IEEE SPS GS MSP CUP 2024

Data Science Challenge

Tiger Grand Challenge
Transforming Cancer Pathology with AI-Driven Diagnostics

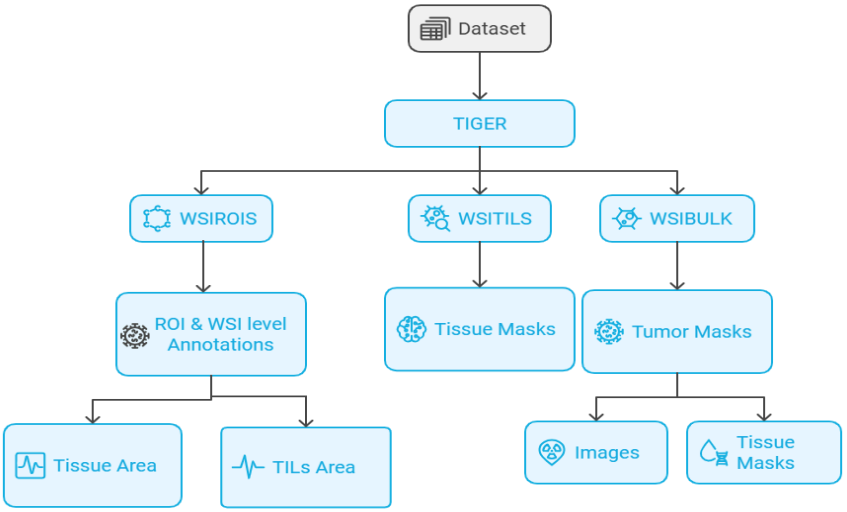
TEAM

- Members: Viren Mehta, Perin Modi, Tirth Patel
- Email: virenmehta711@gmail.com
- Presentation: MSP Cup - TIGER - Presentation

DATASET

- **wsibulk**: Contains whole-slide images (WSIs) and manual annotations for tumour bulk and tissue-background masks, provided in formats: TIF, XML.
- **wsirois**: Includes cropped regions of interest (ROIs) with detailed annotations for tissue and cells, including images and masks in PNG format, and cell annotations in COCO format.
    1. **TCGA**: TNBC cases from TCGA-BRCA archive (n =151). The annotations provided for this dataset were generated by adapting the publicly available BCSS and NuCLS datasets.
    2. **RUMC**: 26 cases of TNBC and HER2+ cases from Radboud University Medical Center (Netherlands), with annotations by a panel of board-certified breast pathologists.
    3. **JB**: 18 cases of TNBC and HER2+ cases from Jules Bordet Institute (Belgium). Annotations for these were made by a panel of board-certified breast pathologists.

- **wsitils**: Comprises whole-slide images, tissue-background masks, and TIL scores in CSV format for evaluating automated TIL scoring models.



TASKS

1. Detection of Lymphocytes and Plasma Cells

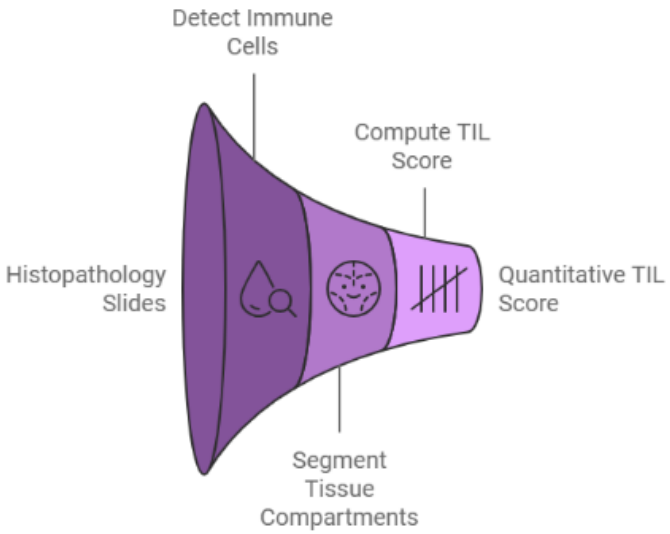
- Identify and classify the key immune cells involved in TILs.
- Lymphocytes (T cells and B cells) and plasma cells.

