Panoptic segmentation of Unclei and tissue in advanced MelanomA:

Mervat Khaled Emam, ID:********

In this project, I have used a pretrained U-net from segmentation model's module with EfficientNetB0 as a backbone “Encoder”, to solve task 1 in Track 1, Semantic tissue segmentation. The performance was evaluated by Dice Coefficient (DSC), Dice Loss, IOU (Intersection Over Union) score.

EDA

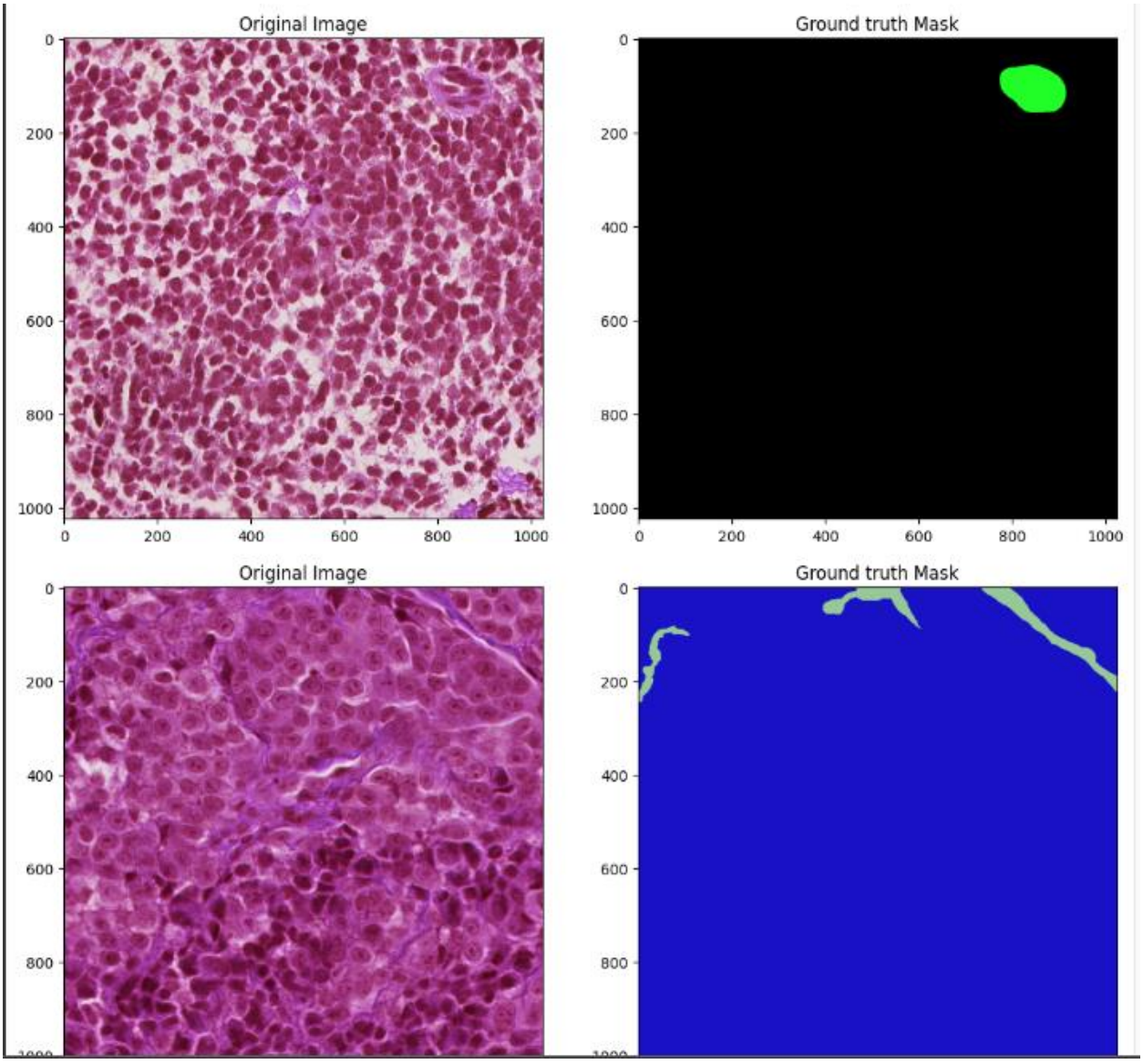
I implemented the EDA steps, to provide insights about both the tissue images and its masks. And the findings as follows:

- We have variation in image quality and contrast/intensities due to manual preparation of histological slides. which leads to high color variation, some areas in the image are blurred, tissue artifacts, and stain variability.
- Based on masks images, we have imbalanced classes that might affect model training, tissue blood vessel and tissue storma are more dominant classes.
- There is an overlapping between tissue segments boundaries.
- There are 61 unlabeled masks, and we excluded them and their images from the data.
- From image statistics for each color, we can observe that there is a high Mean in each color channel, indicates brighter images, and Low Standard Deviation indicates Low contrast and variation in tissue Images. Red and blue color channels (Left-Skewed): which indicates that the majority of the pixel values in the red and blue channels are low (darker). and this suggests that there is a relative scarcity of intense red and blue colors in the images. Green Channel (Right Skewed): indicates that the majority of the pixel values in the green channel are high (brighter), which means

the green is more dominant in images. Recommendation: We should do the same analysis on the different tissue segments/classes individually to find the areas that have color dominance, that may help in data augmentation strategies to balance the data.

Graphs:

Samples of tissues and masks.



