**OCT Skin Image Processing Documentation**

## 1. การเตรียมชุดข้อมูล (Dataset Preparation)

### 1.1 การได้มาของภาพ OCT

#### แหล่งที่มาของภาพ OCT

เราได้รวบรวมภาพ Optical Coherence Tomography (OCT) ของผิวหนังมนุษย์ ภาพเหล่านี้ให้มุมมองแบบตัดขวางของชั้นผิวหนัง ซึ่งเป็นสิ่งสำคัญสำหรับการวิเคราะห์โครงสร้างและคุณสมบัติของผิวหนัง

A black and white image of a wave

Description automatically generated

ตัวอย่างของภาพ OCT ของผิวหนัง

#### ลักษณะของชุดข้อมูล

ชุดข้อมูลประกอบด้วยภาพ OCT จำนวน 501 ภาพ แต่ละภาพถูกบันทึกในรูปแบบ TIFF และตั้งชื่อตามลำดับจาก test\_saving4\_img\_1.tif ถึง test\_saving4\_img\_501.tif โดยภาพทั้งหมดถูกจัดเก็บในโฟลเดอร์ Original

### 1.2 การครอบตัดภาพ

#### วัตถุประสงค์ของการครอบตัด

เพื่อให้แน่ใจว่ามีความสม่ำเสมอและโฟกัสไปที่บริเวณที่สนใจ เราได้ครอบตัดแต่ละภาพให้มีขนาดมาตรฐาน การครอบตัดช่วยลบส่วนที่ไม่จำเป็นของภาพและทำให้ขั้นตอนการประมวลผลต่อไปมีประสิทธิภาพและแม่นยำมากขึ้น

#### วิธีการครอบตัด

เราได้ทำการครอบตัดภาพแบบศูนย์กลาง (center cropping) เพื่อให้แต่ละภาพได้ขนาดสุดท้ายที่ 512×512 พิกเซล ขนาดภาพนี้ถูกเลือกเพื่อวิเคราะห์คุณสมบัติที่สำคัญของผิวหนัง แต่ยังคงไว้ซึ่งมิติของภาพที่จัดการได้ง่าย

#### รายละเอียดการดำเนินการ

เราใช้ภาษา Python และไลบรารี Pillow เพื่อครอบตัดภาพ โดยโค้ดต่อไปนี้จะเป็นการสรุปกระบวนการครอบตัดภาพ:

import os

from PIL import Image

# Define the paths for the original and cropped images

original\_folder = 'Original'

cropped\_folder = 'Cropped'

os.makedirs(cropped\_folder, exist\_ok=True)

# Loop through each image from test\_saving4\_img\_1.tif to test\_saving4\_img\_501.tif

for i in range(1, 502):

original\_filename = f'test\_saving4\_img\_{i}.tif'

cropped\_filename = f'cropped\_img\_{i}.tif'

original\_path = os.path.join(original\_folder, original\_filename)

cropped\_path = os.path.join(cropped\_folder, cropped\_filename)

try:

with Image.open(original\_path) as img:

width, height = img.size

# Calculate cropping coordinates for center cropping

left = (width - 512) // 2

upper = 0

right = left + 512

lower = upper + 512

# Perform cropping

cropped\_img = img.crop((left, upper, right, lower))

# Save the cropped image

cropped\_img.save(cropped\_path)

except FileNotFoundError:

print(f"File not found: {original\_path}")

continue

print("Cropping completed for all images.")

