Module_2:

Team Members:

Max Calcoen, Jack O'Hearn

Project Title: Imaging-based IPF severity prediction for biopsy device development

Project Goal:

The objective of this project is to develop a computational pipeline to quantify and predict the extent of fibrosis in a mouse model of IPF to guide development of a better lung biopsy device.

Disease Background:

Fill in information and please note that this module is truncated and only has 5 bullets (instead of the 11 that you did for Module #1).

- Prevalence & incidence (source)
  - north america / europe: ~2.8 to 9.3 per 100,000 person-years (varies by definition)
  - usa, narrow definition: ~6.8 per 100,000; broader: up to ~16.3–17.4 per 100,000
  - prevalence in u.s. estimated ~10 to 60 cases / 100,000 (rare disease threshold)
- Risk factors (genetic, lifestyle) (source)
  - older age / aging (disease primarily presents in middle to older adults)
  - male sex bias
  - cigarette smoking / smoking history (strongest environmental risk)
  - genetic predispositions: mutations in telomerase genes, surfactant genes, shorter telomeres, MUC5B promoter variant (minor allele)
  - microbiome / microbial burden in lung (higher bacterial load in BAL, certain genera like Streptococcus, Staphylococcus)
  - comorbidities / cofactors: gastroesophageal reflux disease (GERD), environmental exposures (metal dusts, silica, wood dust), viral injury, lung injury agents
- Symptoms (source)
  - gradual onset progressive (shortness of breath) on exertion
  - dry nonproductive chronic cough

    - bibasilar “velcro” crackles on auscultation (fine inspiratory crackles)
    - reduced exercise tolerance, fatigue, weight loss (constitutional)
    - pulmonary function test: restrictive pattern, decreased forced vital capacity (FVC), decreased DLCO (diffusing capacity)
- Standard of care treatment(s) (source)
    - antifibrotic therapy: Pirfenidone (slows decline in lung function, reduces fibrosis mediators)
    - antifibrotic therapy: Nintedanib (tyrosine kinase inhibitor, slows FVC decline)
    - supportive therapies: supplemental oxygen, pulmonary rehabilitation, symptom control (cough management)
    - management of comorbidities (GERD, pulmonary hypertension)
    - lung transplantation
    - clinical trials / emerging therapies (targeting profibrotic pathways, biomarkers)
- Biological mechanisms (anatomy, organ physiology, cell & molecular physiology) (source)
    - impaired epithelial regeneration / aberrant repair, with persistent activation of epithelial cells and failed re-epithelialization
    - senescence, mitochondrial dysfunction, oxidative stress in epithelial cells leading to pro-fibrotic signaling
    - release of profibrotic mediators (TGF-β, connective tissue growth factor, PDGF, fibronectin, integrins) from injured epithelium & mesenchymal cells
    - recruitment, activation and differentiation of fibroblasts → myofibroblasts → excessive ECM (extracellular matrix) deposition (collagen, fibronectin, proteoglycans)
    - mechanical stress feedback: matrix stiffening, mechanotransduction, latent TGF-ß activation via contraction of myofibroblasts
    - cross talk with immune cells / inflammation: dysregulated wound healing, low grade chronic inflammation, altered macrophage / fibrocyte responses
    - epigenetic alterations, noncoding RNAs, altered gene expression networks in fibrotic lung cells

Data-Set:

(Describe the data set(s) you will analyze. Cite the source(s) of the data. Describe how the data was collected -- What techniques were used? What units are the data measured in? Etc.)

- The dataset is composed of Unpublished data collected by the Peirce-Cottler Lab (Dept. of Biomedical Engineering) and Kim Lab (Division of Pulmonary and Critical Care) at the University of Virginia School of Medicine.
- The dataset is composed of 78 black and white images (.jpg), collected at various depths of a fibrotic mouse lung.

- White in the images symbolizes fibrotic lesion, and black symbolizes healthy lung tissue.
- The images come from a Bleomycin-induced Lung Injury Model, where an antibiotic primarily used as chemotherapy (but also causes lung fibrosis) is administered to a mouse.
- 3 weeks later, the mice were harvested.
- The mouse lung tissue is then fixed, mounted, and sliced, then fluorescent microscopy was performed.
- The mouse lungs were immunostained for 3 proteins of interest:
  - desmin (myofibroblasts)
  - smooth muscle alpha actin (large blood vessel smooth muscle cells)
  - CD-31 (endothelial cells)

Data Analyis:

(Describe how you analyzed the data. This is where you should intersperse your Python code so that anyone reading this can run your code to perform the analysis that you did, generate your figures, etc.)

First, we used edge detection and contour detection to find blood vessels / airways to remove from analysis- as there is no tissue in the lumen, it should not be considered in our analysis. We used manual_contours_subset.ipynb, which contained methods of dilation, erosion, and contour size-based filtering in order to generate masks of these regions. We stored the area that we generated. Then, we calculated the white pixel percentage in each image while considering the areas or not. Then, we fit 2 generalized linear models to predict white pixel percentage with the depth of the image as the dependent variable. We finally interpolated multiple points using the fitted model.

Additional analysis was conducted to analyze the difference between slobe and llobe in both white pixel percent and depth; however, MANOVA showed no significant correlation between white pixel percent and any other features (especially depth).

Below is the analysis. It is also available here.

data folder setup (for running code below)

```
/data
    /imaging
        MASK_Sk658 Llobe ch010017.jpg
        ...here contains all the images
    depths.csv
    Filenames and Depths (old).csv
    pct_white_pixels.csv
```

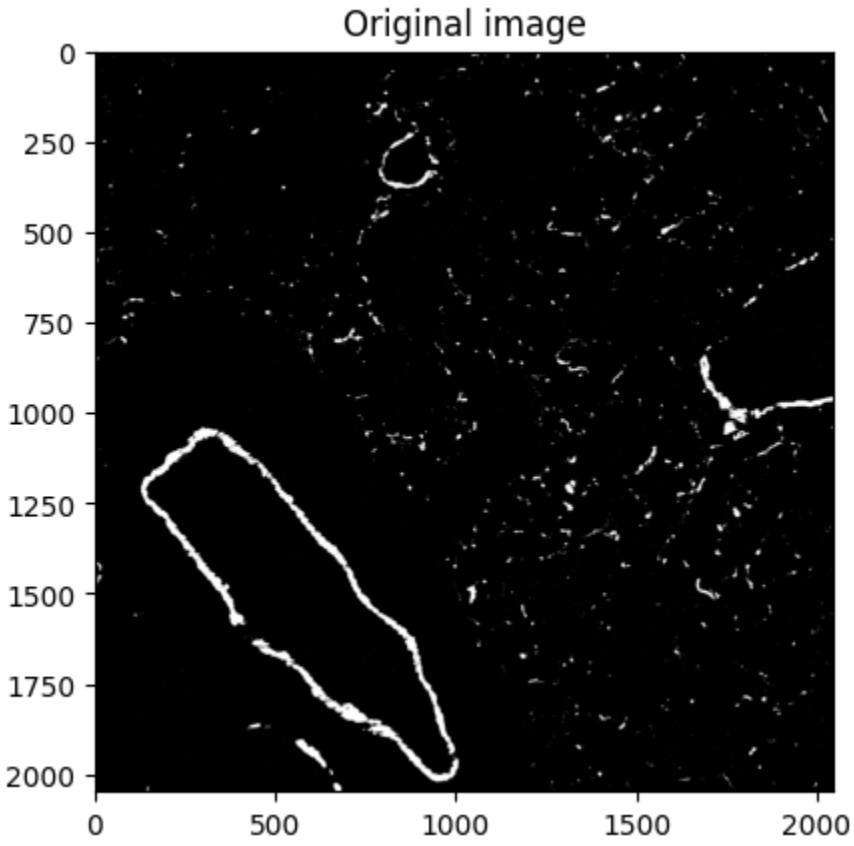
In [4]:

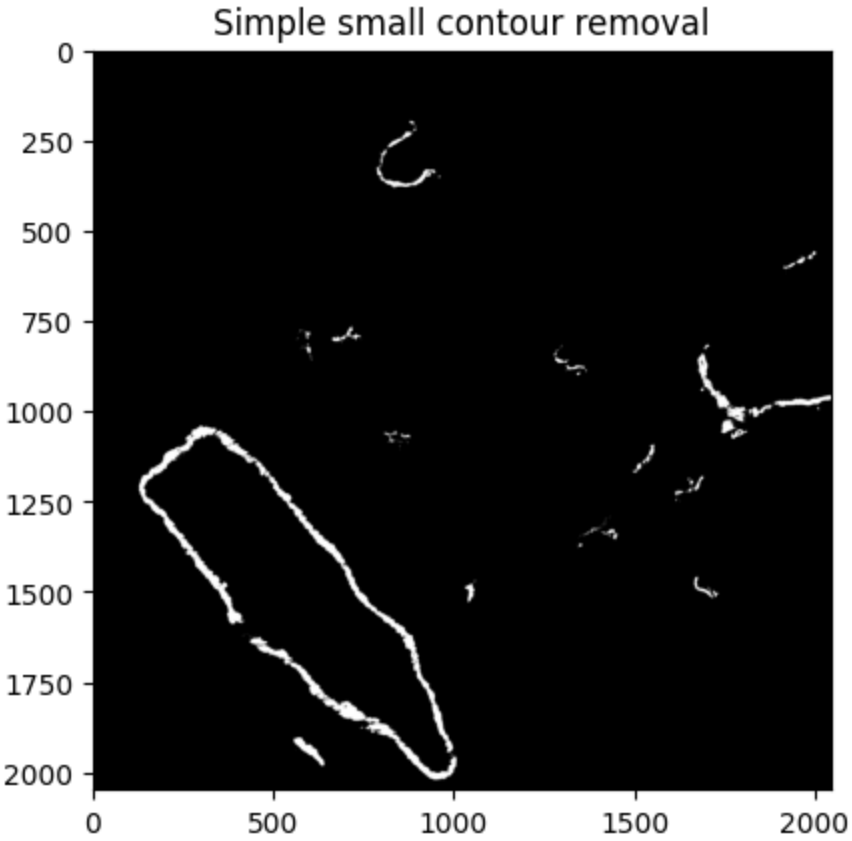
```
"""FILE: contour_detection.ipynb"""
# %%
import pandas as pd
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt

images = pd.DataFrame()

data_path = r"data"
imaging_path = r"imaging"
filenames = os.listdir(os.path.join(data_path, imaging_path))
depths = pd.read_csv(os.path.join(data_path, "depths.csv"))

# display(filenames)
for i in filenames:
    img = cv2.imread(os.path.join(data_path, imaging_path, i), 0)
    try:
        depth = depths[depths["Filenames"].str.lower() == i.lower()][
            "Depth from lung surface (in micrometers) where image was acquir
        ].values[0]
        # some files are named with SK658 and some with Sk658
    except IndexError:
        print(f"couldn't find depth for file {i}")
        continue
    images = pd.concat(
        [images, pd.DataFrame([{"filename": i, "image": img, "depth": depth}
        ignore_index=True,
    )

print(images.shape)

# %%
print(images["filename"])

# %%
img = (
    images[images["filename"] == "MASK_Sk658 Llobe ch010034.jpg"]["image"]
    .values[0]
    .copy()
)

# convert to rgb instead of grayscale
plt.imshow(img, cmap="gray")
plt.title("Original image")
plt.show()

# %%
# remove small contours

img_contour_simple = img.copy()

contours, _ = cv2.findContours(
    img_contour_simple, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE
)
```