2. Segmentation of Invasive Tumor and Tumor-Associated Stroma

- Accurately segment the tissue compartments in the histopathology slides.
- Distinguish between invasive tumor and tumor-associated stroma.

3. Computation of Automated TIL Score

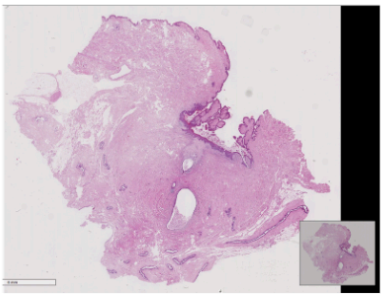
- Generate a quantitative TIL score for each slide.
- Integrate detection and segmentation results to compute the TIL score.



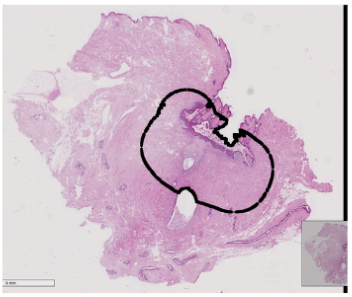
Approach

Task	Patch Size	Model	Procedure
Segmentation	512x512	Unet Segmentation	•
Detection	512x512	Yolo v8 Detection	• Used COCO format annotations from coco.json for lymphocytes and plasma cells. • Conversion of annotations from json to yolov8 supportable format. • Applied annotations to 1,879 images to create a labeled dataset.
TILs for Computation			

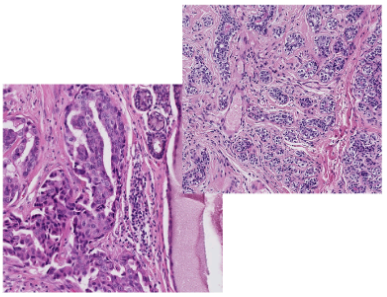
Conversion of Whole Slide Images to Patches



WSI



WSI Mask



PATCHES

```
import patchify
import numpy as np
import matplotlib.pyplot as plt
from wholeslidedata import WholeSlideImage
from wholeslidedata.interoperability.openslide.backend import
OpenSlideWholeSlideImageBackend
```

```
# Load your WSI
wsi =
WholeSlideImage(path='../data/wsiroi/wsi/tissue-mask/133S_tissue.tif',
backend=OpenSlideWholeSlideImageBackend)

# Get the slide at a specific spacing
slide = wsi.get_slide(spacing=8.0)

# Define the patch size
patch_size = 256

# Patchify the slide
patches = patchify.patchify(slide, (patch_size, patch_size, 3),
step=patch_size)

# Flatten the patches array
patches = patches.reshape(-1, patch_size,patch_size,3)
```

```
# how can i save these patches in a folder
import os

# Create a directory to save the patches
output_dir = 'tissue_masks_patches'
os.makedirs(output_dir, exist_ok=True)

# Save each patch to the output directory
for i, patch in enumerate(patches):
    patch_filename = f'patch_{i}.png'
    patch_path = os.path.join(output_dir, patch_filename)
    plt.imsave(patch_path, patch)

print(f"All {len(patches)} patches have been saved to {output_dir}")
```

```
All 154 patches have been saved to tissue_masks_patches
```

```
# Load your WSI
wsi = WholeSlideImage(path='../data/wsiroi/wsi/images/133S.tif',
backend=OpenSlideWholeSlideImageBackend)

# Get the slide at a specific spacing
slide = wsi.get_slide(spacing=8.0)

# Define the patch size
patch_size = 256

# Patchify the slide
patches = patchify.patchify(slide, (patch_size, patch_size, 3),
step=patch_size)

# Flatten the patches array
patches = patches.reshape(-1, patch_size,patch_size,3)
```