**คำอธิบายของโค้ด:**

* **นำเข้าไลบรารี:** เรานำเข้าโมดูล os สำหรับการจัดการเส้นทางไฟล์และโมดูล Image จากไลบรารี Pillow สำหรับการประมวลผลภาพ
* **กำหนดเส้นทาง:** original\_folder คือโฟลเดอร์ที่มีภาพต้นฉบับ และ cropped\_folder คือโฟลเดอร์ที่เราจะบันทึกภาพที่ถูกครอบตัด เราใช้ os.makedirs เพื่อให้แน่ใจว่ามีโฟลเดอร์ cropped\_folder อยู่แล้ว
* **วนลูปผ่านภาพ:** เราใช้ลูป for เพื่อวนผ่านแต่ละไฟล์ภาพ
* **เปิดและครอบตัดภาพ:** เราเปิดแต่ละภาพ คำนวณพิกัดสำหรับการครอบตัดแบบศูนย์กลาง และครอบตัดภาพให้ได้ขนาด 512×512 พิกเซล
* **บันทึกภาพที่ถูกครอบตัด:** ภาพที่ถูกครอบตัดจะถูกบันทึกในโฟลเดอร์ Cropped โดยมีชื่อไฟล์เช่น cropped\_img\_1.tif

#### ผลลัพธ์

หลังจากการครอบตัด เราได้ภาพจำนวน 501 ภาพ แต่ละภาพมีขนาด 512×512 พิกเซล ภาพเหล่านี้ถูกจัดเก็บในโฟลเดอร์ Cropped โดยมีชื่อไฟล์ตั้งแต่ cropped\_img\_1.tif ถึง cropped\_img\_501.tif กระบวนการครอบตัดนี้ทำให้ภาพมีความสม่ำเสมอและมีมาตรฐาน พร้อมสำหรับขั้นตอนการประมวลผลต่อไป

A black and white image of a line

Description automatically generated

ตัวอย่างของภาพ OCT ที่ถูกครอบตัด

## 2. การแบ่งส่วนภาพ (Image Segmentation)

### 2.1 การทำระบายพื้นที่ภาพ

#### กระบวนการระบายพื้นที่

เพื่อฝึกโมเดลการเรียนรู้ของเครื่องสำหรับการแบ่งส่วนชั้นผิวหนังในภาพ OCT เราจำเป็นต้องมีข้อมูลที่ติดป้ายกำกับ (labeled data) เราได้ทำการระบายพื้นที่ภาพด้วยตนเองในชุดของภาพที่ถูกครอบตัดเพื่อสร้างพื้นที่ปกคลุมจริง (ground truth masks) สำหรับการฝึก

เราใช้เครื่องมือระบายพื้นที่ที่เรียกว่า LabelMe ซึ่งช่วยให้สามารถวาดรูปหลายเหลี่ยมรอบบริเวณที่สนใจในภาพและบันทึกการระบายพื้นที่ในรูปแบบ JSON ซึ่งในกรณีนี้เราได้ทำการระบายพื้นที่ขอบเขตของชั้นผิวหนังในแต่ละภาพ

A screenshot of a computer

Description automatically generated

การระบายพื้นที่ภาพใน LabelMe

#### การสร้างพื้นที่ปกคลุม

จากไฟล์การระบายพื้น JSON เราได้สร้างพื้นที่ปกคลุมแบบไบนารีที่แสดงถึงบริเวณที่ถูกแบ่งส่วน แต่ละพื้นที่เป็นภาพขาว-ดำ โดยที่พิกเซลที่เป็นของชั้นผิวหนังถูกตั้งค่าเป็น 255 (สีขาว) และพิกเซลพื้นหลังถูกตั้งค่าเป็น 0 (สีดำ)