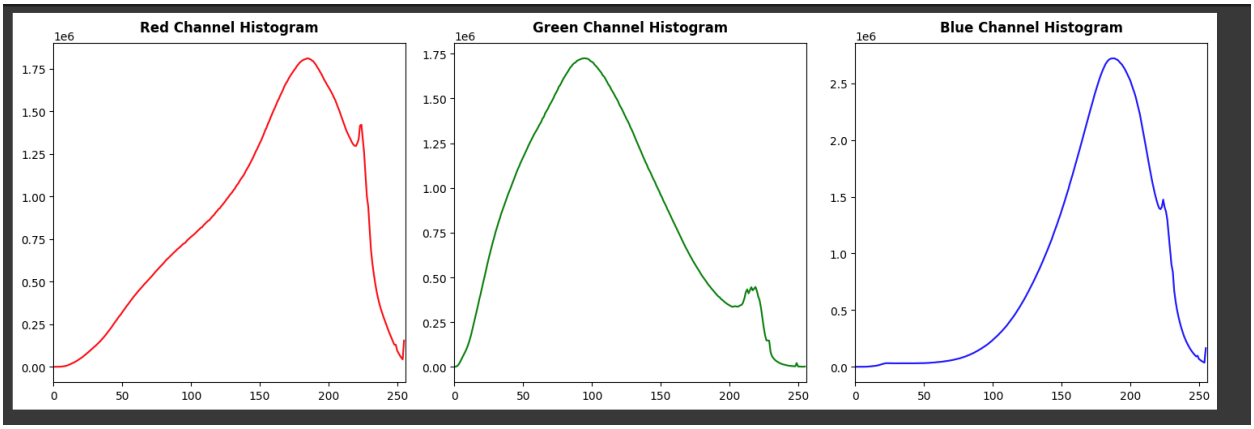
Classes Frequency in masks after color mapping for each class:
0: refers to tissue blood vessel, 1: storma, 2: tumor 3: necrosis

```
color_mapping={
            0:(0, 255, 0),          # tissue_blood_vessel
            1: (150, 200, 150),      # tissue_stroma
            2:(200, 0, 0),          # tissue_tumor
            3:(51, 0, 51),          # tissue_necrosis
            4:(99, 145, 164)        # tissue_epidermis
        }
```

```
flat_labels=labels.flatten()
unique_labels,label_counts=np.unique(flat_labels,return_counts=True)
total_frequencies=np.sum(label_counts)

print(f"Total Frequencies: {total_frequencies}")
print(f"Frequencies per class: {dict(zip(unique_labels,label_counts))}")
```

```
Total Frequencies: 10551296
Frequencies per class: {0: 8290403, 1: 2251190, 2: 148, 3: 9555}
```

Histogram calculation for each color channel.



Pre-processing

- Due to GPU constraints/limitations we reduced the image size to 256 * 256.
- We applied a simple normalization technique: divide the image by 255.

- We converted the tissue color images to grayscale images, to reduce complexity and high dimension of color images.[1]

Image Enhancement:

To improve the contrast in images, which can make edges and boundaries more distinct, we applied 4 preprocessing techniques:

- **CLAHE (contrast limited adaptive histogram equalization):**
  CLAHE was initially used to improve low contrast medical images. CLAHE improves the contrast of images by redistributing the pixel intensity values. **Noise Reduction:** And it differs from ordinary AHE (Adaptive Histogram Equalization) in that it limits contrast, to address the issue of noise amplification, the CLAHE implemented a clipping limit before computing distribution function. The clahe limits the amplification by clipping the histogram at a predefined value (CDF).

```
def clahe(image):
  image=np.array(image)
  gray_image=cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

  clahe=cv2.createCLAHE(clipLimit=2.0,tileGridSize=(8,8))
  clahe_image=clahe.apply(gray_image)
  return clahe_image
```

  CLAHE as a pre-processing step before segmentation, it applied with two parameters: clipLimit = 2.0, and tileGridsize =(8,8).
  We used a medium tile grid size because the smaller ones like (4,4), they allow for very fine local contrast enhancement but might miss small local details. And we avoided large tiles because they provide a smoother overall enhancement but might miss small local details.
  We used a moderate clip limit value of 2.0, because the smaller one prevented noise amplification but resulting in subtler changes to the image., and the larger one can highlight more details but also increases noise and artifacts.

- **Contrast Starching**
  Contrast searching is similar in idea to histogram equalization except that the pixel intensities are rescaled using the pixel values instead of probabilities and cdf.

---

[1]Indra Kanta Maitra, Jashojit Mukherjee : **Grayscale Conversion of Histopathological Slide Images as a Preprocessing Step for Image Segmentation**, Article in International Journal of Software Engineering and Its Applications · January 2016 .

Contrast stretching is used to increase the pixel value range by rescaling the pixel values in the input image.

The contrast stretching transformation, t(i,j) is given by the following equation:

$$t(i,j) = 255 * \frac{I(i,j) - a}{b - a}$$

Where I(i,j), a, and b are the pixel intensity at (i,j), the minimum pixel value and the maximum pixel value in the input image respectively.

```
def contrast_stretchingTrans(image):
  image=np.array(image)
  gray_image=cv2.cvtColor(image,cv2.COLOR_RGB2GRAY)
  b = gray_image.max()
  a = gray_image.min()
# converting im1 to float
  c = gray_image.astype(float)
  # contrast stretching transformation
  im2 = 255*(c-a)/(b-a)
  # im2 is converted from an ndarray to an image
  # im3 = scipy.misc.toimage(im2)
  # saving im3 as contrast_output.png in
  # Figures folder
  return im2
```

- **Power Law transformation**

  Power law transformation, also known as gamma-correction, is used to enhance the quality of the image. The power transformation at (i,j) is given by

$$t(i,j) = k\, I(i,j)^{\gamma}$$

Where K and gamma are positive constant, and I is the intensity value of the pixel in the input image at (i,j). In most cases K =1. When gamma < 1, a narrow range of dark or low intensity pixel values in the input image get mapped to a wide range of intensities in the output image, while a wide range of bright or high intensity pixel values in the input image

det mapped to a narrow range of high intensities in the output image. The effect from values of gamma > 1 is the opposite that of values gamma < 1.[2]

```
[ ]  #Power Law transformation
     import cv2
     def PowerLawTrans(image):
       image=np.array(image)

       image1 = cv2.cvtColor(image,cv2.COLOR_RGB2GRAY)

       # gamma is initialized
       gamma = .5
       # b is converted to type float
       image1 = image1.astype(float)
       # maximum value in b1 is determined
       image3 = np.max(image1)
       # b1 is normalized
       image2 = image1/image3
       # gamma-correction exponent is computed
       image3 = np.log(image2)*gamma
       # gamma-correction is performed
       c = np.exp(image3)*255.0
       # c is converted to type int
       c1 = c.astype('uint8')
       # c1 is converted from ndarray to image
       # d = scipy.misc.toimage(c1)
       # displaying the image
       return c1
```