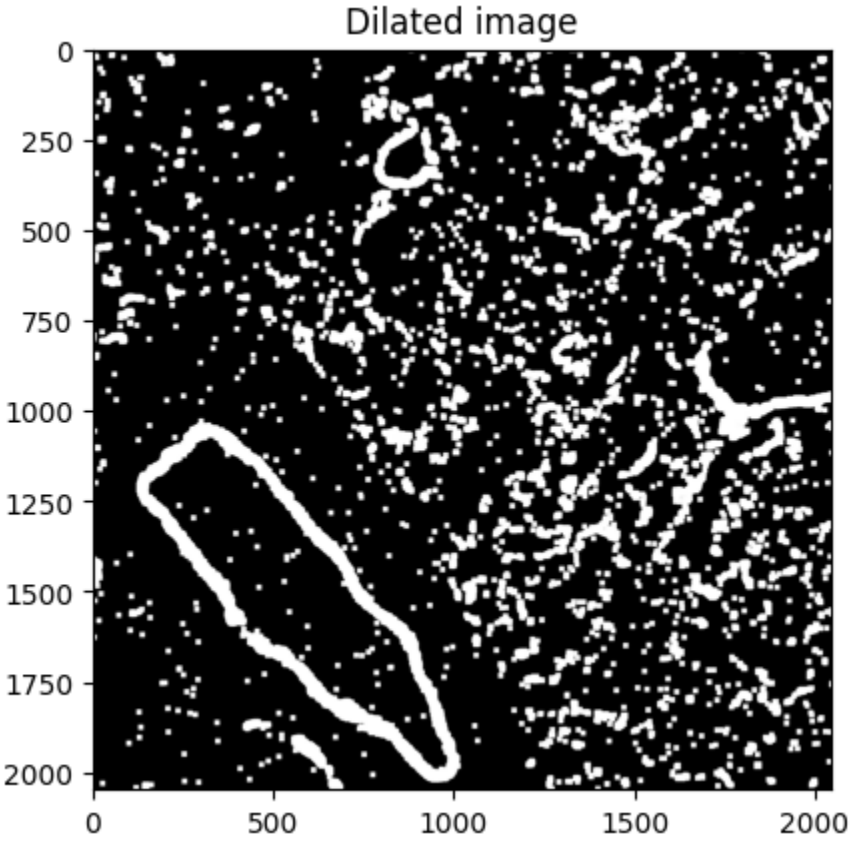
```
for cntr in contours:
    if cv2.contourArea(cntr) > 1000:
        continue
    convHull = cv2.convexHull(cntr)
    cv2.drawContours(
        img_contour_simple, [convHull], -1, (0, 0, 0), thickness=cv2.FILLED
    )
img_contour_simple = cv2.cvtColor(img_contour_simple, cv2.COLOR_GRAY2RGB)
plt.imshow(img_contour_simple)
plt.title("Simple small contour removal")
plt.show()

# %%
img_dilate = img.copy()
img_dilate = cv2.dilate(img_dilate, np.ones((15, 15), np.uint8), iterations=

plt.imshow(img_dilate, cmap="gray")
plt.title("Dilated image")
plt.show()

# %%
# look at contours on dilated image
img_dilated_contour = img_dilate.copy()
contours, _ = cv2.findContours(
    img_dilated_contour, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE
)

for cntr in contours:
    if cv2.contourArea(cntr) > 10000:
        continue
    convHull = cv2.convexHull(cntr)
    cv2.drawContours(
        img_dilated_contour, [convHull], -1, (0, 0, 0), thickness=cv2.FILLED
    )
img_dilated_contour = cv2.cvtColor(img_dilated_contour, cv2.COLOR_GRAY2RGB)
plt.imshow(img_dilated_contour)
plt.title("Dilated contours")
plt.show()

# %%
# combine
img_combine = img_contour_simple.copy()
img_combine = cv2.dilate(img_combine, np.ones((15, 15), np.uint8), iteration
# Convert to grayscale before thresholding
img_combine_gray = cv2.cvtColor(img_combine, cv2.COLOR_RGB2GRAY)
_, img_combine_thresh = cv2.threshold(img_combine_gray, 128, 255, cv2.THRESH
contours, _ = cv2.findContours(img_combine_thresh, cv2.RETR_EXTERNAL, cv2.CH

for cntr in contours:
    if cv2.contourArea(cntr) > 50000:
        continue
    convHull = cv2.convexHull(cntr)
    cv2.drawContours(img_combine_thresh, [convHull], -1, (0, 0, 0), thicknes
img_combine = cv2.cvtColor(img_combine_thresh, cv2.COLOR_GRAY2RGB)
plt.imshow(img_combine)
```

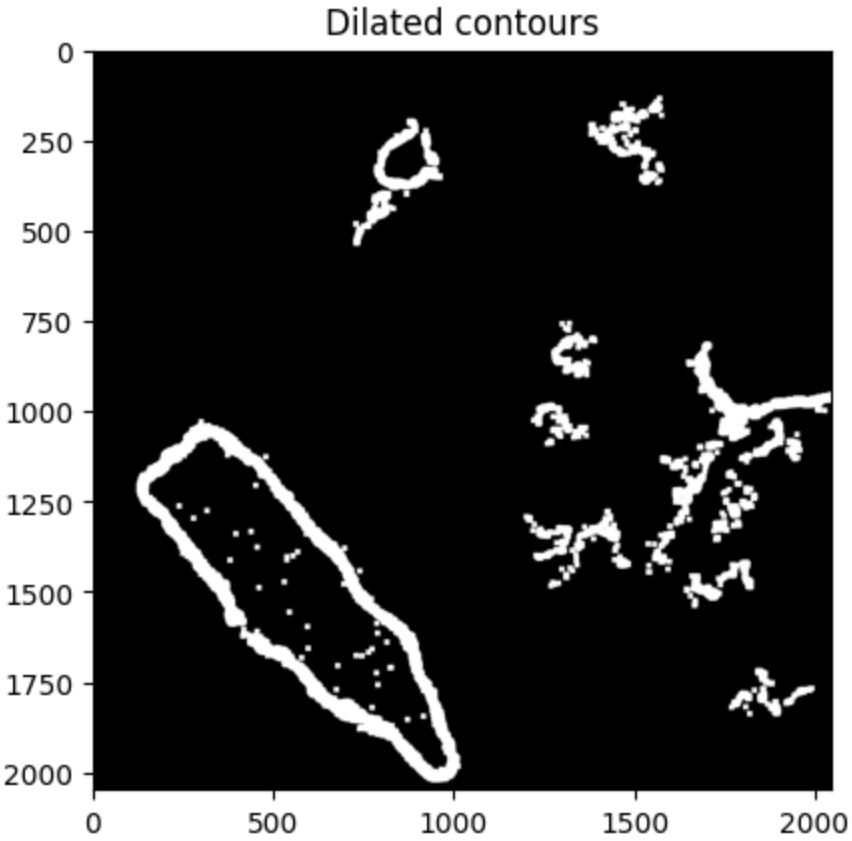
```
plt.title("Combined: removed small contours then dilated and\n removed small
plt.show()
```

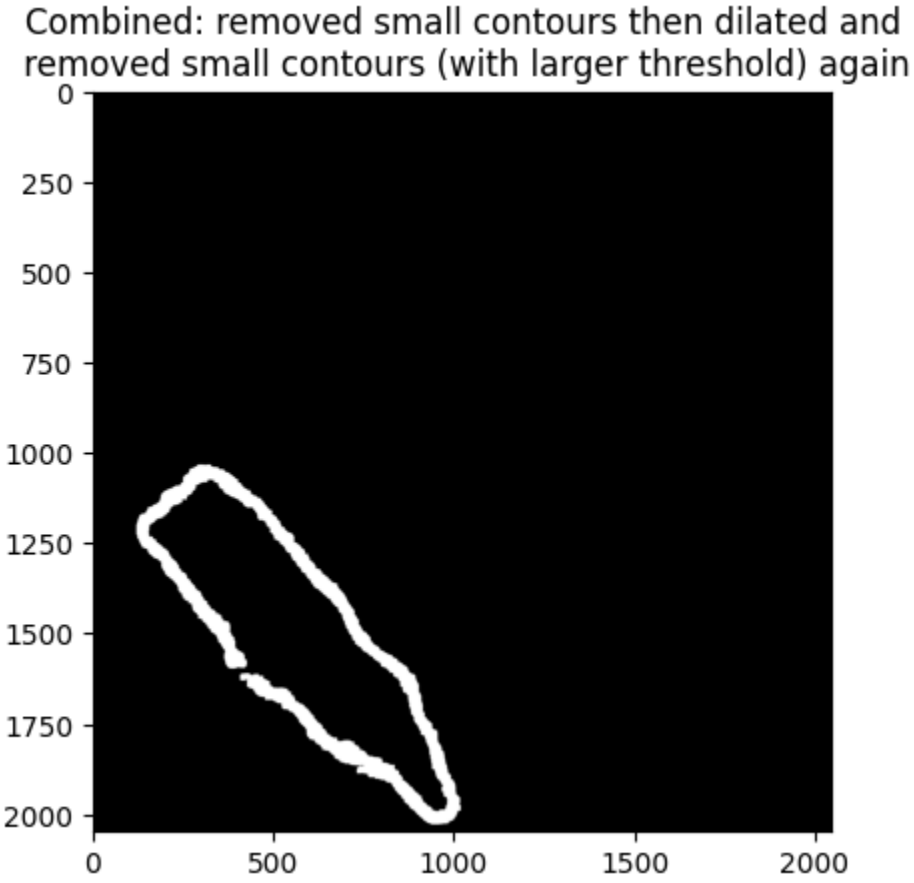
```
couldn't find depth for file MASK_Sk658 Llobe ch010053.jpg
(78, 3)
0     MASK_SK658 Slobe ch010129.jpg
1     MASK_SK658 Slobe ch010115.jpg
2     MASK_SK658 Slobe ch010114.jpg
3     MASK_SK658 Slobe ch010060.jpg
4     MASK_SK658 Slobe ch010048.jpg
                  ...
73    MASK_SK658 Slobe ch010118.jpg
74    MASK_SK658 Slobe ch010130.jpg
75    MASK_SK658 Slobe ch010078.jpg
76    MASK_SK658 Slobe ch010087.jpg
77    MASK_SK658 Slobe ch010093.jpg
Name: filename, Length: 78, dtype: object
```




Simple small contour removal
0
250
500
750
1000
1250
1500
1750
2000
0
500
1000
1500
2000