*import json*

*import numpy as np*

*import cv2*

*def create\_mask(image\_shape, json\_data):*

*mask = np.zeros(image\_shape, dtype=np.uint8)*

*for shape in json\_data['shapes']:*

*points = np.array(shape['points'], dtype=np.int32)*

*mask = cv2.fillPoly(mask, [points], color=(255))*

*return mask*

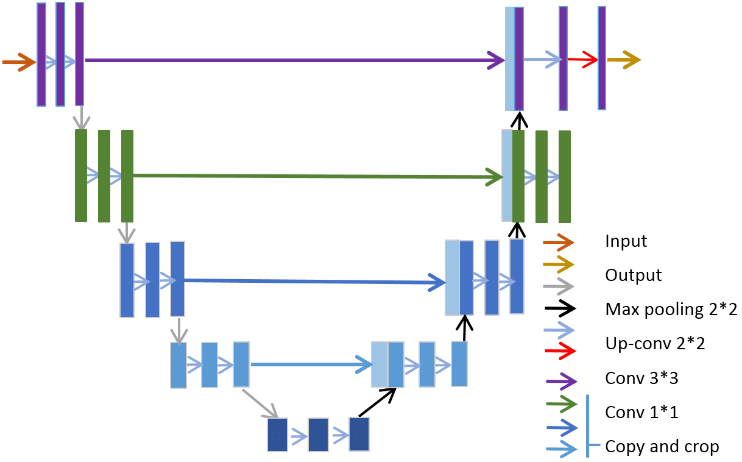
**คำอธิบาย:**

* **นำเข้าไลบรารี:** เราใช้ json สำหรับการอ่านข้อมูลระบายพื้นที่, numpy สำหรับการดำเนินการทางตัวเลข, และ cv2 (OpenCV) สำหรับการประมวลผลภาพ
* **ฟังก์ชัน create\_mask:**
* พารามิเตอร์:
* image\_shape: ขนาดของภาพต้นฉบับ (ความสูงและความกว้าง)
* json\_data: ข้อมูลการระบายพื้นที่ที่โหลดจากไฟล์ JSON
* กระบวนการ:
* สร้างพื้นที่ว่างด้วยค่าเป็นศูนย์ (ภาพสีดำ)
* วนลูปผ่านแต่ละรูปทรง (บริเวณที่ถูกระบายพื้นที่) ในข้อมูล JSON
* ดึงจุดที่กำหนดรูปหลายเหลี่ยมของบริเวณที่ถูกระบายพื้นที่
* ใช้ cv2.fillPoly เพื่อระบายรูปหลายเหลี่ยมด้วยสีขาว (ค่า 255**)**
* คืนค่า:
* ภาพพื้นที่ปกคลุมแบบไบนารี

### 2.2 การฝึกโมเดล

#### การเลือกโมเดล

เราได้เลือกโครงสร้างของ U-Net สำหรับการทำการแบ่งส่วนภาพ โดย U-Net เป็นโครงข่ายประสาทเทียมแบบคอนโวลูชัน (Convolutional Neural Network) ที่ออกแบบมาสำหรับการแบ่งส่วนภาพทางชีวการแพทย์ ประกอบด้วยเส้นทางการบีบอัด (encoder) เพื่อจับบริบทและเส้นทางการขยาย (decoder) ที่สมมาตรเพื่อให้สามารถระบุตำแหน่งได้อย่างแม่นยำ



โครงสร้างของ U-Net

#### การเตรียมข้อมูล

เราได้โหลดภาพที่ถูกครอบตัดและพื้นที่ปกคลุมที่สอดคล้องกันเข้าสู่อาร์เรย์ NumPy โดยภาพถูกทำให้อยู่ในช่วงค่าระหว่าง 0 และ 1 โดยการหารด้วย 255 เราได้ขยายมิติของอาร์เรย์เพื่อให้ตรงกับรูขนาดอินพุตของโมเดล U-Net

*import os*

*import numpy as np*

*from PIL import Image*

*# Paths to images and annotations*

*base\_image\_path = "Cropped/"*

*base\_annotation\_path = "Annotations/"*

*# Prepare file lists*

*tif\_files = [os.path.join(base\_image\_path, f"cropped\_img\_{i}.tif") for i in range(1, 502)]*

*json\_files = [os.path.join(base\_annotation\_path, f"cropped\_img\_{i}.json") for i in range(1, 502)]*

