and not utilizing the uniqueness of the different platforms. Identifying strengths that could be applied to this project and leveraging them on these platforms could provide more thorough coverage.

Other implementations of medical imagery image segmentation can also be considered. There are other forms of medical imagery that have similar shape and consistency to pathological slides. An example of this would be

immunohistochemistry images:



Immunohistochemistry Techniques, Strengths, Limitations and Applications (Verma 2024)

There are a many medical images that are loosely similar to pathology slides in terms of their physical attributes that could be considered for segmentation and labeling. This kind of implementation would require new datasets for each type of imaging and training on the new types.

Works Cited

Haematoxylin & Eosin (H&E) Staining. 16 Dec. 2020, https://www.cancer.ox.ac.uk/support/THL/HE-Staining.

“What Is Computer Vision? | IBM.” IBM, 20 Mar. 2024, https://www.ibm.com/topics/computer-vision

“What Is Image Segmentation? | IBM.” IBM, 4 Mar. 2024, https://www.ibm.com/topics/image-segmentation.

Elharrouss, Omar et al. “Backbones-Review: Feature Extraction Networks for Deep Learning and Deep Reinforcement Learning Approaches.” ArXiv abs/2206.08016 (2022): n. pag.https://arxiv.org/pdf/1512.03385

Mukherjee, Suvaditya. “The Annotated ResNet-50.” Towards Data Science, 18 Aug. 2022, https://towardsdatascience.com/the-annotated-resnet-50-a6c536034758.

Rastogi, Aditi. “ResNet50.” Dev Genius, 14 Mar. 2022, https://blog.devgenius.io/resnet50-6b42934db431.

Lathashreeharisha. “Dice Coefficient! What Is It?” Medium, 19 Feb. 2023, https://medium.com/@lathashreeh/dice-coefficient-what-is-it-ff090ec97bda.

Cruz-Roa, Angel et al. (2018). Data from: High-throughput adaptive sampling for whole-slide histopathology image analysis (HASHI) via convolutional neural networks: application to invasive breast cancer detection [Dataset]. Dryad. https://doi.org/10.5061/dryad.1g2nt41

Angel. “High-Throughput Adaptive Sampling for Whole-Slide Histopathology Image Analysis (HASHI) via Convolutional Neural Networks: Application to Invasive Breast Cancer Detection.” PLOS ONE, 24 May 2018, https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0196828.

J. Deng, W. Dong, R. Socher, L. -J. Li, Kai Li and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, USA, 2009, pp. 248-255, doi: 10.1109/CVPR.2009.5206848.

Khened, M., Kori, A., Rajkumar, H. et al. A generalized deep learning framework for whole-slide image segmentation and analysis. Sci Rep 11, 11579 (2021). https://doi.org/10.1038/s41598-021-90444-8

Brownlee, J. (2020, December 22). A gentle introduction to cross-entropy for Machine Learning. MachineLearningMastery.com. https://machinelearningmastery.com/cross-entropy-for-machine-learning

Verma, Aditi. “Immunohistochemistry Techniques, Strengths, Limitations and Applications.” Analysis & Separations from Technology Networks, 5 July 2022,

http://www.technologynetworks.com/analysis/articles/immunohistochemistry-techniques-strengths-limitations-and-applications-363107.