Detection of Lymphocytes and Plasma Cells

Dataset with annotations: wsiroisImages

Training Script

```
from ultralytics import YOLO

# # Load a model
model = YOLO("yolov8n.pt")  # load a pretrained model (recommended for
training)

# # Train the model with 2 GPUs
results = model.train(data="/kaggle/working/wsiroisImages-1/data.yaml",
epochs=1000, imgsz=256, device=[0, 1], patience=20)
```

```
# # Train the model with 2 GPUs
results = model.train(data="/kaggle/working/wsiroisImages-1/data.yaml",
epochs=1000, imgsz=256, device=[0, 1], patience=20, batch=32,
workers=10)
```

We have trained the dataset on 2 models: nano and medium.

```
import shutil
import os
from IPython.display import FileLink, display, HTML

def zip_folder(input_folder_path):
    # Define the name for the zip file
    zip_file_path = input_folder_path + '.zip'

    # Zip the folder
    shutil.make_archive(input_folder_path, 'zip', input_folder_path)
```

```
    # Check if the zip file is created successfully
    if os.path.exists(zip_file_path):
        print(f"Folder zipped successfully at: {zip_file_path}")
    else:
        print("Error in zipping the folder.")

    return zip_file_path

def create_download_link(zip_file_path):
    # Create a download link for the zip file
    display(HTML(f'<a href="{zip_file_path}" download>Click here to
download {zip_file_path}</a>'))

    # JavaScript to automatically trigger the download
    display(HTML(f"""
        <script>
        var link = document.createElement('a');
        link.href = '{zip_file_path}';
        link.download = '{os.path.basename(zip_file_path)}';
        document.body.appendChild(link);
        link.click();
        document.body.removeChild(link);
        </script>
    """))

# Example usage
input_folder_path = '/kaggle/working/runs/detect/train'  # Replace with
your folder path
zip_file_path = zip_folder(input_folder_path)

# Automatically trigger the download
create_download_link(zip_file_path)
```

Training Parameters

```
task: detect
mode: train
model: yolov8m.pt
data: /kaggle/working/wsiroisImages-2/data.yaml
epochs: 10000
time: null
patience: 25
```