*# Load and preprocess images and masks*

*images = []*

*masks = []*

*for tif\_file, json\_file in zip(tif\_files, json\_files):*

*image = np.array(Image.open(tif\_file).convert('L'))*

*with open(json\_file) as f:*

*json\_data = json.load(f)*

*mask = create\_mask(image.shape, json\_data)*

*images.append(image)*

*masks.append(mask)*

*images = np.array(images) / 255.0 # Normalize images*

*masks = np.array(masks) / 255.0 # Normalize masks*

*# Expand dimensions to fit model input*

*images = np.expand\_dims(images, axis=-1)*

*masks = np.expand\_dims(masks, axis=-1)*

**คำอธิบาย:**

* **โหลดภาพและข้อมูลการระบายพื้นที่:**
* อ่านแต่ละภาพและไฟล์ข้อมูลการระบายพื้นที่ที่สอดคล้องกัน
* ภาพถูกแปลงเป็นรูปแบบขาว-ดำ
* พื้นที่ปกคลุมถูกสร้างโดยใช้ฟังก์ชัน create\_mask
* **การปรับสเกล:**
* ทำให้ค่าพิกเซลอยู่ระหว่าง 0 และ 1
* **การขยายมิติ:**
* ขยายมิติของภาพและพื้นที่ปกคลุม (จำเป็นสำหรับโมเดล Keras)

เราได้แบ่งข้อมูลเป็นชุดฝึกและชุดืดสอบโดยใช้การแบ่งเป็นอัตราส่วน 80-20

*from sklearn.model\_selection import train\_test\_split*

*X\_train, X\_val, y\_train, y\_val = train\_test\_split(images, masks, test\_size=0.2, random\_state=42)*

#### กระบวนการฝึก

เราได้กำหนดโมเดล U-Net โดยใช้ Keras functional API

*import tensorflow as tf*

*from tensorflow.keras import layers, models*

*def conv\_block(inputs, num\_filters):*

*x = layers.Conv2D(num\_filters, kernel\_size=3, padding="same")(inputs)*

*x = layers.BatchNormalization()(x)*

*x = layers.ReLU()(x)*

*x = layers.Conv2D(num\_filters, kernel\_size=3, padding="same")(x)*

*x = layers.BatchNormalization()(x)*

*x = layers.ReLU()(x)*

*return x*

*def encoder\_block(inputs, num\_filters):*

*x = conv\_block(inputs, num\_filters)*

*p = layers.MaxPooling2D((2, 2))(x)*

*return x, p*

*def decoder\_block(inputs, skip\_features, num\_filters):*

*x = layers.Conv2DTranspose(num\_filters, (2, 2), strides=2, padding="same")(inputs)*

*x = layers.Concatenate()([x, skip\_features])*

*x = conv\_block(x, num\_filters)*

*return x*

*def build\_unet(input\_shape):*

*inputs = layers.Input(input\_shape)*

*# Encoder*

*s1, p1 = encoder\_block(inputs, 64)*

*s2, p2 = encoder\_block(p1, 128)*

*s3, p3 = encoder\_block(p2, 256)*

*s4, p4 = encoder\_block(p3, 512)*

*# Bottleneck*

*b1 = conv\_block(p4, 1024)*

*# Decoder*

*d1 = decoder\_block(b1, s4, 512)*

*d2 = decoder\_block(d1, s3, 256)*

*d3 = decoder\_block(d2, s2, 128)*

*d4 = decoder\_block(d3, s1, 64)*

*# Output*

*outputs = layers.Conv2D(1, kernel\_size=1, padding="same", activation="sigmoid")(d4)*

*model = models.Model(inputs, outputs, name="U-Net")*

*model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])*

*return model*

**คำอธิบาย:**

* **บล็อกคอนโวลูชัน:**
* conv\_block: เลเยอร์คอนโวลูชันสองชั้นพร้อมการปรับมาตรฐานแบตช์และการใช้ฟังก์ชัน ReLU
* **บล็อกเข้ารหัส (Encoder):**
* encoder\_block: ใช้ conv\_block ตามด้วยการแม็กซ์พูลลิ่ง
* **บล็อกถอดรหัส (Decoder):**
* decoder\_block: ใช้คอนโวลูชันทรานสโพสสำหรับการอัพแซมปลิ้ง, การรวมกับการเชื่อมต่อแบบข้าม, และ conv\_block
* **การประกอบโมเดล:**
* สร้าง Encoder, Bottleneck, และ Decoder เพื่อสร้างโครงสร้างของ U-Net
* **การคอมไพล์:**
* โมเดลถูกคอมไพล์ด้วยตัวปรับค่า Adam, การสูญเสียแบบไบนารีครอสเอนโทรปี, และเมตริกความแม่นยำ