Dilated image
0
250
500
750
1000
1250
1500
1750
2000
0
500
1000
1500
2000



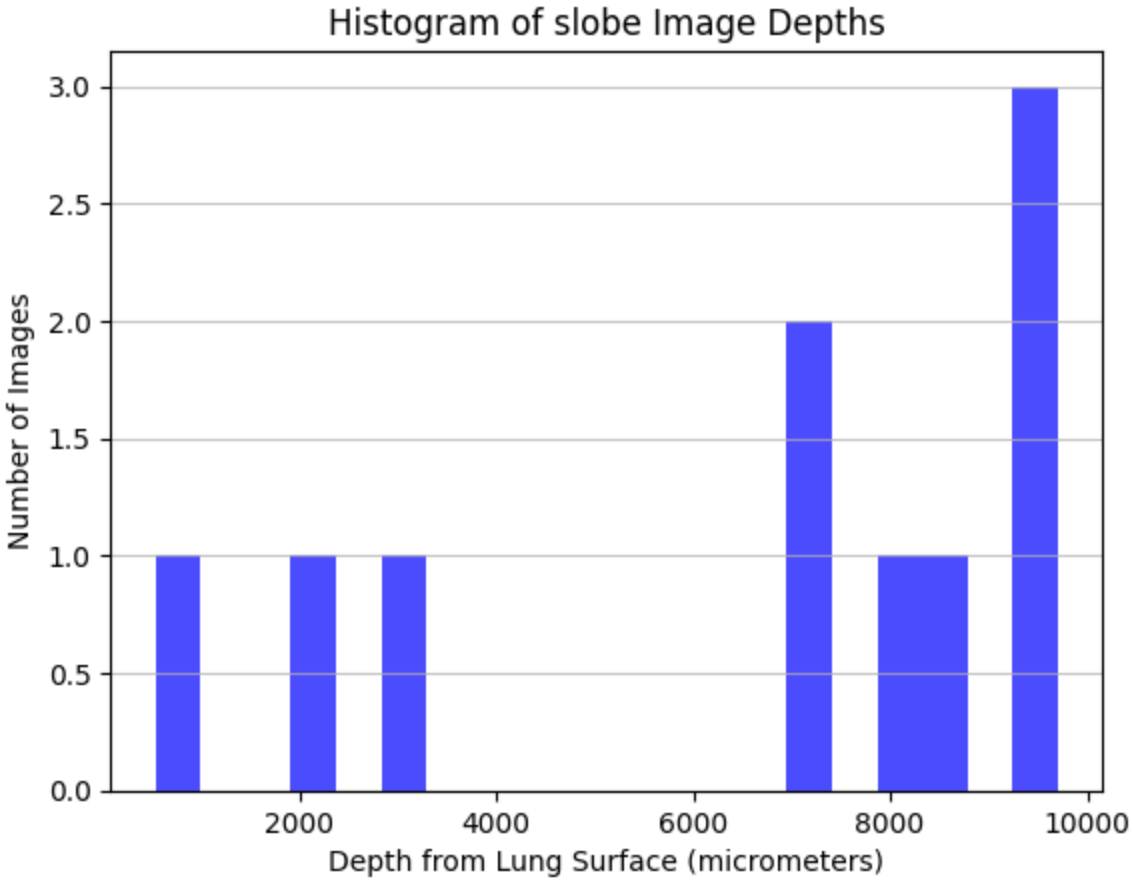


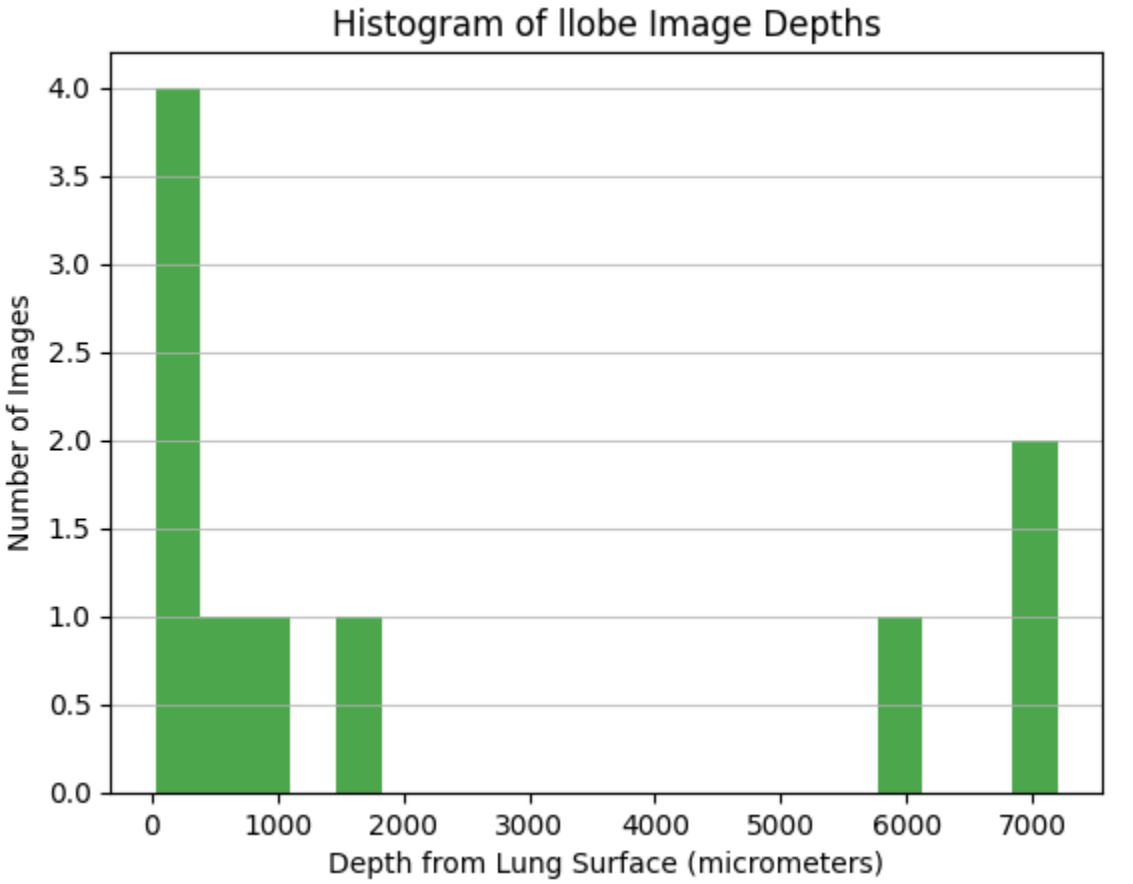
In [12]:
```
"""FILE: get_random_subset.ipynb"""
# %%
```

```python
import random
import pandas as pd
import os
import cv2
import matplotlib.pyplot as plt

data_path = r"data"
imaging_path = os.path.join(data_path, "imaging")
subset_path = os.path.join(data_path, "imaging_subset")

filenames = os.listdir(imaging_path)

# %%
# get random subset with even amount of slobe and llobe
slobe_files = [f for f in filenames if "slobe" in f.lower()]
llobe_files = [f for f in filenames if "llobe" in f.lower()]
random_slobe = random.sample(slobe_files, 10)
random_llobe = random.sample(llobe_files, 10)
random_subset = random_slobe + random_llobe
# copy images over to imaging_subset folder

if input("copy images to imaging_subset folder? (y/n): ") == "y":
    os.makedirs(subset_path, exist_ok=True)
    # make sure no existing files exist
    if os.listdir(subset_path):
        if (
            input(
                "imaging_subset folder is not empty. continue? this will rep
            )
            != "y"
        ):
            raise Exception("imaging_subset folder is not empty")
        # wipe the folder
        for f in os.listdir(subset_path):
            os.remove(os.path.join(subset_path, f))
    for f in random_subset:
        src = os.path.join(imaging_path, f)
        dst = os.path.join(subset_path, f)
        if not os.path.exists(dst):
            os.system(f'cp "{src}" "{dst}"')
            print(f"Copied {f} to imaging_subset folder.")
        else:
            print(f"{f} already exists in imaging_subset folder.")
else:
    print("images not copied.")

# %%
images_subset = pd.DataFrame()

data_path = r"data"
imaging_path = os.path.join(data_path, r"imaging_subset")
filenames = os.listdir(imaging_path)
depths = pd.read_csv(os.path.join(data_path, "depths.csv"))

for i in filenames:
    img = cv2.imread(os.path.join(imaging_path, i), 0)
```

```
    try:
        depth = depths[depths["Filenames"].str.lower() == i.lower()][
            "Depth from lung surface (in micrometers) where image was acquir
        ].values[0]
        # some files are named with SK658 and some with Sk658
    except IndexError:
        print(f"couldn't find depth for file {i}")
        continue
    images_subset = pd.concat(
        [images_subset, pd.DataFrame([{"filename": i, "image": img, "depth":
        ignore_index=True,
    )

print(images_subset.shape)

# %%
# plot the depths of all images of both groups

slobe_images = images_subset[
    images_subset["filename"].str.contains("slobe", case=False)
]
llobe_images = images_subset[
    images_subset["filename"].str.contains("llobe", case=False)
]

plt.hist(slobe_images["depth"], bins=20, color="blue", alpha=0.7)
plt.title("Histogram of slobe Image Depths")
plt.xlabel("Depth from Lung Surface (micrometers)")
plt.ylabel("Number of Images")
plt.grid(axis="y", alpha=0.75)
plt.show()
plt.close("all")


plt.hist(llobe_images["depth"], bins=20, color="green", alpha=0.7)
plt.title("Histogram of llobe Image Depths")
plt.xlabel("Depth from Lung Surface (micrometers)")
plt.ylabel("Number of Images")
plt.grid(axis="y", alpha=0.75)
plt.show()
plt.close("all")
```

```
images not copied.
(20, 3)
```





In [ ]:
```
"""FILE: manual_contours_subset.ipynb"""
```

```python
# %%
import pandas as pd
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
import time

images = pd.DataFrame()

data_path = r"data"
imaging_path = os.path.join(data_path, r"imaging_subset")
filenames = os.listdir(imaging_path)
depths = pd.read_csv(os.path.join(data_path, "depths.csv"))

for i in filenames:
    img = cv2.imread(os.path.join(imaging_path, i), 0)
    try:
        depth = depths[depths["Filenames"].str.lower() == i.lower()][
            "Depth from lung surface (in micrometers) where image was acquir
        ].values[0]
        # some files are named with SK658 and some with Sk658
    except IndexError:
        print(f"couldn't find depth for file {i}")
        continue
    images = pd.concat(
        [images, pd.DataFrame([{"filename": i, "image": img, "depth": depth}
        ignore_index=True,
    )
slobe_images = images[images["filename"].str.contains("slobe", case=False)]
llobe_images = images[images["filename"].str.contains("llobe", case=False)]
print(images.shape)
print(slobe_images.shape)
print(llobe_images.shape)

# %%
# read progress from csv
progress_path = os.path.join(data_path, "manual_contour_area.csv")
if os.path.exists(progress_path):
    progress = pd.read_csv(progress_path)
    labeled_files = progress["filename"].tolist()
    images = images[~images["filename"].isin(labeled_files)]
    print(f"Resuming from {len(labeled_files)} labeled files")
    print(f"{images.shape[0]} files remaining to label")
manual_contour_area = pd.DataFrame()

# %% [markdown]
# # come back here to process another image

# %%
# one at a time: display image, plot contours to find areas, save to csv, re

# display image on top of stack
orig_img = images.iloc[-1]["image"]
plt.imshow(orig_img, cmap="gray")
```

```
plt.show()

# reset updating_img
updating_img = orig_img.copy()

# %% [markdown]
# # remove small contours

# %%
prev_img = updating_img.copy()
curr_img = orig_img.copy()
# work on updating img?
if input("Work on updating image? (y/n) ") == "y":
    curr_img = updating_img

# ensure the image is in grayscale before finding contours
if len(curr_img.shape) == 3:
    curr_img = cv2.cvtColor(curr_img, cv2.COLOR_RGB2GRAY)

# remove small contours
contours, _ = cv2.findContours(curr_img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX
print("areas of top 10 contours:")
areas = [cv2.contourArea(cntr) for cntr in contours]
areas.sort(reverse=True)
print(areas[:10])
time.sleep(0.1)  # to ensure print completes before input
cutoff_area = input("enter cutoff area for contours (default 1000): ")
try:
    cutoff_area = int(cutoff_area)
except:  # default
    cutoff_area = 1000
for cntr in contours:
    if cv2.contourArea(cntr) > cutoff_area:
        continue
    convHull = cv2.convexHull(cntr)
    cv2.drawContours(curr_img, [convHull], -1, (0, 0, 0), thickness=cv2.FILL
curr_img = cv2.cvtColor(curr_img, cv2.COLOR_GRAY2RGB)
plt.imshow(curr_img)
plt.show()

updating_img = curr_img.copy()

# %% [markdown]
# # remove large contours

# %%
prev_img = updating_img.copy()
curr_img = orig_img.copy()
# work on updating img?
if input("Work on updating image? (y/n) ") == "y":
    curr_img = updating_img

# ensure the image is in grayscale before finding contours
if len(curr_img.shape) == 3:
    curr_img = cv2.cvtColor(curr_img, cv2.COLOR_RGB2GRAY)
```

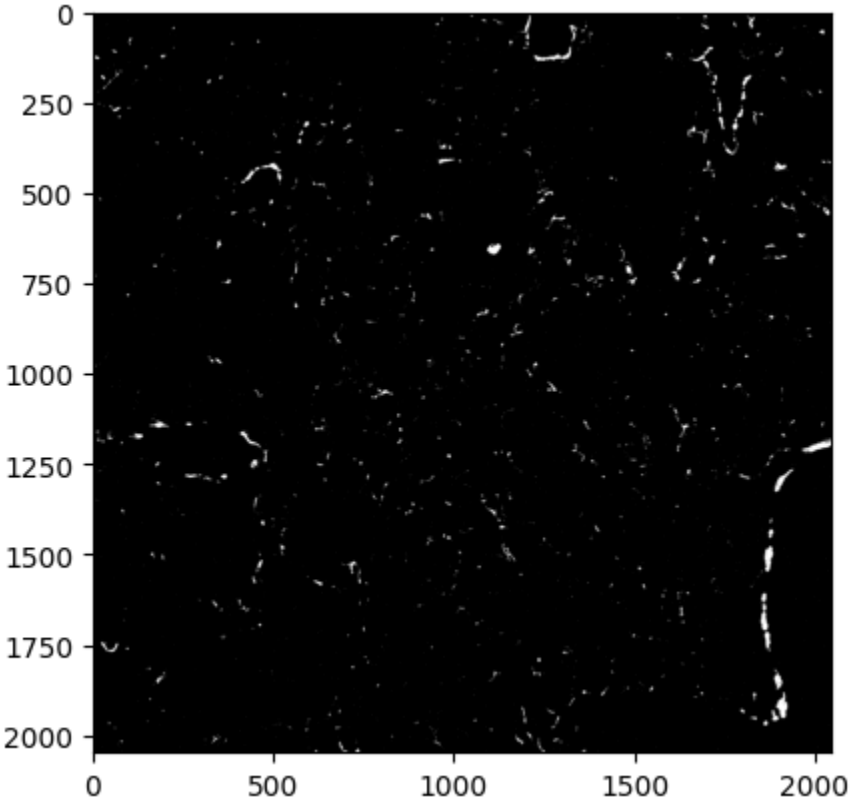
```
# remove large contours
contours, _ = cv2.findContours(curr_img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX
print("areas of top 10 contours:")
areas = [cv2.contourArea(cntr) for cntr in contours]
areas.sort(reverse=True)
print(areas[:10])
cutoff_area = input("enter cutoff area for contours (default 1000): ")
try:
    cutoff_area = int(cutoff_area)
except:  # default
    cutoff_area = 1000
for cntr in contours:
    if cv2.contourArea(cntr) < cutoff_area:
        continue
    convHull = cv2.convexHull(cntr)
    cv2.drawContours(curr_img, [convHull], -1, (0, 0, 0), thickness=cv2.FILL
curr_img = cv2.cvtColor(curr_img, cv2.COLOR_GRAY2RGB)
plt.imshow(curr_img)
plt.show()

updating_img = curr_img.copy()

# %% [markdown]
# # remove specific contour

# %%
prev_img = updating_img.copy()
curr_img = orig_img.copy()
# work on updating img?
if input("Work on updating image? (y/n) ") == "y":
    curr_img = updating_img

# ensure the image is in grayscale before finding contours
if len(curr_img.shape) == 3:
    curr_img = cv2.cvtColor(curr_img, cv2.COLOR_RGB2GRAY)

contours, _ = cv2.findContours(curr_img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX
areas = [cv2.contourArea(cntr) for cntr in contours]
areas.sort(reverse=True)
print("areas of top 10 contours:")
print(areas[:10])

plt.imshow(curr_img, cmap="gray")
plt.show()

# remove specific contours
time.sleep(0.1)  # to ensure print completes before input
cutoff_area = input("enter specific area for contours (default 1000): ")
try:
    cutoff_area = float(cutoff_area)
except:  # default
    cutoff_area = -1  # remove nothing
for cntr in contours:
    if cv2.contourArea(cntr) != cutoff_area:
        continue
    convHull = cv2.convexHull(cntr)
```

```
    cv2.drawContours(curr_img, [convHull], -1, (0, 0, 0), thickness=cv2.FILL
curr_img = cv2.cvtColor(curr_img, cv2.COLOR_GRAY2RGB)
plt.imshow(curr_img)
plt.show()

updating_img = curr_img.copy()

# %% [markdown]
# # dilate

# %%
# dilate
prev_img = updating_img.copy()
curr_img = orig_img.copy()
if input("Work on updating image for dilation? (y/n) ") == "y":
    curr_img = updating_img

plt.imshow(curr_img, cmap="gray")
plt.show()

dilate_size = input("Enter kernel size for dilation (default 15): ")
try:
    dilate_size = int(dilate_size)
except:
    dilate_size = 15

img_dilate = cv2.dilate(
    curr_img, np.ones((dilate_size, dilate_size), np.uint8), iterations=1
)
updating_img = img_dilate.copy()
plt.imshow(img_dilate, cmap="gray")
plt.show()

# %% [markdown]
# # erode

# %%
prev_img = updating_img.copy()
curr_img = orig_img.copy()
if input("Work on updating image for erosion? (y/n) ") == "y":
    curr_img = updating_img

plt.imshow(curr_img, cmap="gray")
plt.show()

erode_size = input("Enter kernel size for erosion (default 15): ")
try:
    erode_size = int(erode_size)
except:
    erode_size = 15

curr_img = cv2.erode(
    curr_img, np.ones((erode_size, erode_size), np.uint8), iterations=1
)
plt.imshow(curr_img, cmap="gray")
plt.show()
```