- **CIELAP method to convert a color image to gray scale**

  This method is applied to reduce complexity and dimension to segment color images, particularly in the context of histology images, this offers several benefits:

  1- **Better representation of lightness**: The **L** channel in CIELAB color space, which ranges from 0 (black) to 100 (white), represents the lightness of the image, capturaing more detail in terms of shadows and highlights.

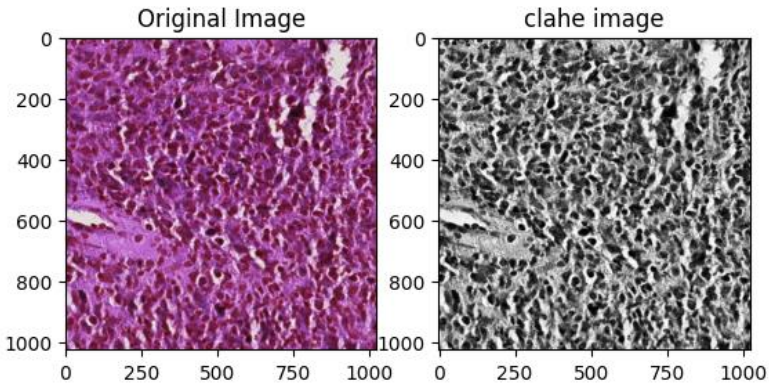
  2- Enhanced contrast: Histology images often contain regions with varying contrast due to different staining techniques. Using the L * channel ensures that these contrasts are preserved, making it easier to identify and segment small stains.

[2] Ravishanker chityala, Sridevi Pudipedddi: **Image processing and acquisitionn using python,** CRC Press Tylor & Francis Group, UK, 2014, p 93-96.

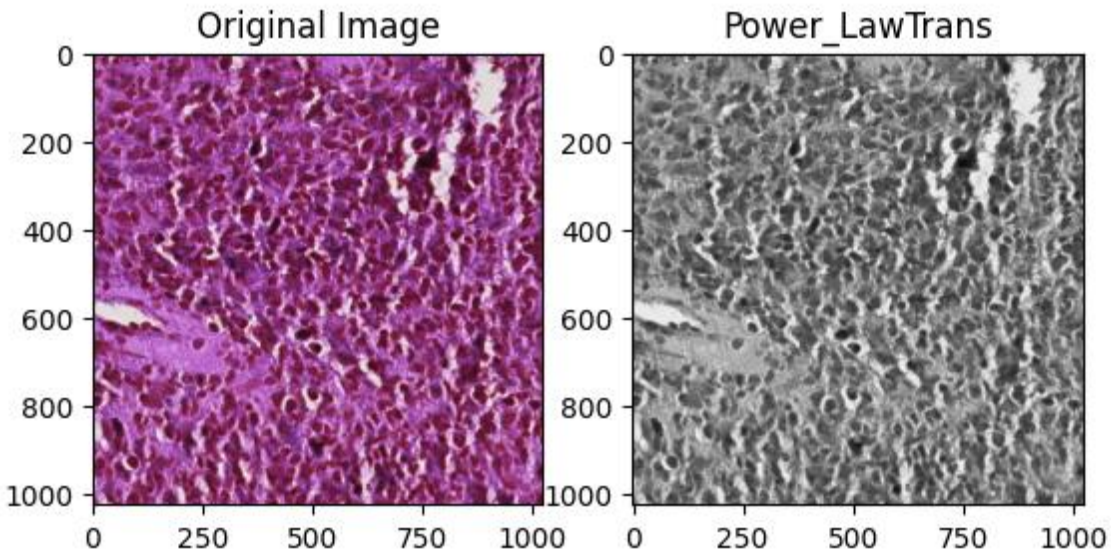
3- **Noise Reduction:** By focusing on the lightness component, we can often reduce color noise that might be present in the RGB channels.

```
#Final   technique to enhance image contrast and details we will combine different filters togeth
import cv2
# # from skimage import io, color, filters, morphology
def cielab(image):
  image=np.array(image)
  #first contrast streaching
  cielab_image=cv2.cvtColor(image,cv2.COLOR_RGB2Lab)

  L_channel,_,_=cv2.split(cielab_image)
  L_channel_normalized=cv2.normalize(L_channel,None,alpha=0,beta=255,norm_type=cv2.NORM_MINMAX)


  return L_channel_normalized
```

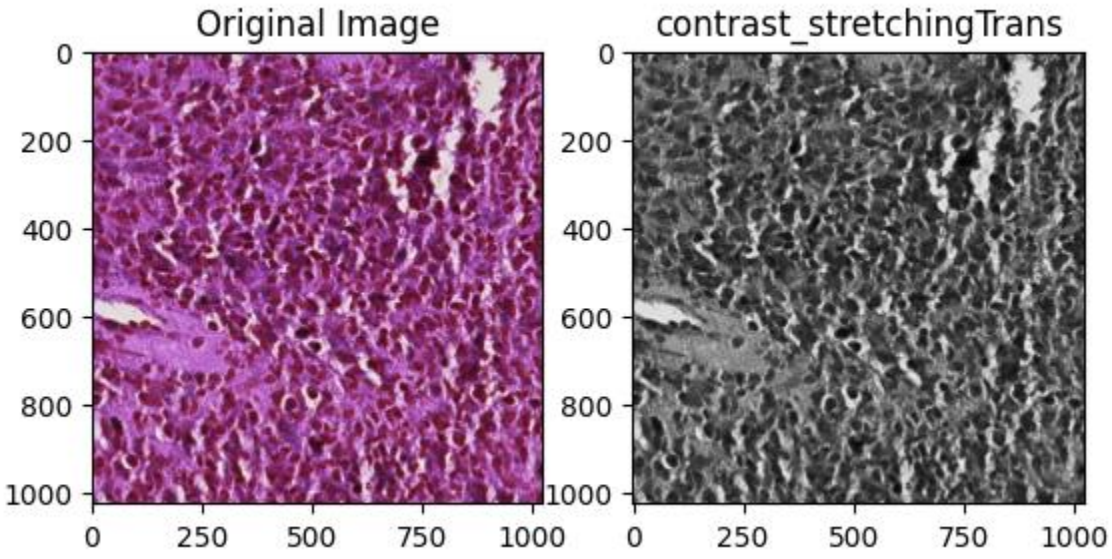
Images after CLAHE processing:



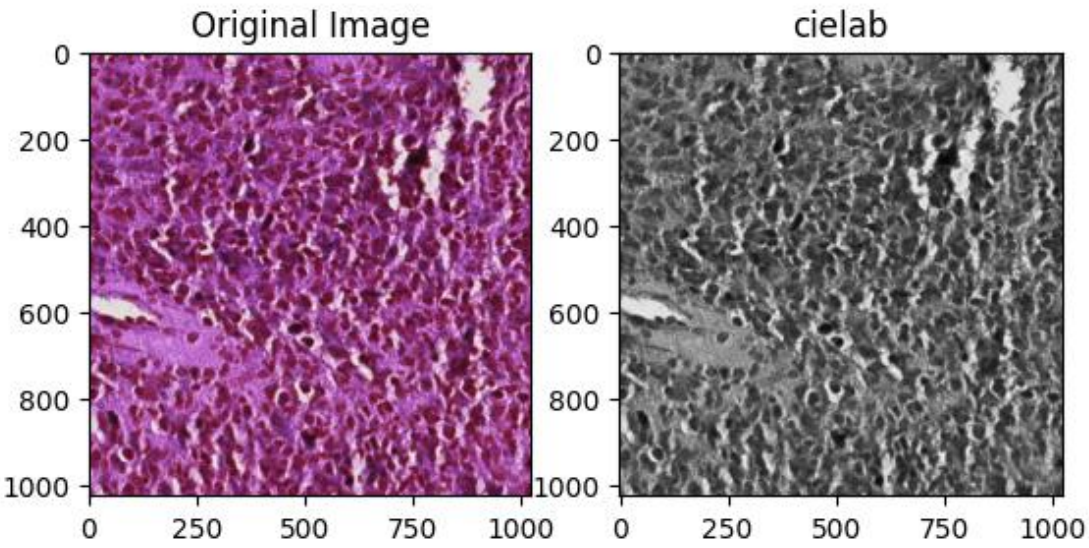
Images after Power Law transformation:



Images after contrast starching



Images after CIELAB



Data Augmentation:

Before data augmentation the shape of the images was (161, 256 ,256 ,1), and the shape of the masks was (161,256,256,4).
After we combined the preprocessed images and masks with augmented images and masks, the shape of combined images was (483, 256, 256, 1) and combined masks with shape (483, 256, 256, 4).

We have applied two data augmentation methods: **Elastic transformation**, and **random brightness**. The **Elastic transformation** was applied on the images and masks**,** whereas **random brightness** was applied on the images only**.**

- **Elastic Transformation: t**his involves applying random, smooth deformation to an image. It simulates natural variations and can help models generalize better.
- **Random Brightness:** Histology images can vary in brightness due to differences in staining, lighting condition, and imaging equipment. Random brightness augmentation ensures that the model is robust to these variations, improving its performance in real-world applications.

Model

I have used a pretrained U-net model from segmentation models module with EfficientNetB0 as a backbone “Encoder”. Why EfficientNetB0?

- **The EfficientNetB0 backbone** is a convolutional nerual network architecture designed for image classification. It’s part of the EfficientNet family, which focuses on scaling up CNN in a more structured way to achieve better performance. When

used in segmentation models, EfficientNetB0 serves as the encoder part of the network, extracting features from the input images.

- **Squeeze-and- Excitation Blocks: These** blocks are components of EfficientNetB0, they adaptively recalibrate channel- wise feature responses by explicitly modeling interdependencies between channels. In case we use only one-color channel 'gray', that is still efficient; because it enhances the network's ability to focus on important features, resulting in better boundary detection, more accurate segmentation maps, even with a single channel.
  We are incorporating SE blocks with a reduction ratio of 16, so the model becomes more sensitive to the most critical features.

```
def se_block(input_tensor, reduction_ratio=16):
    filters = input_tensor.shape[-1]
    se_shape = (1, 1, filters)

    se = layers.GlobalAveragePooling2D()(input_tensor)
    se = layers.Reshape(se_shape)(se)
    se = layers.Dense(filters // reduction_ratio, activation='relu', kernel_initializer='he_normal', use_bias=False)(se
    se = layers.Dense(filters, activation='softmax', kernel_initializer='he_normal', use_bias=False)(se)
    x = layers.multiply([input_tensor, se])
    return x

# Define the SE U-Net model
def build_se_unet(input_shape):

    backbone = sm.get_preprocessing('efficientnetb0') #, input_shape=input_shape, include_top=False
    inputs=Input(shape=input_shape)
    base_model=sm.Unet(backbone_name='efficientnetb0', encoder_weights=None, classes=4, activation='softmax',input_shap
    # Add SE blocks to each stage of the backbone
    for layer in base_model.layers:


        if 'conv' in layer.name:

            layer_output = layer.output
            se_output = se_block(layer_output)
            backbone = tf.keras.Model(inputs=base_model.input, outputs=se_output)

    # Build U-Net model with SE backbone
    base_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate= 0.0001),loss =dice_loss ,metrics=[IOU_score,di

    return base_model

# Input shape
input_shape = (256, 256, 1)
# Build the model
model = build_se_unet(input_shape)



# Print the model summary
model.summary()
```

decoder_stage4b_conv (Conv2D)	(None, 256, 256, 16)	2,304
decoder_stage4b_bn (BatchNormalization)	(None, 256, 256, 16)	64
decoder_stage4b_relu (Activation)	(None, 256, 256, 16)	0
final_conv (Conv2D)	(None, 256, 256, 4)	580
softmax (Activation)	(None, 256, 256, 4)	0

Total params: 10,115,360 (38.59 MB)

Trainable params: 10,071,360 (38.42 MB)

Non-trainable params: 44,000 (171.88 KB)

Train and test data:

We divided the data to %80 training and %20 testing, the training images: 386, and the test images: 97.