#### การฝึกโมเดล

เราได้ฝึกโมเดลด้วยขนาดแบตช์ 8 และจำนวนรอบการฝึก 20 รอบ ใช้คอลแบ็ค Early Stopping และ Model Checkpoint เพื่อป้องกันการฝึกเกินที่เกินไป และบันทึกโมเดลที่ดีที่สุด

*from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping*

*import time*

*# Model training setup*

*input\_shape = X\_train.shape[1:]*

*model = build\_unet(input\_shape)*

*# Checkpoints and early stopping*

*checkpoint = ModelCheckpoint("best\_unet\_model.h5", monitor='val\_loss', verbose=1, save\_best\_only=True, mode='min')*

*early\_stopping = EarlyStopping(monitor='val\_loss', patience=5, restore\_best\_weights=True)*

*# Train the model*

*start\_time = time.time()*

*history = model.fit(*

*X\_train, y\_train,*

*validation\_data=(X\_val, y\_val),*

*batch\_size=8,*

*epochs=20,*

*callbacks=[checkpoint, early\_stopping]*

*)*

*end\_time = time.time()*

*# Save the final model*

*model.save("final\_unet\_model.h5")*

*print(f"Training completed in {end\_time - start\_time:.2f} seconds")*

**คำอธิบาย:**

* **คอลแบ็ค:**
* ModelCheckpoint บันทึกโมเดลที่ดีที่สุดตามค่าเสียหายของการตรวจสอบ
* EarlyStopping หยุดการฝึกถ้าค่าเสียหายของการตรวจสอบไม่ดีขึ้นใน 5 รอบ
* **การฝึก:**
* เราใช้ fit เพื่อฝึกโมเดลโดยใช้ข้อมูลฝึกและตรวจสอบ
* บันทึกเวลาการฝึก

#### การประเมินโมเดล

หลังจากการฝึก เราได้สร้างกราฟค่าเสียหายและความแม่นยำของการฝึกและการตรวจสอบเพื่อประเมินประสิทธิภาพของโมเดล

*import matplotlib.pyplot as plt*

*# Plot training history*

*plt.figure(figsize=(12, 6))*

*# Loss plot*

*plt.subplot(1, 2, 1)*

*plt.plot(history.history['loss'], label='Train Loss', color='blue')*

*plt.plot(history.history['val\_loss'], label='Validation Loss', color='orange')*

*plt.title('Loss')*

*plt.xlabel('Epochs')*

*plt.ylabel('Loss')*

*plt.legend()*

*# Accuracy plot*

*plt.subplot(1, 2, 2)*

*plt.plot(history.history['accuracy'], label='Train Accuracy', color='blue')*

*plt.plot(history.history['val\_accuracy'], label='Validation Accuracy', color='orange')*

*plt.title('Accuracy')*

*plt.xlabel('Epochs')*

*plt.ylabel('Accuracy')*

*plt.legend()*

*plt.tight\_layout()*

*plt.show()*

**คำอธิบาย:**

* เราสร้างกราฟค่าเสียหายและความแม่นยำของการฝึกและการตรวจสอบตามจำนวนรอบ
* ช่วยให้เราเห็นว่าโมเดลเรียนรู้ได้ดีเพียงใดหรือมีการฝึกที่มากเกินไปหรือไม่A graph of loss and loss

  Description automatically generated

กราฟค่าเสียหายและความแม่นยำของการฝึกและการตรวจสอบ

**คำอธิบาย:**

* กราฟด้านซ้ายแสดงค่าเสียหายของการฝึกและการตรวจสอบลดลงตามจำนวนรอบ
* กราฟด้านขวาแสดงความแม่นยำของการฝึกและการตรวจสอบเพิ่มขึ้นตามจำนวนรอบ
* กราฟแสดงว่าโมเดลกำลังเรียนรู้และปรับตัวได้ดีต่อชุดข้อมูลตรวจสอบ