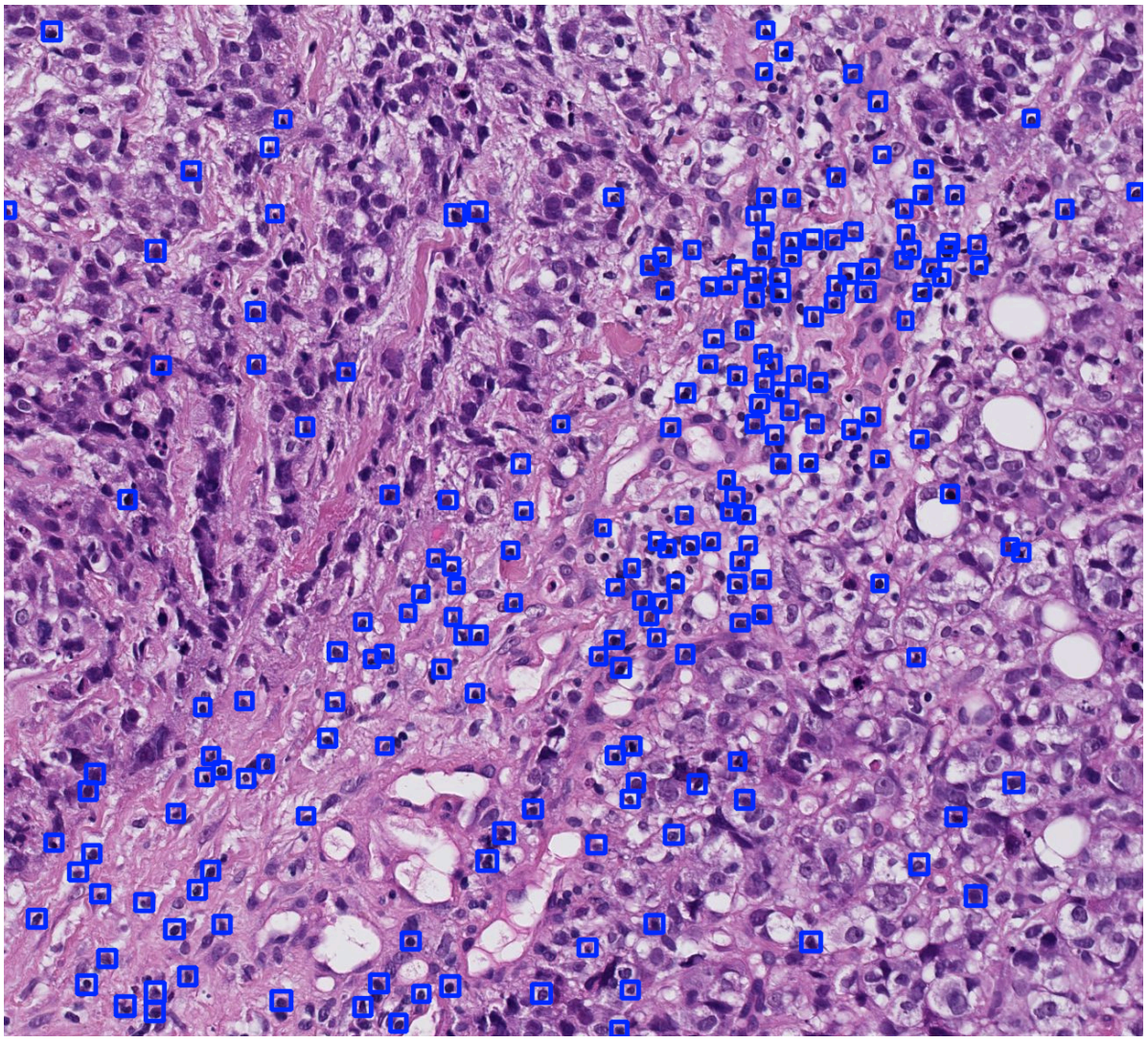
```yaml
batch: 32
imgsz: 512
save: true
save_period: -1
cache: false
device:
- 0
- 1
workers: 10
project: null
name: train
exist_ok: false
pretrained: true
optimizer: auto
verbose: true
seed: 0
deterministic: true
single_cls: false
rect: false
cos_lr: false
close_mosaic: 10
resume: false
amp: true
fraction: 1.0
profile: false
freeze: null
multi_scale: false
overlap_mask: true
mask_ratio: 4
dropout: 0.0
val: true
split: val
save_json: false
save_hybrid: false
conf: null
iou: 0.7
max_det: 300
half: false
dnn: false
plots: true
source: null
vid_stride: 1
stream_buffer: false
visualize: false
augment: false
agnostic_nms: false
classes: null
retina_masks: false
```

```
embed: null
show: false
save_frames: false
save_txt: false
save_conf: false
save_crop: false
show_labels: true
show_conf: true
show_boxes: true
line_width: null
format: torchscript
keras: false
optimize: false
int8: false
dynamic: false
simplify: false
opset: null
workspace: 4
nms: false
lr0: 0.01
lrf: 0.01
momentum: 0.937
weight_decay: 0.0005
warmup_epochs: 3.0
warmup_momentum: 0.8
warmup_bias_lr: 0.1
box: 7.5
cls: 0.5
dfl: 1.5
pose: 12.0
kobj: 1.0
label_smoothing: 0.0
nbs: 64
hsv_h: 0.015
hsv_s: 0.7
hsv_v: 0.4
degrees: 0.0
translate: 0.1
scale: 0.5
shear: 0.0
perspective: 0.0
flipud: 0.0
fliplr: 0.5
bgr: 0.0
mosaic: 1.0
mixup: 0.0
copy_paste: 0.0
auto_augment: randaugment
```

```
erasing: 0.4
crop_fraction: 1.0
cfg: null
tracker: botsort.yaml
save_dir: runs/detect/train
```

Inference:

```
image = '../data/wsiroi/roi/tissue_cells/images/100B_[21444, 12438, 22739, 13618].png'
model = YOLO('yolov8-lymphocytes/train/weights/best.pt')
results = model(image)
results[0].show(labels=False)
```