Learning Rate: 0.0001 Adam optimizer

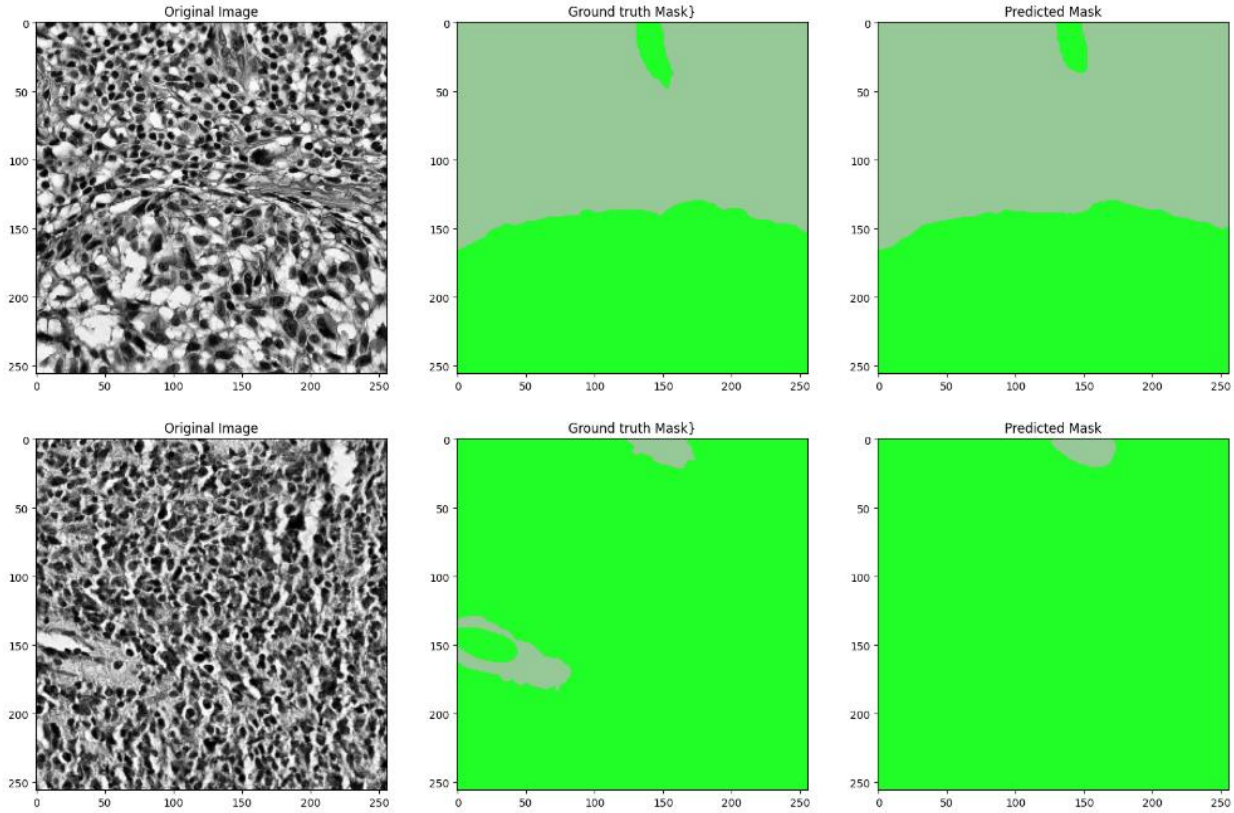
Batch Size: 8 batch size due to the size of the data, which is small.

Epochs: 100 epochs with early stopping constrain if the validation loss, which is dice loss, doesn't decreased after 20 epochs, and why 20 epochs: because the model takes a lot of iterations to converge and capture the patterns.