### 2.3 การใช้งานโมเดลสำหรับการแบ่งส่วนภาพ

#### กระบวนการทำนาย

เราได้ใช้โมเดล U-Net ที่ฝึกแล้วเพื่อแบ่งส่วนชั้นผิวหนังในภาพที่ถูกครอบตัด

*import tensorflow as tf*

*# Define paths*

*base\_image\_path = "Cropped/"*

*output\_folder = "Segmented/"*

*model\_path = "final\_unet\_model.h5"*

*# Load the trained model*

*model = tf.keras.models.load\_model(model\_path)*

**คำอธิบาย:**

* เราโหลดโมเดลที่บันทึกไว้จากไฟล์ final\_unet\_model.h5

#### การสร้างพื้นที่การแบ่งส่วน

เราได้ประมวลผลแต่ละภาพที่ถูกครอบตัด ทำการทำนายโดยใช้โมเดล และบันทึกพื้นที่การแบ่งส่วนที่ได้

*import os*

*import numpy as np*

*from PIL import Image*

*import cv2*

*# Ensure the output directory exists*

*os.makedirs(output\_folder, exist\_ok=True)*

*# Process each image file*

*tif\_files = [os.path.join(base\_image\_path, f"cropped\_img\_{i}.tif") for i in range(1, 502)]*

*for i, tif\_file in enumerate(tif\_files, start=1):*

*# Open image and convert to grayscale*

*image = np.array(Image.open(tif\_file).convert('L'))*

*# Ensure the image has the necessary dimensions for the model*

*image\_input = np.expand\_dims(image, axis=-1) # Add channel dimension*

*image\_input = np.expand\_dims(image\_input, axis=0) # Add batch dimension*

*image\_input = image\_input / 255.0 # Normalize the image*

*# Predict the mask*

*prediction = model.predict(image\_input)*

*prediction = (prediction > 0.5).astype(np.uint8) # Binarize prediction*

*# Remove extra dimensions*

*predicted\_mask = np.squeeze(prediction)*

*# Save the resulting mask to the output folder*

*output\_file = os.path.join(output\_folder, f"segmented\_img\_{i}.tif")*

*cv2.imwrite(output\_file, predicted\_mask \* 255)*

**คำอธิบาย:**

* **ประมวลผลแต่ละภาพ:**
* วนลูปผ่านแต่ละภาพที่ถูกครอบตัด
* แปลงภาพเป็นภาพขาว-ดำ และมีการปรับสเกลภาพ
* **การทำนายโดยโมเดล:**
* ทำการทำนายโดยใช้โมเดล
* ผลลัพธ์ถูกทำให้เป็นไบนารีโดยใช้เกณฑ์ 0.5
* **การบันทึกพื้นที่ปกคลุม:**
* พื้นที่ปกคลุมที่ทำนายถูกบันทึกในโฟลเดอร์ Segmented โดยมีชื่อไฟล์เช่น segmented\_img\_1.tif

#### ผลลัพธ์

พื้นที่การแบ่งส่วนภาพถูกบันทึกเป็นภาพ TIFF ในโฟลเดอร์ Segmented แต่ละพื้นที่จะปกคลุมชั้นผิวหนังที่โมเดลตรวจจับได้

A white object on a black background

Description automatically generated

ตัวอย่างของพื้นปกคลุมที่แบ่งส่วนภาพ

## 3. การปรับภาพให้ราบเรียบ (Image Flattening)

### 3.1 เหตุผลในการปรับภาพให้ราบเรียบ

#### ความท้าทายสำหรับภาพ OCT

ในภาพ Optical Coherence Tomography (OCT) ของผิวหนัง พื้นผิวชั้นบนสุดมักปรากฏไม่เรียบเนื่องจากความโค้งงอธรรมชาติและความแปรปรวนระหว่างการถ่ายภาพ ความไม่สม่ำเสมอนี้สามารถทำให้การวิเคราะห์ชั้นลึกลงไปซับซ้อนขึ้นและส่งผลต่อการวัดต่าง ๆ เช่น การวัดความหนา