Segmentation

Image Preprocessing : For image preprocessing, patches of 512x512 are created from all the images instead of resizing. This approach is preferred because resizing can lead to the loss of useful information from the images. Using patches helps retain detailed information and avoids the loss of important features that might occur with resizing. By working with 512x512 patches, more granular details are preserved, which is crucial for accurate segmentation, especially in medical imaging or other detailed tasks.

```
def process_image_pair(img_path, mask_path, img_dir, mask_dir, output_dir, patch_size):
    img = cv2.imread(os.path.join(img_dir, img_path))
    mask = cv2.imread(os.path.join(mask_dir, mask_path))

    if img is None or mask is None:
        return f"Failed to load {img_path} or {mask_path}"

    img_patches = patchify(img, (patch_size, patch_size, 3), step=patch_size).reshape(-1, patch_size, patch_size, 3)
    mask_patches = patchify(mask, (patch_size, patch_size, 3), step=patch_size).reshape(-1, patch_size, patch_size, 3)

    for i, (img_patch, mask_patch) in enumerate(zip(img_patches, mask_patches)):
        unique_id = uuid.uuid4().hex
        patch_filename = f'patch_{unique_id}.png'
        img_patch_path = os.path.join(output_dir, 'images', patch_filename)
        mask_patch_path = os.path.join(output_dir, 'masks', patch_filename)

        plt.imsave(img_patch_path, img_patch)
        plt.imsave(mask_patch_path, mask_patch)

    return f"Processed {img_path}"
```

```
def main():
    img_dir = 'roi_tissue\\bcss\\images'
    mask_dir = 'roi_tissue\\bcss\\masks'
    output_dir = 'patches'
    patch_size = 512

    os.makedirs(os.path.join(output_dir, 'images'), exist_ok=True)
    os.makedirs(os.path.join(output_dir, 'masks'), exist_ok=True)

    img_paths = os.listdir(img_dir)
    mask_paths = os.listdir(mask_dir)

    with ThreadPoolExecutor(max_workers=os.cpu_count()) as executor:
        futures = [executor.submit(process_image_pair, img_path, mask_path, img_dir, mask_dir, output_dir, patch_size)
                   for img_path, mask_path in zip(img_paths, mask_paths)]

        for future in tqdm(futures, total=len(futures), desc="Processing images"):
            print(future.result())

main()
```

- **Color/Stain Normalization in Histopathology :**The accuracy of histopathology image analysis can be compromised by color variations due to different scanning equipment, staining methods, and tissue reactivity. These variations affect the performance of computer-aided diagnosis and can impact pathologists' evaluations. The standard histopathology imaging process involves several steps: tissue preparation and whole slide imaging, collection and fixation, dehydration and clearing, paraffin embedding, microtomy, staining, mounting, and digitalization. Hematoxylin and eosin are commonly used routine laboratory stains; hematoxylin imparts a purple or blue color to cells and nuclei, while eosin gives a pinkish hue to eosinophilic structures. Color/stain normalization adjusts the mean color of one image to another to minimize these color variations and improve diagnostic accuracy.

```python
def do_convert(targets, sources, save_root, source_path):
    os.makedirs(os.path.join(save_root, 'images'), exist_ok=True)
    os.makedirs(os.path.join(save_root, 'masks'), exist_ok=True)
    try:
        for target in tqdm(targets):
            target = cv2.imread(target)
            if target is None:
                print(f"Failed to load target image: {target}")
                continue
            target = cv2.cvtColor(target, cv2.COLOR_BGR2RGB)
            for source in tqdm(sources):
                original_name = os.path.basename(source)
                unique_name = f"{uuid.uuid4().hex}.png"
                source = cv2.imread(source)
                if source is None:
                    print(f"Failed to load source image: {source}")
                    continue
                source = cv2.cvtColor(source, cv2.COLOR_BGR2RGB)
                stain_unmixing_routine_params = {
                    'stains': ['hematoxylin', 'eosin'],
                    'stain_unmixing_method': 'macenko_pca',
                }
                tissue_rgb_normalized = deconvolution_based_normalization(
                    source,
                    im_target=target,
                    stain_unmixing_routine_params=stain_unmixing_routine_params
                )
                source_change = cv2.cvtColor(tissue_rgb_normalized, cv2.COLOR_RGB2BGR)
                image_result_path = os.path.join(save_root, 'images', unique_name)
                source_mask_path = os.path.join(source_path, 'masks', original_name)
                dest_mask_path = os.path.join(save_root, 'masks', unique_name)
                if os.path.exists(source_mask_path):
                    shutil.copy2(source_mask_path, dest_mask_path)
                    cv2.imwrite(image_result_path, source_change)
                else:
                        print(f"No corresponding mask found for {original_name}")
```

```python
    except:
        print(f"Error in {target}")

    print("Conversion and copying completed.")
```

```python
tgs_images=[]
src_images=[]
tgs_images_paths=os.walk('roi_tissue\\bcss\images')
src_images_paths=os.walk('roi_tissue\\cells\images')
for path,dir_list,file_list in tgs_images_paths:
    for file in file_list:
        tgs_images.append(path+'\\'+file)
for path,dir_list,file_list in src_images_paths:
    for file in file_list:
        src_images.append(path+'\\'+file)
```

```python
source_path='roi_tissue\\bcss'
do_convert(targets=src_images, sources=tgs_images, source_path=source_path,save_root='color_normalized1')
```