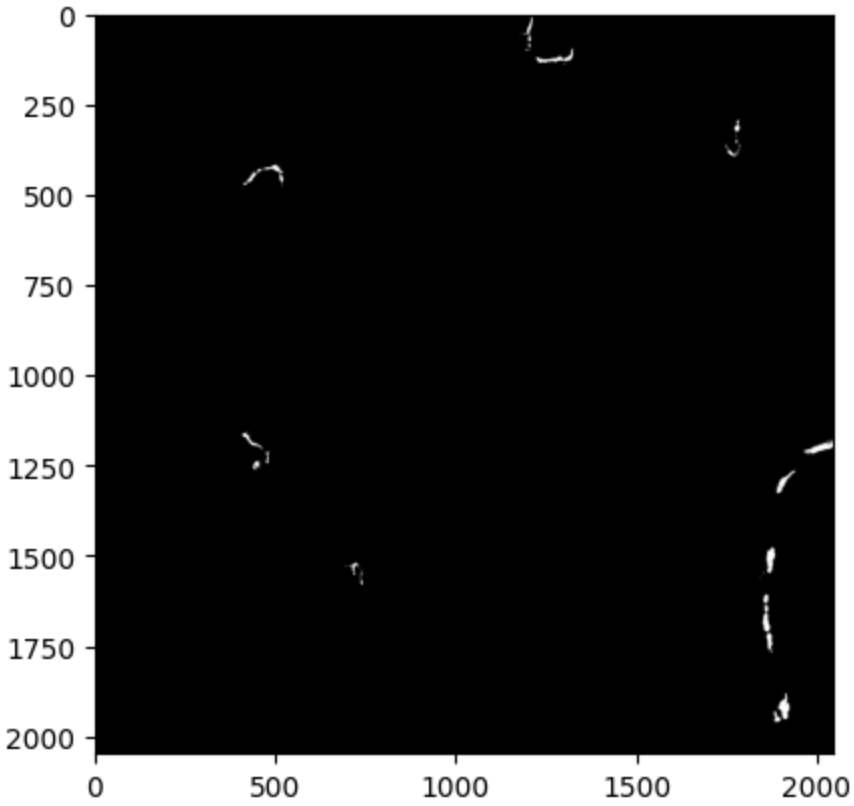
```
updating_img = curr_img.copy()

# %%
# flip image (black-white)
if input("Flip image (black-white)? (y/n) ") == "y":
    curr_img = cv2.bitwise_not(updating_img)
    plt.imshow(curr_img, cmap="gray")
    plt.show()
    updating_img = curr_img.copy()

# %% [markdown]
# # check progress

# %%
plt.imshow(orig_img, cmap="gray")
plt.show()

plt.imshow(updating_img)
plt.show()

# mask original image with updating_img
masked_orig = cv2.bitwise_and(
    orig_img, orig_img, mask=cv2.cvtColor(updating_img, cv2.COLOR_RGB2GRAY)
)
plt.imshow(masked_orig, cmap="gray")
plt.show()

# %% [markdown]
# # undo

# %%
updating_img = prev_img.copy()
plt.imshow(updating_img, cmap="gray")
plt.show()

# %% [markdown]
# # save progress to csv, pop processed image

# %%
if input("save to progress file? (y/n) ") != "y":
    raise Exception("please rerun the cell to edit the contour again.")

area = np.sum(updating_img > 0)
print(f"Area: {area} pixels")
manual_contour_area = (
    pd.read_csv(progress_path)
    if os.path.exists(progress_path)
    else pd.DataFrame(columns=["filename", "depth", "area_pixels"])
)
manual_contour_area = pd.concat(
    [
        manual_contour_area,
        pd.DataFrame(
            [
                {
                    "filename": images.iloc[-1]["filename"],
```

```
                    "depth": images.iloc[-1]["depth"],
                    "area_pixels": area,
                }
            ]
        ),
    ],
    ignore_index=True,
)
manual_contour_area.to_csv(progress_path, index=False)
images = images.iloc[:-1]
print(f"{images.shape[0]} files remaining to label")
```

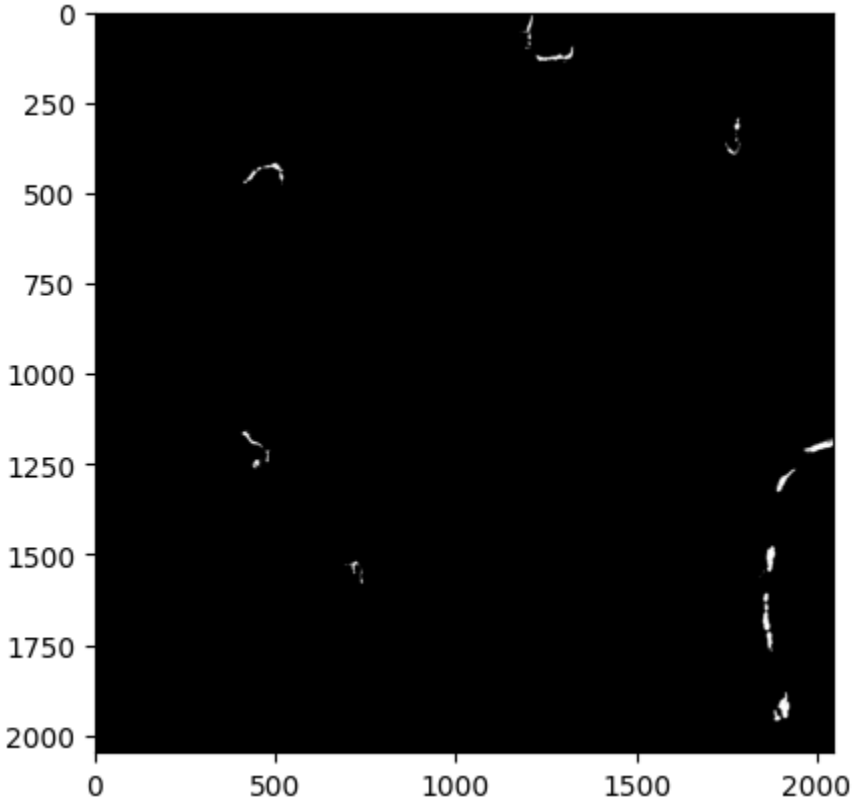
```
(20, 3)
(10, 3)
(10, 3)
```



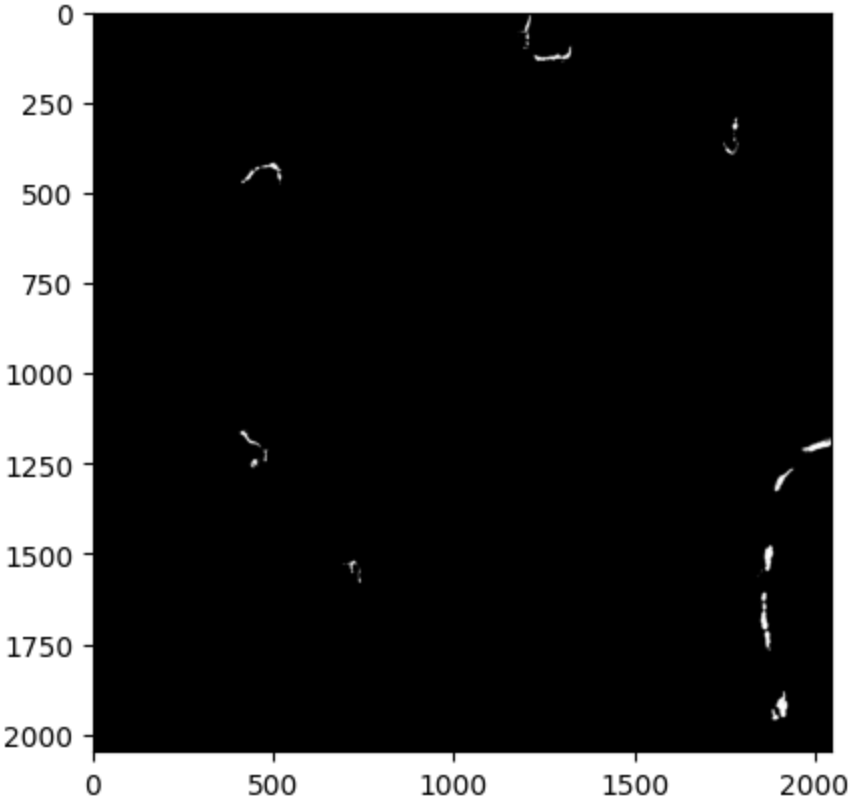

```
areas of top 10 contours:
[2243.5, 1974.0, 1842.5, 1769.0, 1615.5, 1520.5, 1465.5, 1280.5, 1124.5, 108
7.0]
```

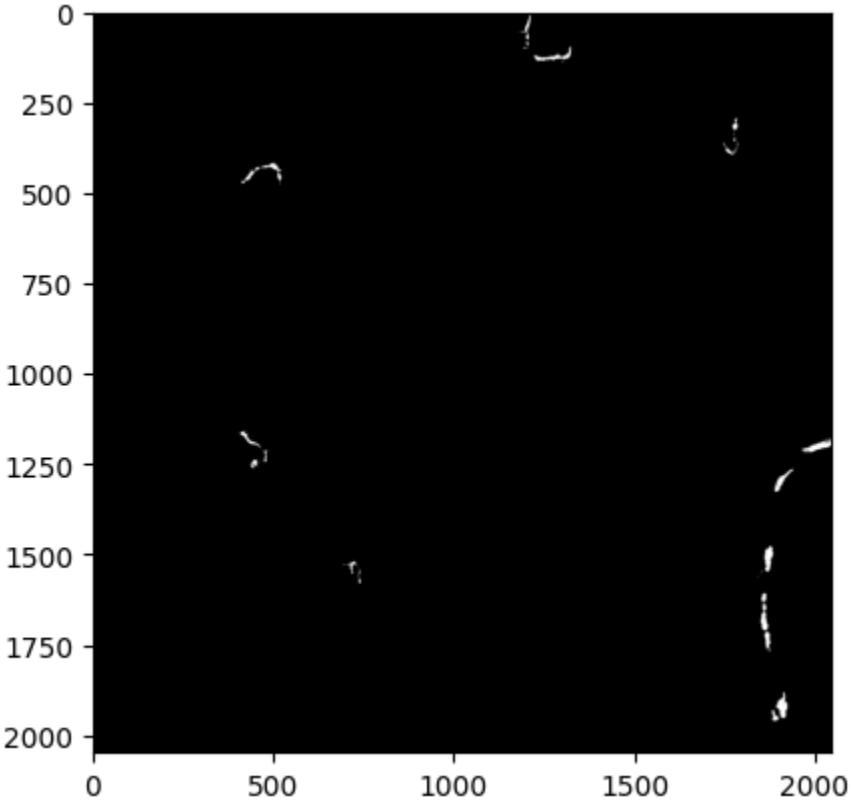


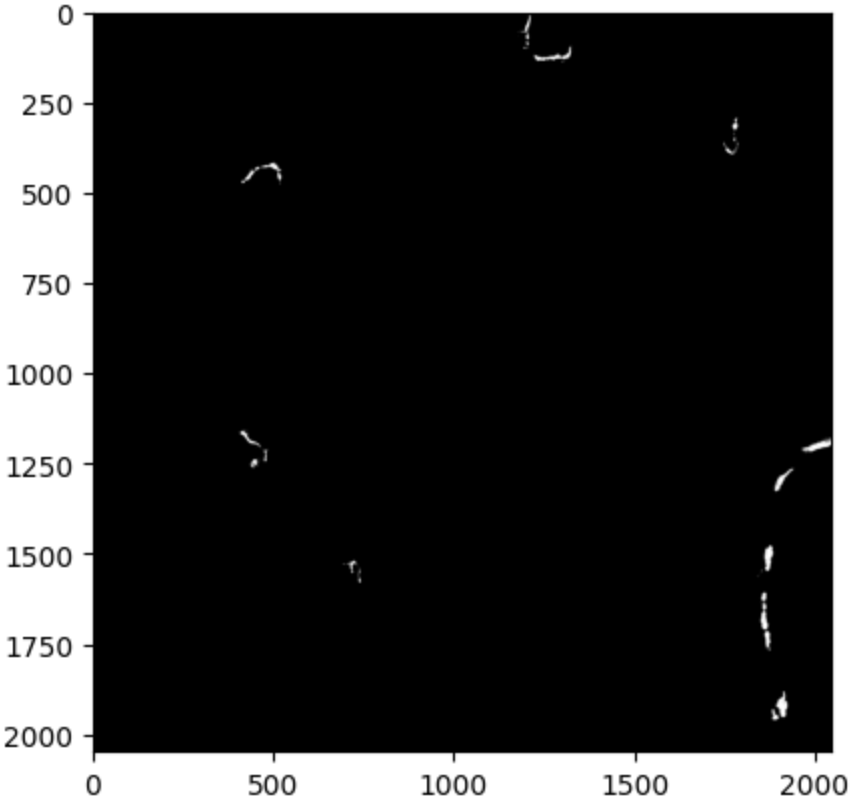
areas of top 10 contours:
[2243.5, 1974.0, 1842.5, 1767.0, 1615.5, 1520.5, 1465.5, 1280.5, 1124.0, 108
7.0]