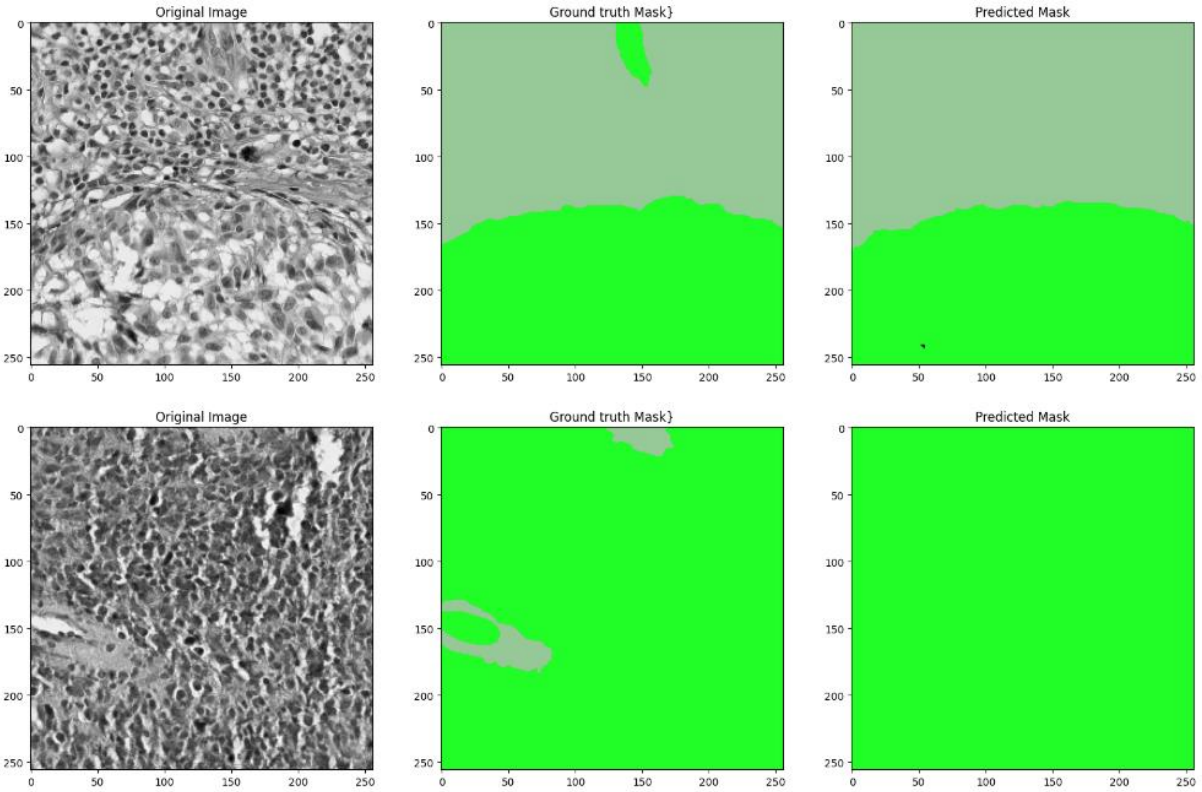
Results:

After we applied the model on the four preprocessing techniques: Clahe, Power law, Contrast stretching, and CIElab, the best performance was achieved on the Clahe preprocessing technique, the model could predicate the small stains and was nearly of the stain shapes in the ground truth masks.

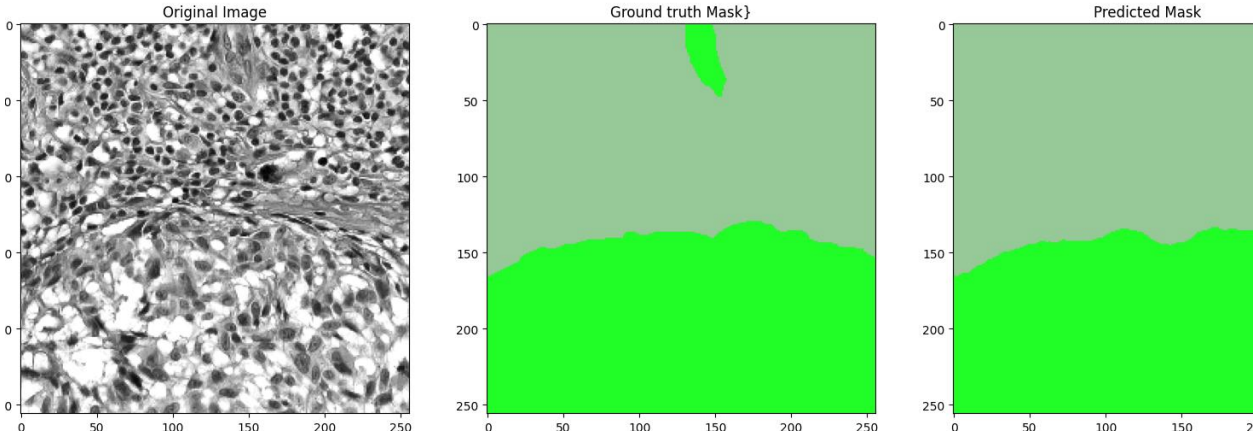
1- The model performance on the **Clahe** processing:

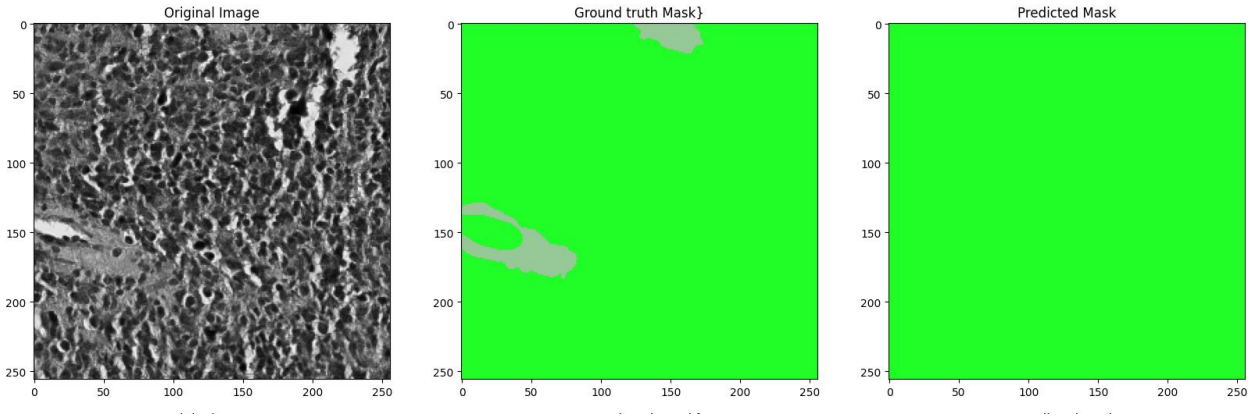


2- The model performance on the **Power law processing**:

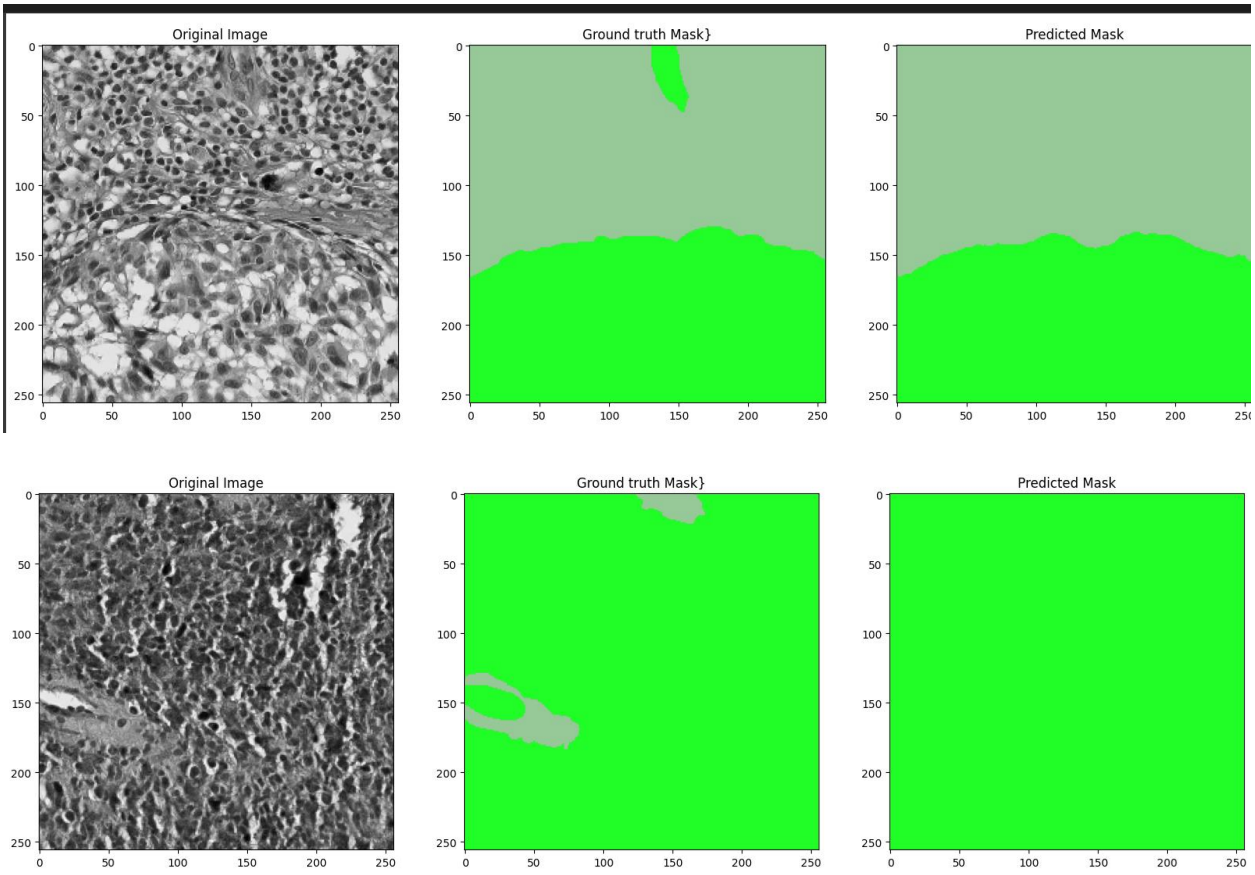


3- The model performance on the **Contrast stretching** processing

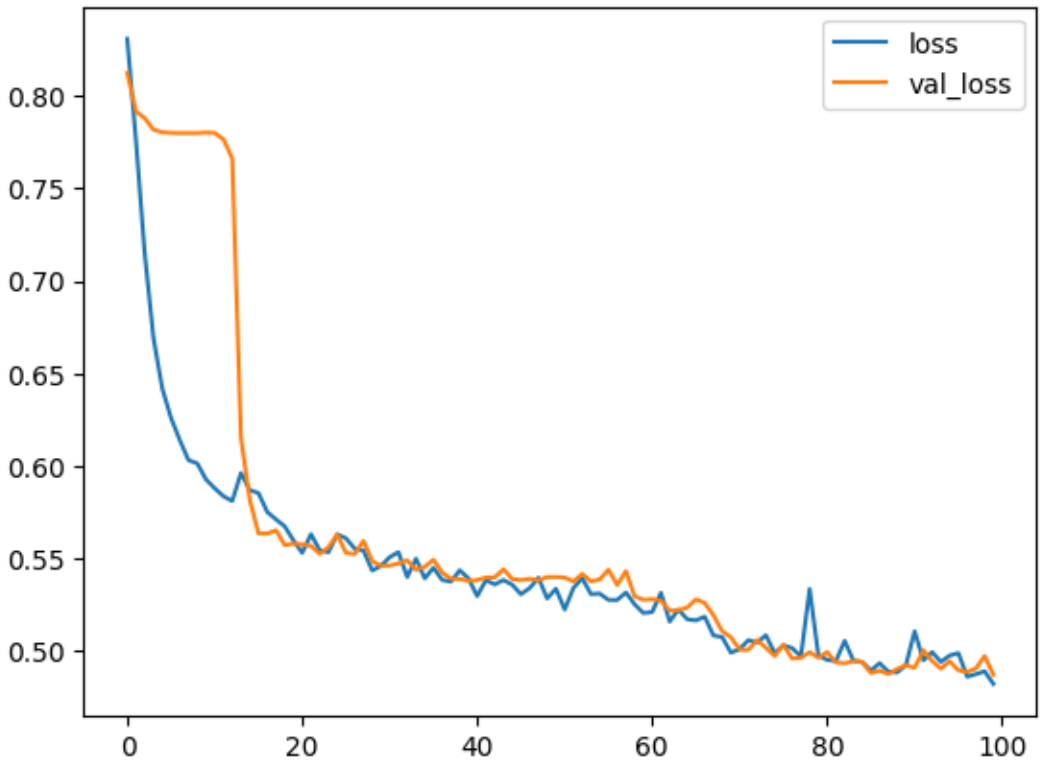


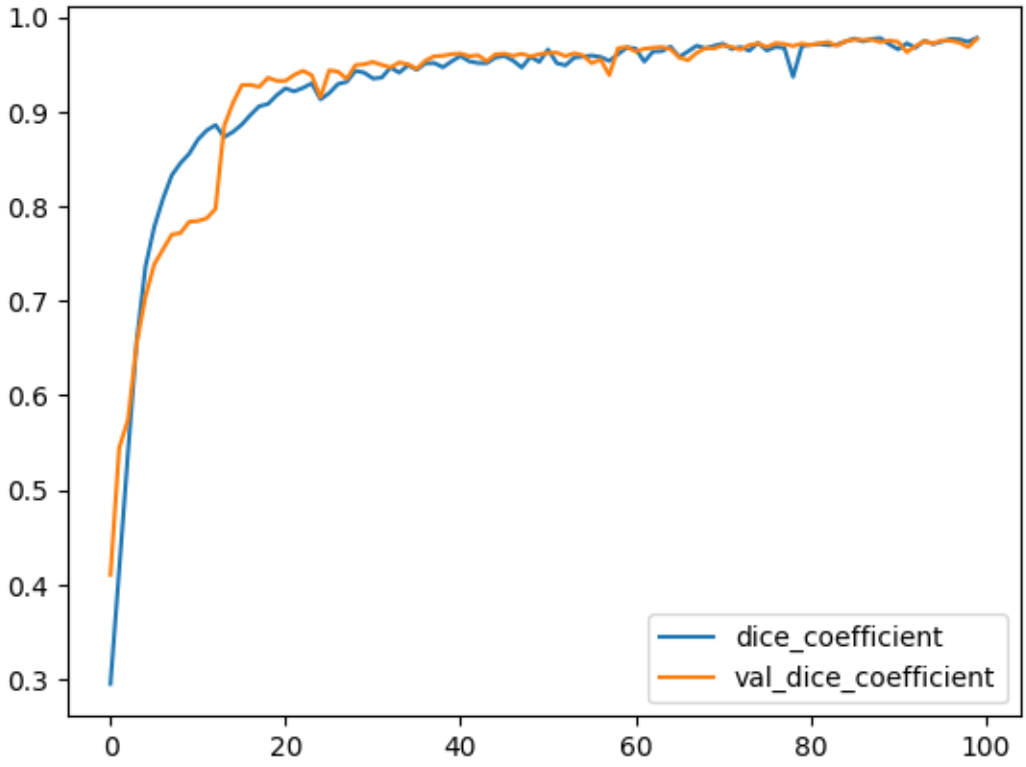


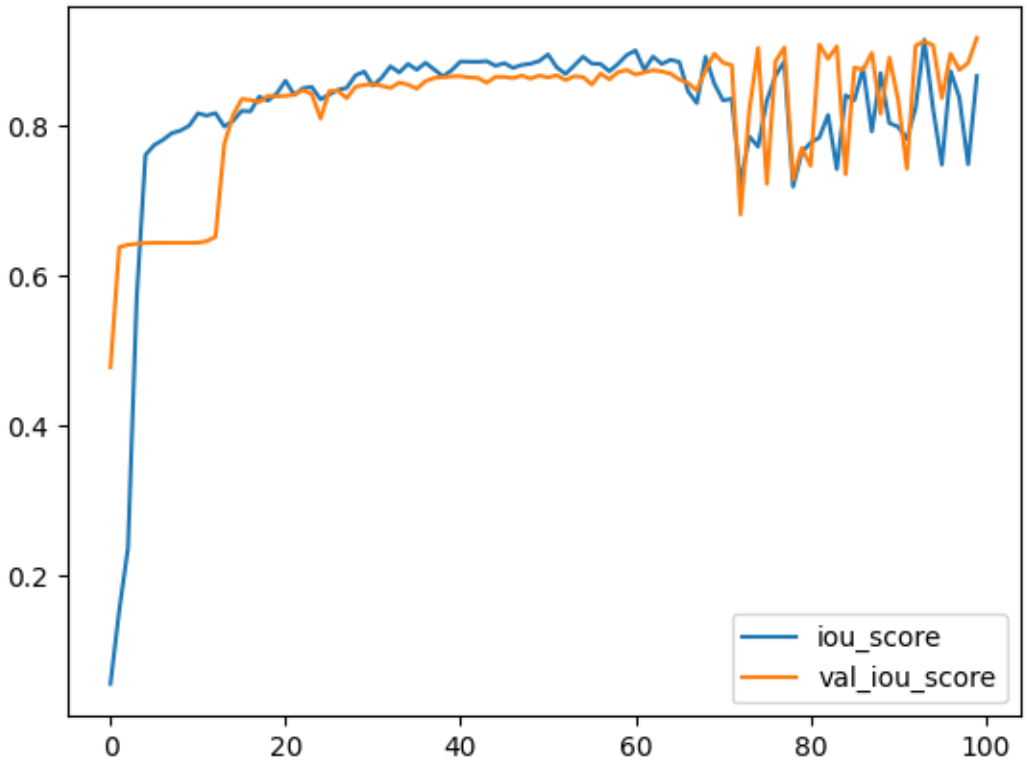
4- The model performance on the **CIElab** processing:



Evaluation Graphs of the best result: (Clahe processing):


loss
val_loss
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0
20
40
60
80
100


dice_coefficient
val_dice_coefficient
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
20
40
60
80
100



Further steps to improve the model prediction:

- **Error Analysis for each class**
- **Using any clustering model to group the similar tissues characteristics to apply specific preprocessing on each one.**

Resources:

- Indra Kanta Maitra, Jashojit Mukherjee : **Grayscale Conversion of Histopathological Slide Images as a Preprocessing Step for Image Segmentation, Article in International Journal of Software Engineering and Its Applications** · January 2016 **.**
- Ravishanker chityala, Sridevi Pudipedddi: **Image processing and acquisitionn using python**, CRC Press Tylor & Francis Group, UK, 2014.