Network Architecture : We employed the **Efficient-UNet** model for segmentation tasks, implemented in TensorFlow. This model uses the **EfficientNet-B0** variant as its encoder backbone, **pre-trained** on **ImageNet**, which enhances feature extraction and domain generalizability through transfer learning.

```
[ ]:
SIZE_X = 512
SIZE_Y = 512
n_classes = 7
BATCH_SIZE = 2
AUTOTUNE = tf.data.AUTOTUNE
IMAGENET_MEAN = np.array([0.485, 0.456, 0.406])
IMAGENET_STD = np.array([0.229, 0.224, 0.225])
```

+ Code + Markdown

```
[ ]:
def imagenet_normalization(image):
    """Normalize the image using ImageNet mean and standard deviation."""
    image = tf.cast(image, tf.float32) / 255.0
    mean = tf.constant(IMAGENET_MEAN, dtype=tf.float32)
    std = tf.constant(IMAGENET_STD, dtype=tf.float32)
    image = (image - mean) / std
    return image
```

```
[ ]:
def convert_mask(mask):
    """Convert 3-channel mask to a single channel with class labels."""
    # Define the mapping of RGB colors to class labels
    color_to_class = {
        (0, 255, 255): 0,  # Invasive tumor (Cyan)
        (0, 255, 0): 1,    # Tumor-associated stroma (Green)
        (255, 255, 0): 2,  # In-situ tumor (Yellow)
        (255, 165, 0): 3,  # Healthy glands (Orange)
        (255, 0, 0): 4,    # Necrosis not in-situ (Red)
        (255, 0, 255): 5,  # Inflamed stroma (Magenta)
        (0, 0, 255): 6,    # Rest (Blue)
    }
    mask = tf.cast(mask * 255, tf.uint8)
    mask_out = tf.zeros_like(mask[..., 0], dtype=tf.uint8)

    for color, cls in color_to_class.items():
        color_mask = tf.reduce_all(tf.equal(mask, color), axis=-1)
        mask_out = tf.where(color_mask, tf.cast(cls, tf.uint8), mask_out)

    return mask_out
```

```
[ ]:
base_path = "/kaggle/input/patches-512-v2/patches"
image_path = os.path.join(base_path, "images")
mask_path = os.path.join(base_path, "masks")

def preprocess_image(image, mask):
    image = imagenet_normalization(image)
    mask = tf.squeeze(mask, axis=-1)
    mask = tf.one_hot(tf.cast(mask, tf.int32), depth=7)
    return image, mask
image_dataset = tf.keras.utils.image_dataset_from_directory(
    image_path,
    labels=None,
    color_mode='rgb',
    batch_size=BATCH_SIZE,
    image_size=(SIZE_Y, SIZE_X),
    shuffle=True
)

mask_dataset = tf.keras.utils.image_dataset_from_directory(
    mask_path,
    labels=None,
    color_mode='grayscale',
    batch_size=BATCH_SIZE,
    image_size=(SIZE_Y, SIZE_X),
    shuffle=True
)
dataset = tf.data.Dataset.zip((image_dataset, mask_dataset))
```

```
image_dataset = tf.keras.utils.image_dataset_from_directory(
    image_path,
    labels=None,
    color_mode='rgb',
    batch_size=BATCH_SIZE,
    image_size=(SIZE_Y, SIZE_X),
    shuffle=True
)

mask_dataset = tf.keras.utils.image_dataset_from_directory(
    mask_path,
    labels=None,
    color_mode='grayscale',
    batch_size=BATCH_SIZE,
    image_size=(SIZE_Y, SIZE_X),
    shuffle=True
)
dataset = tf.data.Dataset.zip((image_dataset, mask_dataset))
dataset = dataset.map(preprocess_image, num_parallel_calls=AUTOTUNE)
dataset_size = tf.data.experimental.cardinality(dataset).numpy() * BATCH_SIZE
train_size = int(0.8 * dataset_size)
train_dataset = dataset.take(train_size // BATCH_SIZE)
val_dataset = dataset.skip(train_size // BATCH_SIZE)
train_dataset = train_dataset.shuffle(buffer_size=1000).prefetch(AUTOTUNE)
val_dataset = val_dataset.prefetch(AUTOTUNE)
```