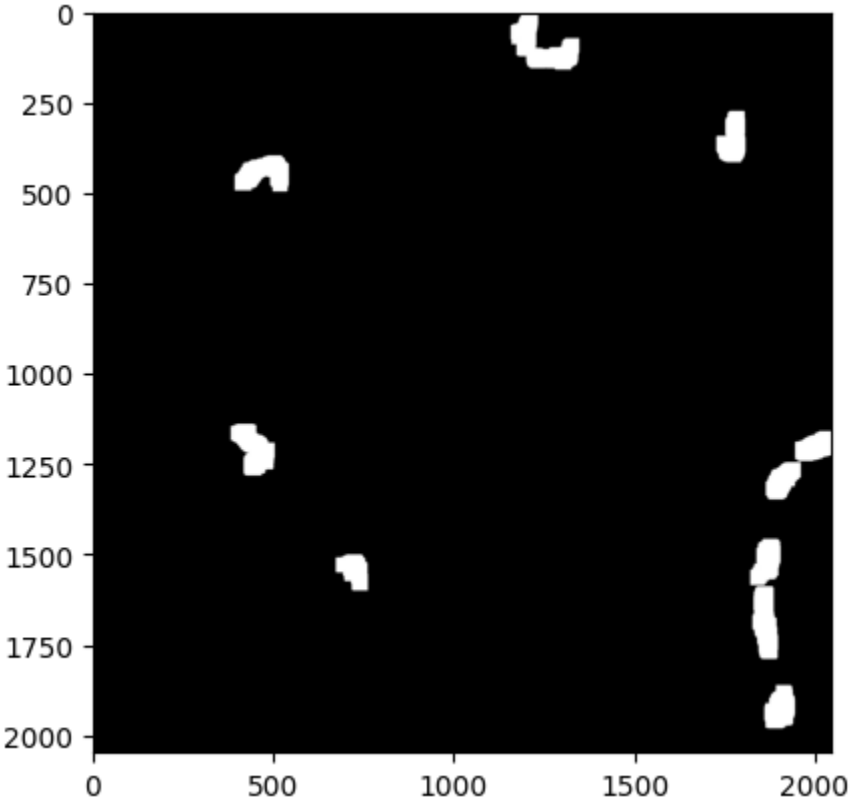


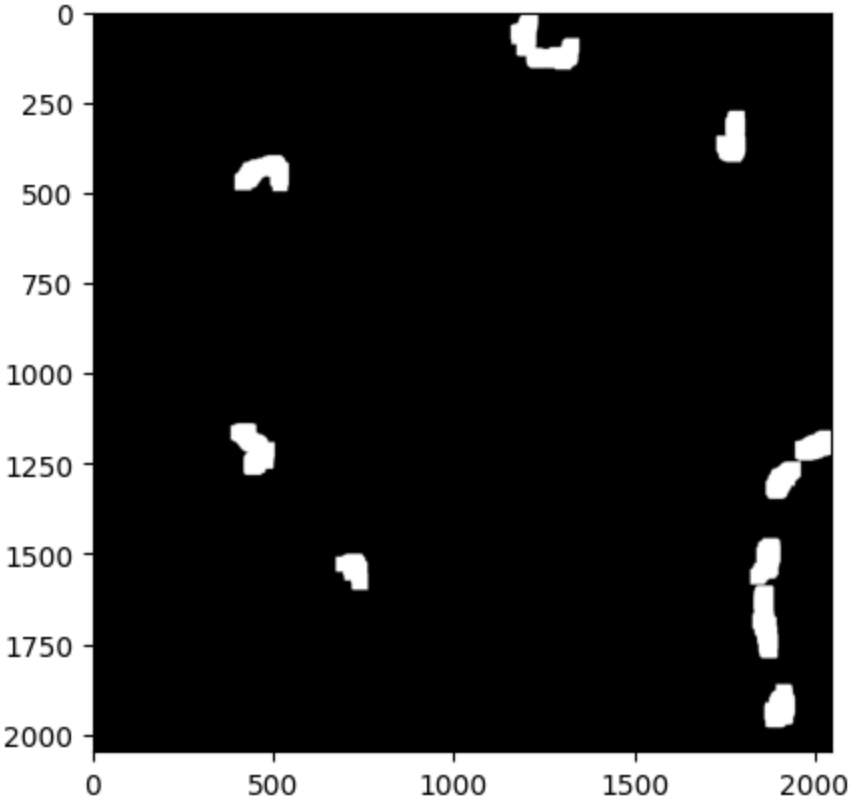
areas of top 10 contours:
[2243.5, 1974.0, 1842.5, 1767.0, 1615.5, 1520.5, 1465.5, 1280.5, 1124.0, 108
7.0]

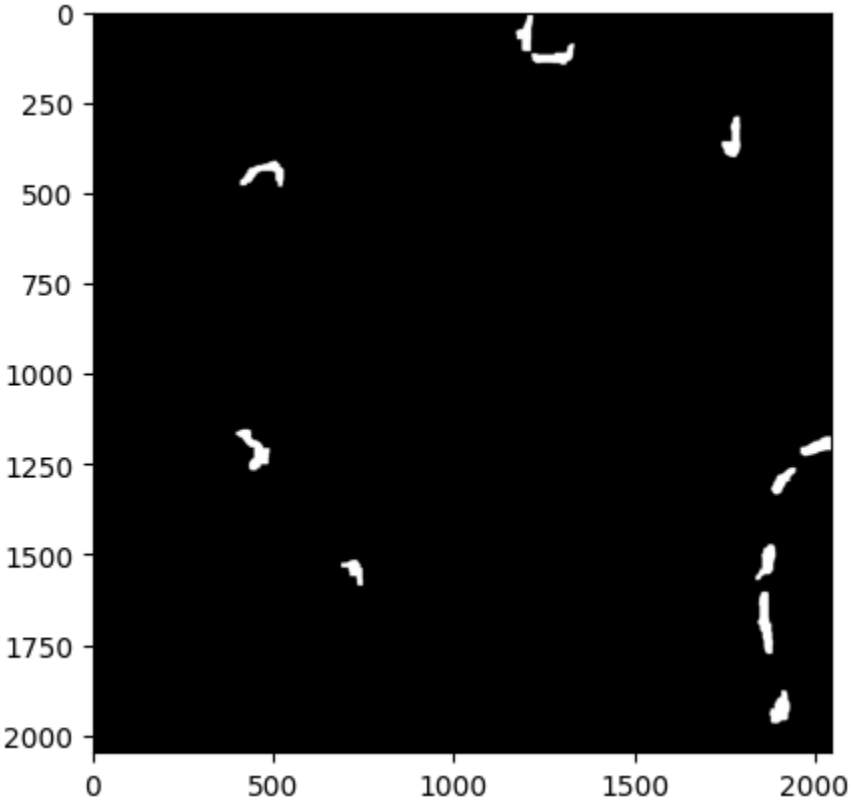

0
250
500
750
1000
1250
1500
1750
2000
0
500
1000
1500
2000

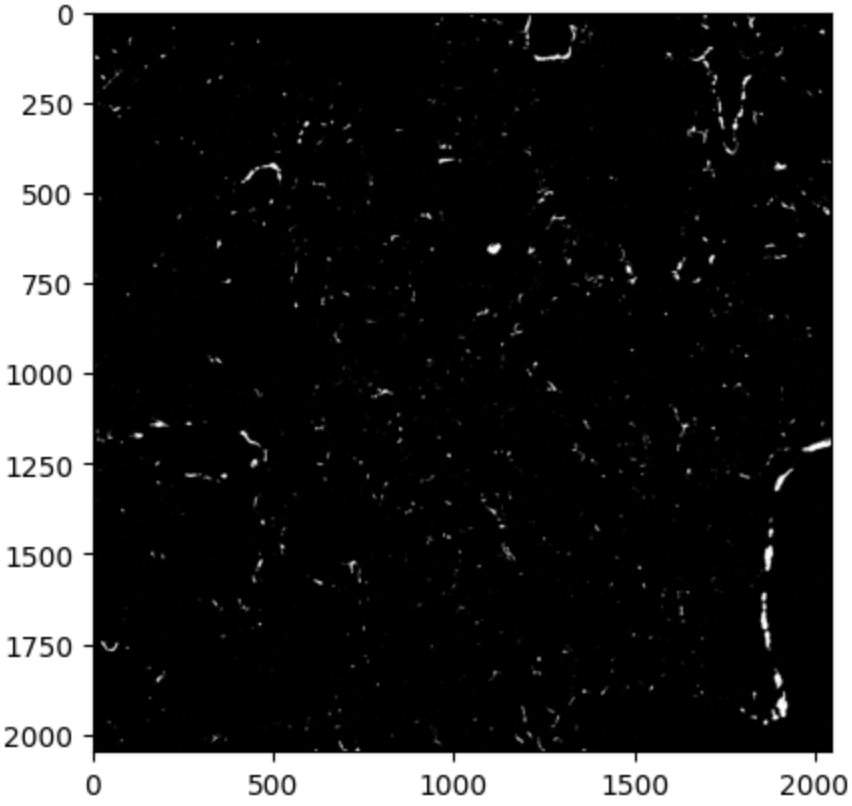

0
250
500
750
1000
1250
1500
1750
2000
0
500
1000
1500
2000

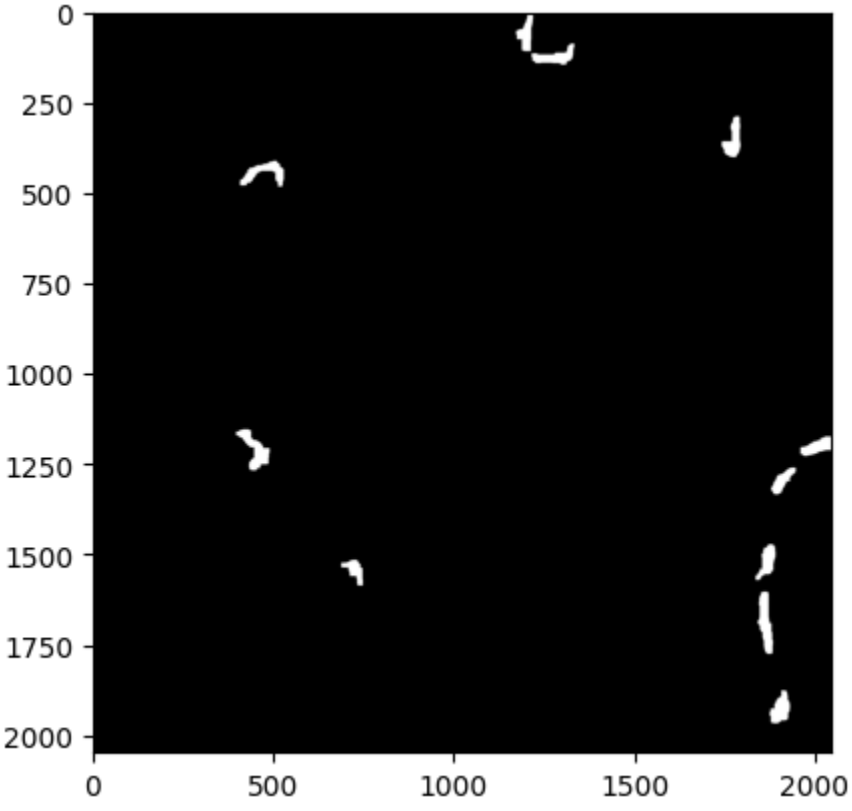

0
250
500
750
1000
1250
1500
1750
2000
0
500
1000
1500
2000

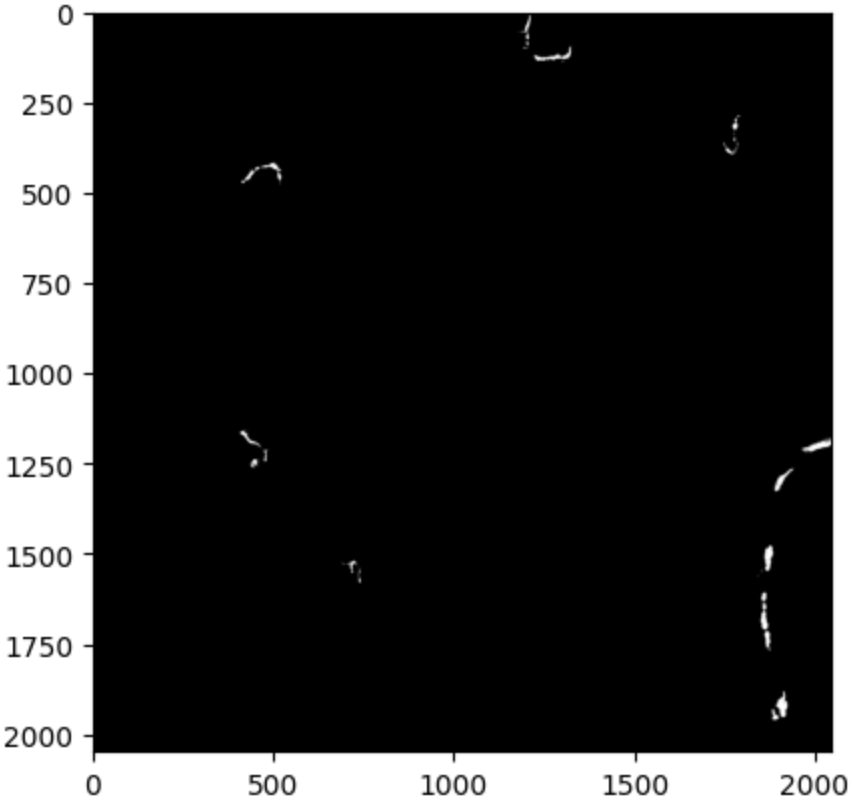

0
250
500
750
1000
1250
1500
1750
2000
0
500
1000
1500
2000

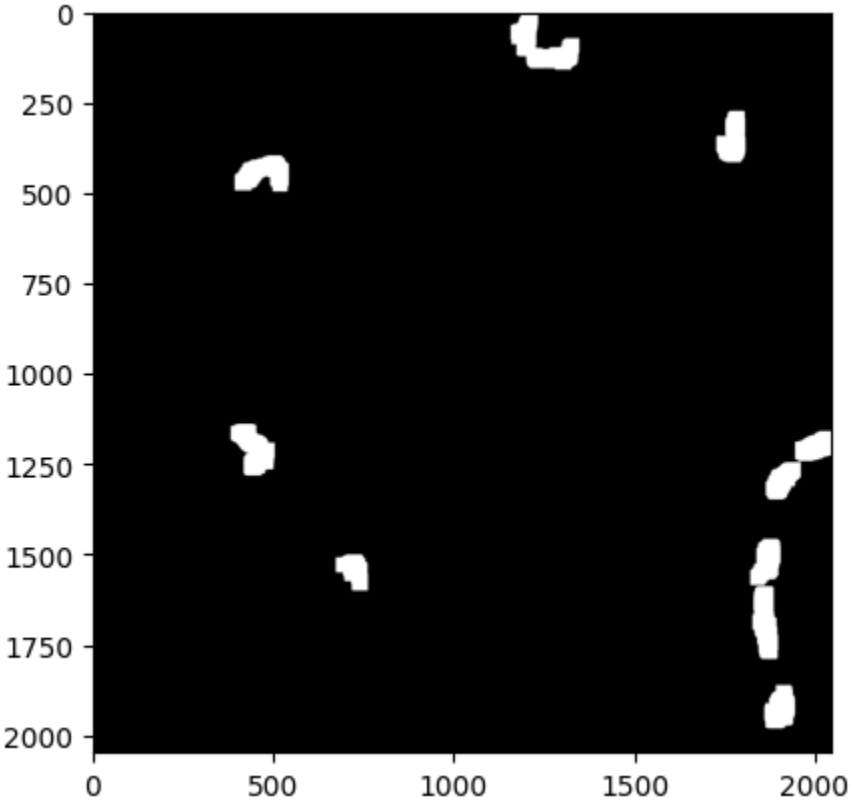

0
250
500
750
1000
1250
1500
1750
2000
0
500
1000
1500
2000


0
250
500
750
1000
1250
1500
1750
2000
0
500
1000
1500
2000


0
250
500
750
1000
1250
1500
1750
2000
0
500
1000
1500
2000


0
250
500
750
1000
1250
1500
1750
2000
0
500
1000
1500
2000


0
250
500
750
1000
1250
1500
1750
2000
0
500
1000
1500
2000


0
250
500
750
1000
1250
1500
1750
2000
0
500
1000
1500
2000

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
Cell In[9], line 268
    263 # %% [markdown]
    264 # # save progress to csv, pop processed image
    265
    266 # %%
    267 if input("save to progress file? (y/n) ") != "y":
--> 268     raise Exception("please rerun the cell to edit the contour agai
n.")
    270 area = np.sum(updating_img > 0)
    271 print(f"Area: {area} pixels")

Exception: please rerun the cell to edit the contour again.
```

In [10]:

```python
"""FILE: better_estimation.ipynb"""
# %%
import pandas as pd
import numpy as np
import cv2
import os
from termcolor import colored
import matplotlib.pyplot as plt

images = pd.DataFrame()

data_path = r"data"
imaging_path = r"imaging_subset"

filenames = os.listdir(os.path.join(data_path, imaging_path))
depths = pd.read_csv(os.path.join(data_path, "depths.csv"))

for i in filenames:
    img = cv2.imread(os.path.join(data_path, imaging_path, i), 0)
    try:
        depth = depths[depths["Filenames"].str.lower() == i.lower()][
            "Depth from lung surface (in micrometers) where image was acquir
        ].values[0]
        # some files are named with SK658 and some with Sk658
    except IndexError:
        print(f"couldn't find depth for file {i}")
        continue
    images = pd.concat(
        [images, pd.DataFrame([{"filename": i, "image": img, "depth": depth}
        ignore_index=True,
    )
display(images.head())

# %%
# add area column

areas = pd.read_csv(os.path.join(data_path, "manual_contour_area.csv"))
if not ("area_pixels" in images.columns):
    images = images.merge(areas, on=["filename", "depth"], how="left")

display(images.head())
```

```
# %%
white_percents_w_area = []

for x_glm_all in range(len(images)):
    _, binary = cv2.threshold(
        images.iloc[x_glm_all]["image"], 127, 255, cv2.THRESH_BINARY
    )
    area = images.iloc[x_glm_all]["area_pixels"]
    white = np.sum(binary == 255)
    black = np.sum(binary == 0)

    white_percent = 100 * (white / (black + white - area))
    white_percents_w_area.append(white_percent)

white_percents = []

for x_glm_all in range(len(images)):
    _, binary = cv2.threshold(
        images.iloc[x_glm_all]["image"], 127, 255, cv2.THRESH_BINARY
    )
    white = np.sum(binary == 255)
    black = np.sum(binary == 0)

    white_percent = 100 * (white / (black + white))
    white_percents.append(white_percent)

# %%
df = pd.DataFrame(
    {
        "filename": images["filename"],
        "depth": images["depth"],
        "white_percent_w_area": white_percents_w_area,
        "white_percent": white_percents,
    }
)
if input("save to file? (y/n) ") == "y":
    df.to_csv(os.path.join(data_path, "merged_all.csv"), index=False)


# %%
# display graph
plt.scatter(
    images["depth"], white_percents_w_area, marker="o", linestyle="-", color
)
plt.title("Plot of depth of image vs percentage white pixels (area accounted
plt.xlabel("depth of image")
plt.ylabel("white pixels as a percentage of total pixels")
plt.grid(True)
plt.show()

plt.scatter(images["depth"], white_percents, marker="o", linestyle="-", colo
plt.title("Plot of depth of image vs percentage white pixels")
plt.xlabel("depth of image")
plt.ylabel("white pixels as a percentage of total pixels")
plt.grid(True)
```

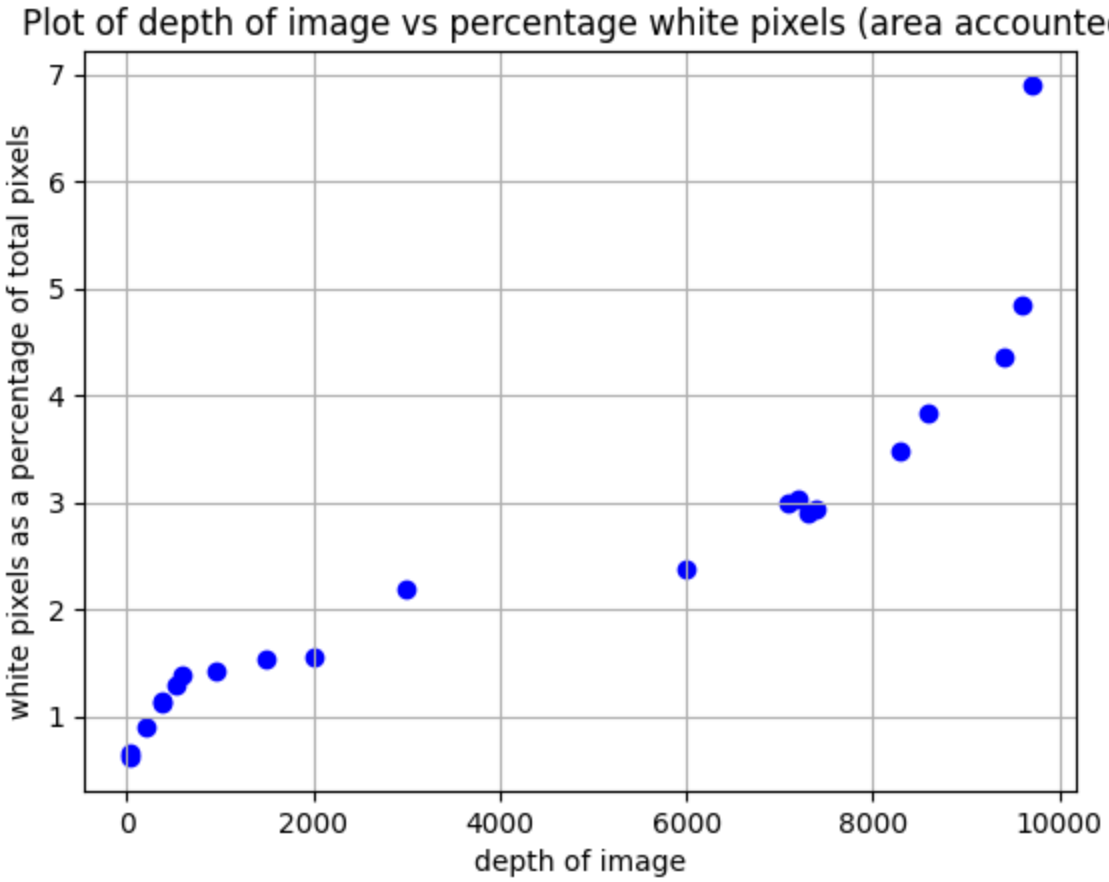
```
plt.show()

# %%
# generalized linear model
import statsmodels.api as sm

# prepare the data
x_glm = df["depth"]
y_glm = df["white_percent_w_area"]

# add a constant for the intercept
x_const = sm.add_constant(x_glm)

# fit glm
glm_model_area = sm.GLM(
    y_glm, x_const, family=sm.families.Gaussian(link=sm.families.links.Log()
).fit()

# print the summary
print(glm_model_area.summary())

# calculate R-squared
fitted_values = glm_model_area.fittedvalues
ss_total = ((y_glm - y_glm.mean()) ** 2).sum()
ss_residual = ((y_glm - fitted_values) ** 2).sum()
r_squared = 1 - (ss_residual / ss_total)
print(f"R-squared: {r_squared}")

# plot the regression curve
x_pred = np.linspace(x_glm.min(), x_glm.max(), 100)
x_pred_const = sm.add_constant(x_pred)
y_pred = glm_model_area.predict(x_pred_const)
plt.scatter(x_glm, y_glm, color="blue", label="Data")
plt.plot(x_pred, y_pred, color="red", linewidth=2, label="GLM Log Link Fit")
plt.text(
    0.05,
    0.95,
    f"R-squared: {r_squared:.4f}",
    transform=plt.gca().transAxes,
    fontsize=10,
    verticalalignment="top",
)
plt.xlabel("Depth of image")
plt.ylabel("White pixel percentage (with area)")
plt.legend()
plt.show()