Model Training : The training involved using a slightly different learning rate for the encoder and decoder, set at 1e-4 and 1e-3 respectively.By incorporating "label smoothing" and "maximal restriction" strategies for the Jaccard Loss calculation. The "background" class was excluded from the loss function. Training utilized the **cosine warm-up** strategy with the **Adam optimizer** over 70 epochs, using a batch size of 8. A total of 12,000 patches were sampled each epoch as described previously. The model was evaluated on the validation set after each epoch, and the best model was selected based on the Minimum Loss achieved for the tumor and stroma classes.

The following **Jaccard loss** function was utilized to train the model

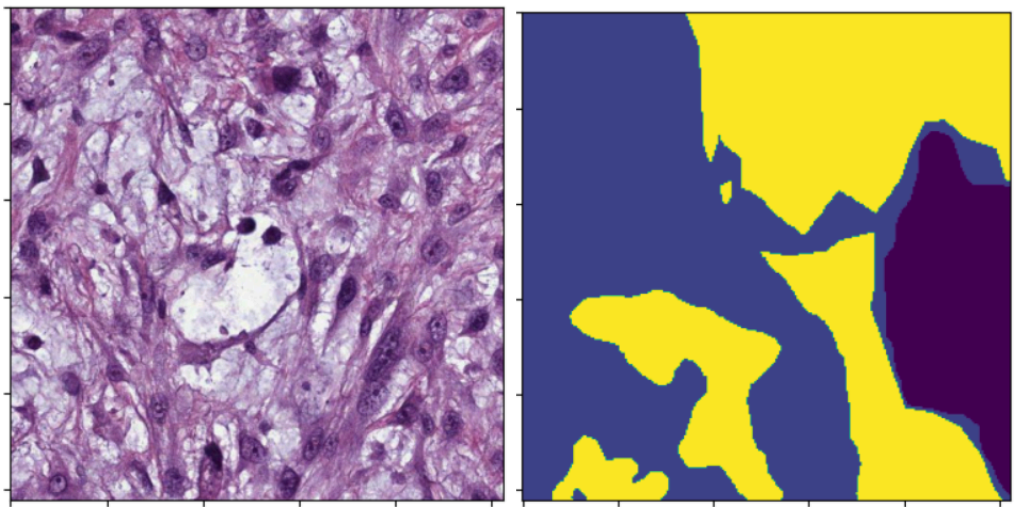
$$\mathcal{L}_{Jaccard}(y_t, y_p) = 1 - \frac{\sum y_t y_p + \epsilon}{\sum y_t^2 + \sum y_p^2 - \sum y_t y_p + \epsilon},$$