# %%
# generalized linear model
import statsmodels.api as sm

images_all = pd.DataFrame()

data_path = r"data"
imaging_path = r"imaging"
filenames = os.listdir(os.path.join(data_path, imaging_path))
```

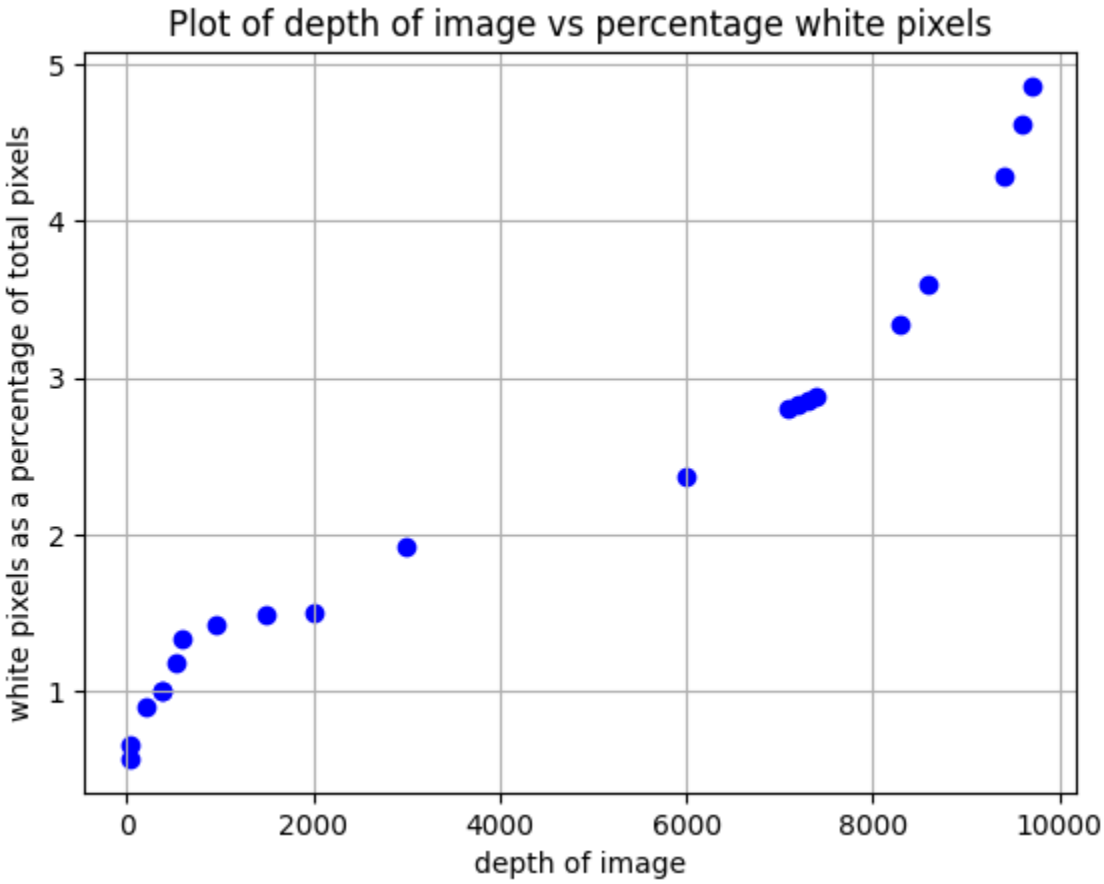
```
images_all = pd.read_csv(os.path.join(data_path, "pct_white_pixels.csv"))

# prepare the data
x_glm_all = images_all["depth"]
y_glm_all = images_all["white_percent"]

# add a constant for the intercept
x_const = sm.add_constant(x_glm_all)

# fit a GLM
glm_model_no_area = sm.GLM(
    y_glm_all, x_const, family=sm.families.Poisson(link=sm.families.links.Lo
).fit()

# print the summary
print(glm_model_no_area.summary())

# calculate R-squared
fitted_values_all = glm_model_no_area.fittedvalues
ss_total_all = ((y_glm_all - y_glm_all.mean()) ** 2).sum()
ss_residual_all = ((y_glm_all - fitted_values_all) ** 2).sum()
r_squared_all = 1 - (ss_residual_all / ss_total_all)
print(f"R-squared: {r_squared_all}")

# plot the regression curve
x_pred_all = np.linspace(x_glm_all.min(), x_glm_all.max(), 100)
x_pred_const = sm.add_constant(x_pred_all)
y_pred = glm_model_no_area.predict(x_pred_const)
plt.scatter(x_glm_all, y_glm_all, color="blue", label="Data")
plt.plot(x_pred_all, y_pred, color="red", linewidth=2, label="GLM Log Link F
plt.legend()
plt.text(
    0.05,
    0.95,
    f"R-squared: {r_squared_all:.4f}",
    transform=plt.gca().transAxes,
    fontsize=10,
    verticalalignment="top",
)
plt.xlabel("Depth of image")
plt.ylabel("White pixel percentage (no area)")
plt.show()

# %%
# interpolate points using glm (with area accounted for)

inp_depth = input("Enter depth to estimate white pixel percentage at: ")
inp_depth = float(inp_depth)

# single-point prediction
x_point = np.array([[1, inp_depth]])  # include constant if the model was fi
y_point = glm_model_area.predict(x_point)[0]

# curve
x_curve = np.linspace(x_glm_all.min(), x_glm_all.max(), 100)
X_curve = sm.add_constant(x_curve)
```

```
y_curve = glm_model_area.predict(X_curve)

plt.scatter(x_glm, y_glm, color="blue", label="Data")
plt.plot(x_pred, y_pred, color="red", linewidth=2, label="GLM Log Link Fit")
plt.text(
    0.05,
    0.95,
    f"R-squared: {r_squared:.4f}",
    transform=plt.gca().transAxes,
    fontsize=10,
    verticalalignment="top",
)
# add interpolated point as star
plt.scatter(
    inp_depth, y_point, color="green", s=100, label="Interpolated Point", ma
)
plt.text(
    inp_depth,
    y_point,
    f"  ({inp_depth}, {y_point:.2f})",
    verticalalignment="bottom",
    horizontalalignment="center",
    fontsize=9,
    color="green",
)
plt.legend()
plt.xlabel("Depth of image")
plt.ylabel("White pixel percentage (with area)")
plt.show()
```

	filename	image	depth
0	MASK_SK658 Slobe ch010048.jpg	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...	540
1	MASK_Sk658 Llobe ch010061.jpg	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...	585
2	MASK_SK658 Slobe ch010103.jpg	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...	9600
3	MASK_SK658 Slobe ch010063.jpg	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...	7400
4	MASK_SK658 Slobe ch010113.jpg	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 1,...	7300

	filename	image	depth	area_pixels
0	MASK_SK658 Slobe ch010048.jpg	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...	540	339495
1	MASK_Sk658 Llobe ch010061.jpg	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...	585	175368
2	MASK_SK658 Slobe ch010103.jpg	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...	9600	192249
3	MASK_SK658 Slobe ch010063.jpg	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...	7400	85161
4	MASK_SK658 Slobe ch010113.jpg	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 1,...	7300	49410


Plot of depth of image vs percentage white pixels (area accounted)
white pixels as a percentage of total pixels
depth of image
0
2000
4000
6000
8000
10000
1
2
3
4
5
6
7

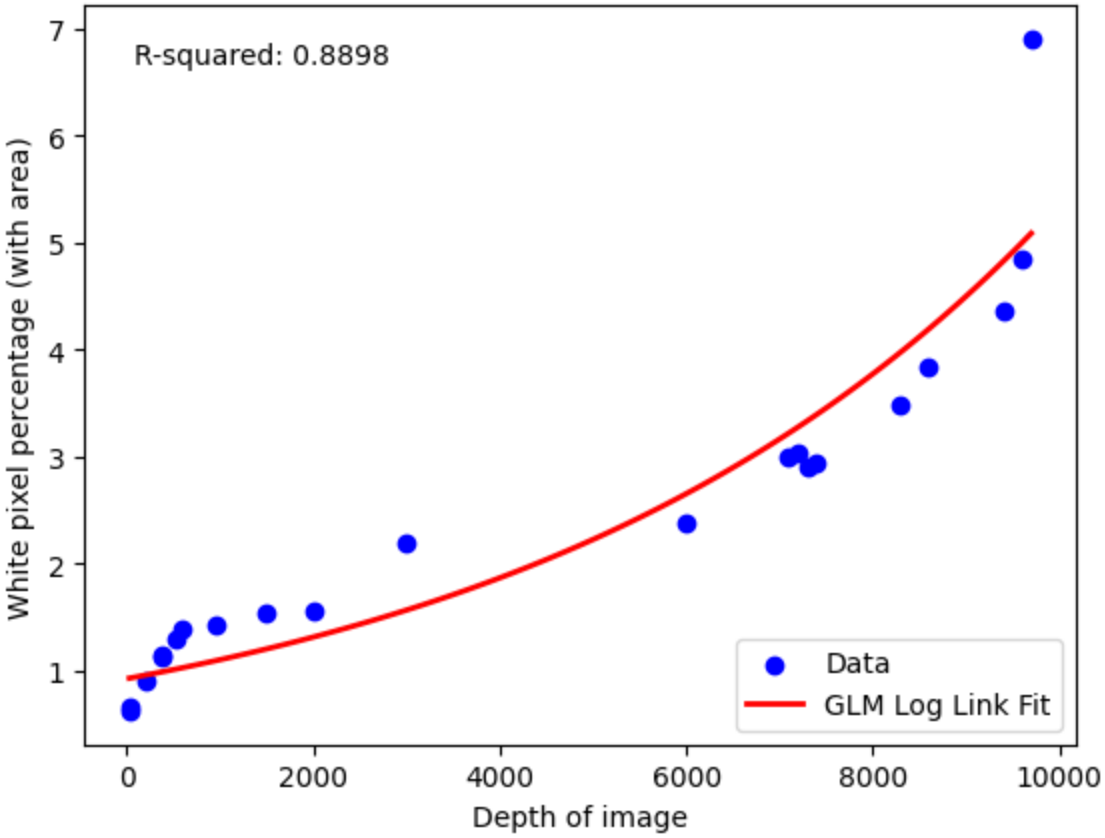

Plot of depth of image vs percentage white pixels
white pixels as a percentage of total pixels
depth of image
0
2000
4000
6000
8000
10000
1
2
3
4
5

Generalized Linear Model Regression Results

Dep. Variable:	white_percent_w_area	No. Observations:	21
Model:	GLM	Df Residuals:	19
Model Family:	Gaussian	Df Model:	1
Link Function:	Log	Scale:	0.29506
Method:	IRLS	Log-Likelihood:	-15.982
Date:	Thu, 16 Oct 2025	Deviance:	5.6062
Time:	09:52:27	Pearson chi2:	5.61
No. Iterations:	8	Pseudo R-squ. (CS):	0.9993
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	-0.0822	0.152	-0.542	0.588	-0.380	0.215
depth	0.0002	1.83e-05	9.610	0.000	0.000	0.000

R-squared: 0.889812301311898

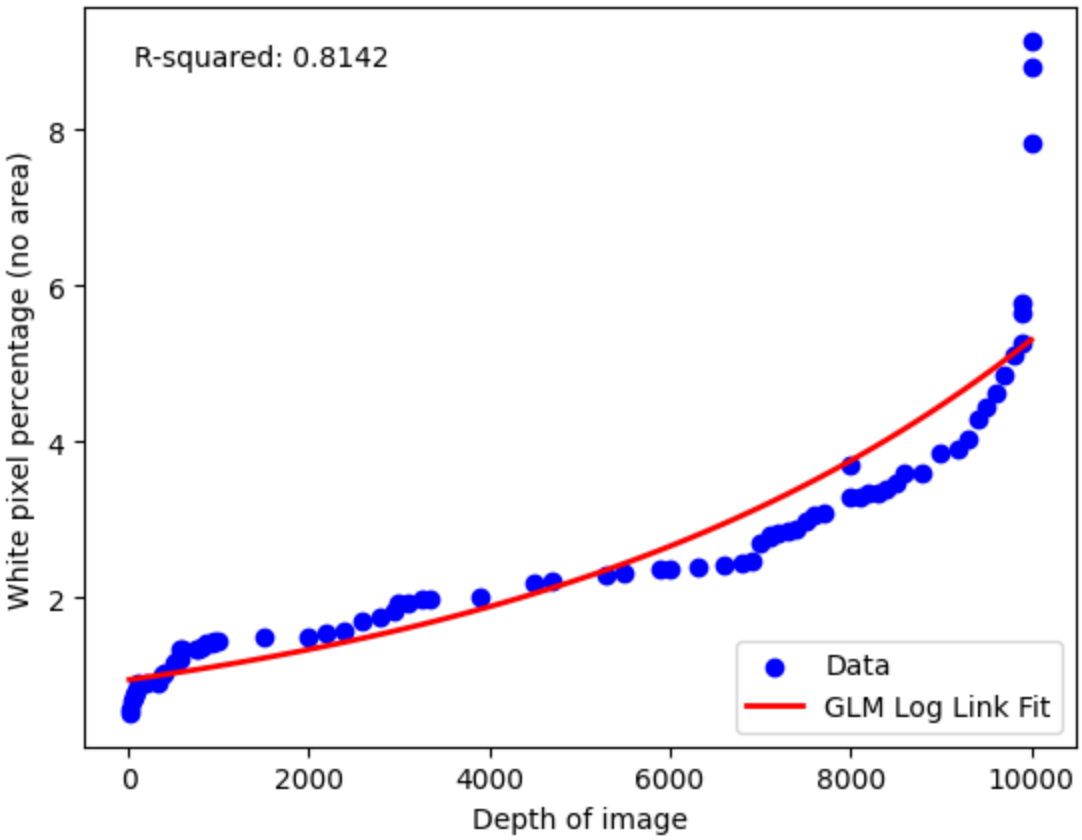

R-squared: 0.8898
White pixel percentage (with area)
Depth of image
Data
GLM Log Link Fit
0
2000
4000
6000
8000
10000
1
2
3
4
5
6
7

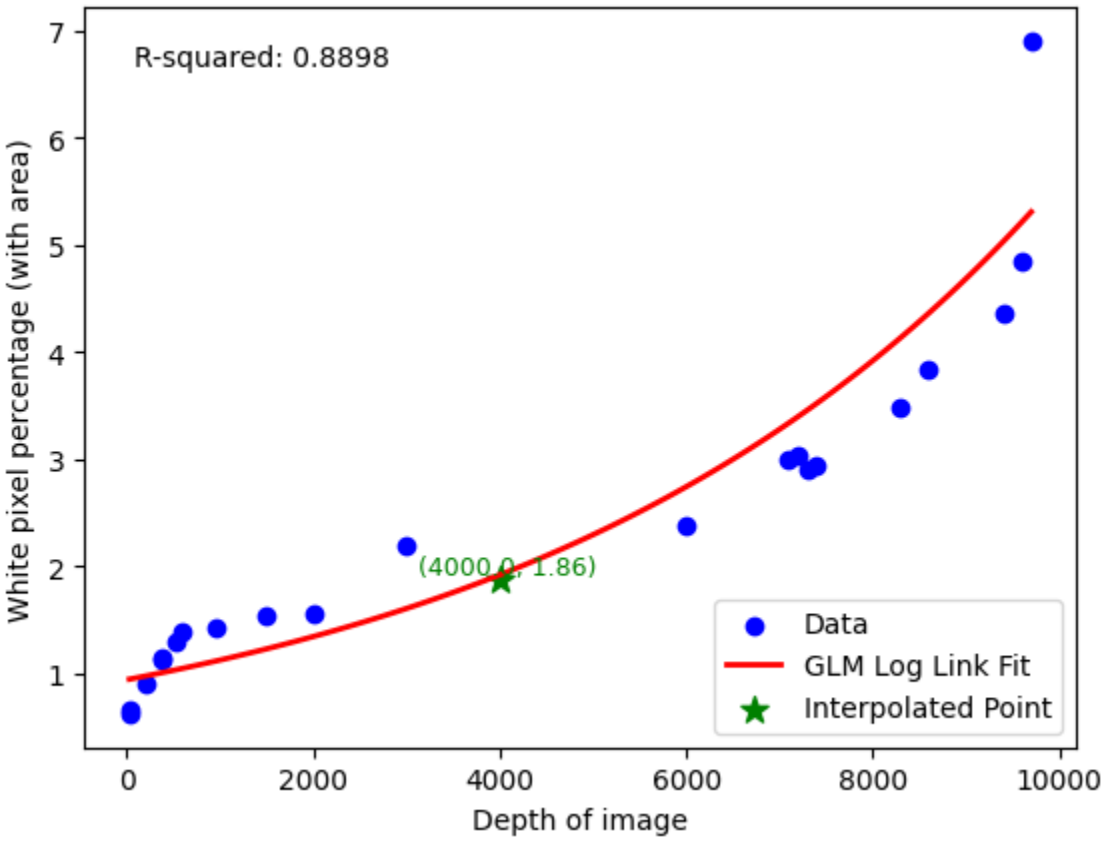
Generalized Linear Model Regression Results

Dep. Variable:	white_percent	No. Observations:	78
Model:	GLM	Df Residuals:	76
Model Family:	Poisson	Df Model:	1
Link Function:	Log	Scale:	1.0000
Method:	IRLS	Log-Likelihood:	-109.93
Date:	Thu, 16 Oct 2025	Deviance:	9.5621
Time:	09:52:27	Pearson chi2:	10.5
No. Iterations:	5	Pseudo R-squ. (CS):	0.5891
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	-0.0617	0.170	-0.363	0.716	-0.395	0.271
depth	0.0002	2.24e-05	7.726	0.000	0.000	0.000

R-squared: 0.8142249596372895





In [11]:
```
"""FILE: main.ipynb"""
# %%
# imports
import os
```

```
import cv2
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# load images


images_subset = pd.DataFrame()

data_path = r"data"
imaging_path = os.path.join(data_path, r"imaging_subset")
filenames = os.listdir(imaging_path)
depths = pd.read_csv(os.path.join(data_path, "depths.csv"))

for i in filenames:
    img = cv2.imread(os.path.join(imaging_path, i), 0)
    try:
        depth = depths[depths["Filenames"].str.lower() == i.lower()][
            "Depth from lung surface (in micrometers) where image was acquir
        ].values[0]
        # some files are named with SK658 and some with Sk658
    except IndexError:
        print(f"couldn't find depth for file {i}")
        continue
    images_subset = pd.concat(
        [images_subset, pd.DataFrame([{"filename": i, "image": img, "depth":
        ignore_index=True,
    )

print(images_subset.shape)

# %%
# merge pct_white_pixels.csv with manual_contour_area.csv on filename
pct_white_pixels = pd.read_csv(os.path.join(data_path, "pct_white_pixels.csv
manual_contour_area = pd.read_csv(os.path.join(data_path, "manual_contour_ar

merged_data = pd.merge(
    pct_white_pixels, manual_contour_area, on=["filename", "depth"], how="in
)
display(merged_data.head())

# %%
# create slobe vs llobe column
df = merged_data.copy()

df["slobe"] = df["filename"].str.contains("slobe", case=False)
display(df.head())

# %%
# T test for slobe vs llobe for white percent
import scipy.stats as stats

y = "slobe"

t_test_slobe_white_percent = stats.ttest_ind(
```

```
    df[df[y] == True]["white_percent"],
    df[df[y] == False]["white_percent"],
)

t_test_slobe_depth = stats.ttest_ind(
    df[df[y] == True]["depth"],
    df[df[y] == False]["depth"],
)

t_test_slobe_area_pixels = stats.ttest_ind(
    df[df[y] == True]["area_pixels"],
    df[df[y] == False]["area_pixels"],
)

# depth vs area pixels
correlation_depth_area_pixels = stats.pearsonr(df["depth"], df["area_pixels"

print("T test for slobe vs llobe for white percent:")
print(t_test_slobe_white_percent)

print("T test for slobe vs llobe for depth:")
print(t_test_slobe_depth)

print("T test for slobe vs llobe for area pixels:")
print(t_test_slobe_area_pixels)

print("Correlation between depth and area pixels:")
print(correlation_depth_area_pixels)

# %%
# MANOVA: joint effect of lobe type (slobe vs llobe) on depth and area_pixel
from statsmodels.multivariate.manova import MANOVA

df = df.copy()
if "lobe" not in df.columns:
    df["lobe"] = df["slobe"].map({True: "slobe", False: "llobe"})

# run MANOVA
maov = MANOVA.from_formula("depth + area_pixels ~ C(lobe)", data=df)
print(maov.mv_test())

# %%
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm

for dv in ["depth", "area_pixels"]:
    model = ols(f"{dv} ~ C(lobe)", data=df).fit()
    print(f"\nANOVA for {dv}:")
    print(anova_lm(model, typ=2))
```

```
(20, 3)
```

	filename	depth	white_percent	area_pixels
0	MASK_SK658 Slobe ch010048.jpg	540	1.179957	339495
1	MASK_Sk658 Llobe ch010061.jpg	585	1.335216	175368
2	MASK_SK658 Slobe ch010103.jpg	9600	4.619193	192249
3	MASK_SK658 Slobe ch010063.jpg	7400	2.882719	85161
4	MASK_SK658 Slobe ch010113.jpg	7300	2.859545	49410

	filename	depth	white_percent	area_pixels	slobe
0	MASK_SK658 Slobe ch010048.jpg	540	1.179957	339495	True
1	MASK_Sk658 Llobe ch010061.jpg	585	1.335216	175368	False
2	MASK_SK658 Slobe ch010103.jpg	9600	4.619193	192249	True
3	MASK_SK658 Slobe ch010063.jpg	7400	2.882719	85161	True
4	MASK_SK658 Slobe ch010113.jpg	7300	2.859545	49410	True

```
T test for slobe vs llobe for white percent:
TtestResult(statistic=np.float64(3.469815814239402), pvalue=np.float64(0.002
565092387819093), df=np.float64(19.0))
T test for slobe vs llobe for depth:
TtestResult(statistic=np.float64(3.133607768897513), pvalue=np.float64(0.005
469240026446949), df=np.float64(19.0))
T test for slobe vs llobe for area pixels:
TtestResult(statistic=np.float64(0.9099205557822232), pvalue=np.float64(0.37
42635498003356), df=np.float64(19.0))
Correlation between depth and area pixels:
PearsonRResult(statistic=np.float64(0.15346973591218055), pvalue=np.float64
(0.5065787295909403))
                 Multivariate linear model
============================================================

------------------------------------------------------------
       Intercept        Value  Num DF  Den DF F Value Pr > F
------------------------------------------------------------
          Wilks' lambda 0.6498 2.0000 18.0000  4.8494 0.0207
         Pillai's trace 0.3502 2.0000 18.0000  4.8494 0.0207
 Hotelling-Lawley trace 0.5388 2.0000 18.0000  4.8494 0.0207
    Roy's greatest root 0.5388 2.0000 18.0000  4.8494 0.0207
------------------------------------------------------------

------------------------------------------------------------
        C(lobe)         Value  Num DF  Den DF F Value Pr > F
------------------------------------------------------------
          Wilks' lambda 0.6458 2.0000 18.0000  4.9365 0.0195
         Pillai's trace 0.3542 2.0000 18.0000  4.9365 0.0195
 Hotelling-Lawley trace 0.5485 2.0000 18.0000  4.9365 0.0195
    Roy's greatest root 0.5485 2.0000 18.0000  4.9365 0.0195
============================================================


ANOVA for depth:
                sum_sq    df         F    PR(>F)
C(lobe)   9.994400e+07   1.0  9.819498  0.005469
Residual  1.933842e+08  19.0       NaN       NaN

ANOVA for area_pixels:
                sum_sq    df         F    PR(>F)
C(lobe)   6.609092e+10   1.0  0.827955  0.374264
Residual  1.516661e+12  19.0       NaN       NaN
```

Verify and validate your analysis:

(Describe how you checked to see that your analysis gave you an answer that you believe (verify). Describe how your determined if your analysis gave you an answer that is supported by other evidence (e.g., a published paper).)

"All IPF lungs showed [...] subpleural and basilar predominant reticulation, traction bronchiectasis, and honeycombing."

Published in Respiratory Research, these fibrotic features were found to be more prevalent in the lower lung: [1]

Subpleural reticulation Ground-glass opacification Honeycombing

“A definite UIP pattern on HRCT is present if the following four radiological criteria are met: 1) subpleural basal predominance; 2) reticular abnormality; 3) honeycombing with or without traction bronchiectasis; and 4) absence of features listed as inconsistent with UIP pattern.” [2]

According to the European Respiratory Journal, one of the 4 features of a usual interstitial pneumonia (UIP) pattern (a pattern key in diagnosing IPF) is “basal predominance”.

“Image also shows subpleural mixed areas of ground-glass opacity and reticulation, with lower lung zone predominance (arrows).” [3]

Published in American Journal of Roentgenology:

58 year old woman with UIP having ground glass opacity and reticulation predominantly in her lower lungs.

Our model captures depth-dependent patterns of fibrosis that align with known pathology of IPF, particularly greater fibrotic prevalence in lower lung regions. We conclude that lung fibrosis increases alongside depth of the transverse plane. Predictions from our pipeline are consistent across multiple validation datasets and supported by findings in published literature.

Conclusions and Ethical Implications:

(Think about the answer your analysis generated, draw conclusions related to your overarching question, and discuss the ethical implications of your conclusions.)

From our data analysis, it is possible to create a pipeline that predicts the extent of lung fibrosis at different depths within the lung.

This was validated by regression models, R2 values, and accurate interpolations.

As we’ve found that the fibrotic density increases logarithmically with depth, lung biopsy device designs should implement a configurable depth “ladder” where increase in fibrotic density is easily detectable.

As increasing device’s maximum depth will have diminishing returns, the device is not required to travel deep into the lungs, decreasing risk of bleeding or other injury.

Model must be trained on diverse populations to generalize fairly to all patient groups, and not risk diagnostic inequity.

Using this tool for quantitative fibrosis predictions may cause patient anxiety if results are miscommunicated.

Limitations and Future Work:

(Think about the answer your analysis generated, draw conclusions related to your overarching question, and discuss the ethical implications of your conclusions.)

Pipeline must be reproducible on new independent datasets

Training on further images to perfect pipeline reliability

Find a way to automate the pipeline

NOTES FROM YOUR TEAM:

This is where our team is taking notes and recording activity.

none.

QUESTIONS FOR YOUR TA:

These are questions we have for our TA.

none.