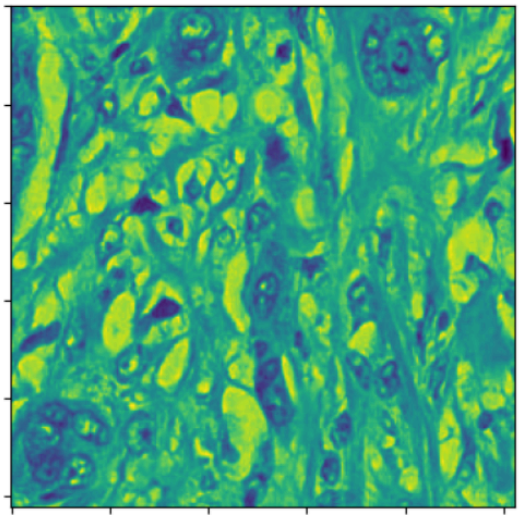
where yp, yt, ǫ are the model prediction, ground truth segmentation, and a constant equal to 1 to avoid division by zero, respectively.

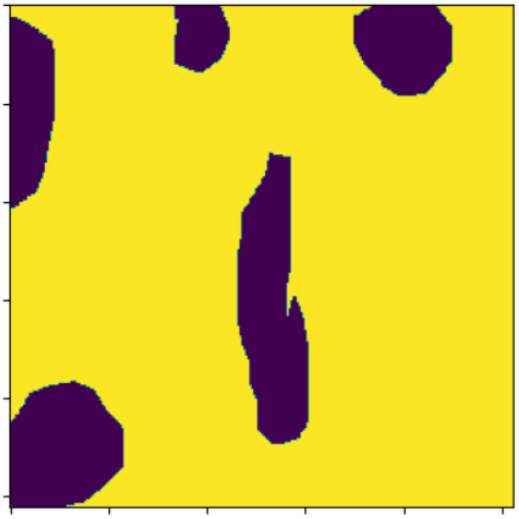
```
[ ]:
    activation='softmax'
    BACKBONE1 = 'efficientnetb0'
    model1 = sm.Unet(BACKBONE1, encoder_weights='imagenet',input_shape=(SIZE_X,SIZE_Y,3), classes=n_classes, a
```

```
LR = 1e-4
cosine_lr_schedule = CosineDecayRestarts(
    initial_learning_rate=1e-3,
    first_decay_steps=10,
)
optim = tf.keras.optimizers.Adam(learning_rate=LR)
metrics = [sm.metrics.IOUScore(threshold=0.5), sm.metrics.FScore(threshold=0.5)]
model1.compile(optim, loss=sm.losses.bce_jaccard_loss, metrics=metrics)
history1 = model1.fit(
    train_dataset,
    batch_size=BATCH_SIZE,
    epochs=10,
    verbose=1,
    validation_data=val_dataset
)
```

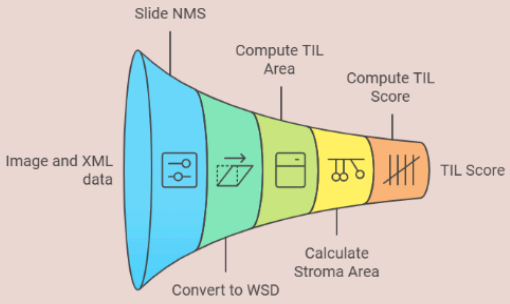
Validation Process : In the validation process, the whole image is divided into 512x512x3 patches and tested on the model. After processing, only the pixels with the maximum value are retained. These pixels are then reassembled to form the full masked image in its original format, without resizing.







Computation of Automated TIL Score



Determine TIL Area: The area of a single TIL in pixels based on its size in micrometers and pixel size.

Total TIL Area: Multiply the number of detected TILs by the area of a single TIL.
Stroma Area Measurement:

Measure Stroma Area: Calculate the area of tumor-associated stroma using the provided mask.

$$\text{TIL Score} = \left(\frac{100 \times \text{Total TIL Area}}{\text{Stroma Area}}\right)$$

Where:

- **Total TIL Area** = Number of TILs × Area of one TIL (in pixels).
- **Stroma Area** = Area of tumor-associated stroma (in pixels).

RESULTS

Detection of Lymphocytes and Plasma Cells