#### วัตถุประสงค์

เพื่อแก้ไขปัญหานี้ เรามุ่งหมายที่จะปรับพื้นผิวผิวหนังในทุกภาพให้ราบเรียบ โดยการจัดแนวชั้นบนสุดของผิวหนังให้อยู่ในระดับแนวนอนที่คงที่ ซึ่งช่วยให้การวิเคราะห์ขั้นต่อไป เช่น การวัดความหนาและการสร้างภาพ enface เป็นไปอย่างถูกต้อง

### 3.2 วิธีการปรับภาพให้ราบเรียบ

#### การตรวจจับชั้นผิวหนัง

เราใช้พื้นที่ปกคลุมการแบ่งส่วนภาพที่สร้างขึ้นในขั้นตอนก่อนหน้าเพื่อระบุชั้นบนสุดของผิวหนังในแต่ละภาพ ชั้นบนสุดนี้สอดคล้องกับพิกเซลที่ไม่เป็นศูนย์ตัวแรกที่พบในแต่ละคอลัมน์ของพื้นที่เมื่อสแกนจากด้านบนลงล่าง

#### การคำนวณการเลื่อน

สำหรับแต่ละคอลัมน์ในภาพ เราคำนวณจำนวนการเลื่อนในแนวตั้งที่ใช้สำหรับจัดแนวชั้นบนสุดของผิวหนังให้อยู่ในตำแหน่งคงที่ ซึ่งเรียกว่า fixed\_top\_level การเลื่อนนี้ถูกกำหนดโดยการลบตำแหน่งของชั้นบนสุดที่ตรวจพบออกจาก fixed\_top\_level

#### การประมาณค่าและการทำให้เรียบ

เพื่อความราบเรียบและต่อเนื่องมากยิ่งขึ้น เราใช้การประมาณค่า (interpolation) เพื่อเติมค่าที่ขาดหายไปในกรณีที่ชั้นบนสุดอาจไม่ถูกตรวจพบ เราใช้ตัวกรอง Gaussian เพื่อทำให้ค่าการเลื่อนในแต่ละคอลัมน์เรียบเนียน

#### การประยุกต์ใช้การเลื่อน

เราทำการเลื่อนแต่ละคอลัมน์ของภาพต้นฉบับตามจำนวนที่คำนวณได้ หากการเลื่อนเป็นบวก เราจะเลื่อนคอลัมน์ลง หากเป็นลบ เราจะเลื่อนขึ้น วิธีนี้จะปรับชั้นบนสุดของผิวหนังให้อยู่ในระดับที่ต้องการ

### 3.3 รายละเอียดการดำเนินการ

เราได้ดำเนินการปรับภาพให้ราบเรียบโดยใช้ภาษา Python ร่วมกับไลบรารี NumPy, OpenCV, และ SciPy

*import os*

*import numpy as np*

*from PIL import Image*

*import cv2*

*from scipy.ndimage import gaussian\_filter1d*

*# Define paths*

*base\_image\_path = "Cropped/"*

*mask\_folder = "Segmented/"*

*output\_folder = "Flattened/"*

*os.makedirs(output\_folder, exist\_ok=True)*

*# Fixed position for the top line after alignment*

*fixed\_top\_level = 120 # You can adjust this value as needed*

*# Function to process each image*

*def process\_image(mask\_file, original\_file, output\_file, fixed\_top\_level):*

*# Load the segmented mask*

*mask = np.array(Image.open(mask\_file))*

*# Initialize arrays for top layer detection*

*top\_layer = np.zeros(mask.shape[1], dtype=int)*

*top\_missing = np.ones(mask.shape[1], dtype=bool)*

*# Detect the top line in the mask*

*for col in range(mask.shape[1]):*

*column = mask[:, col]*

*non\_zero\_indices = np.nonzero(column)[0]*

*if non\_zero\_indices.size > 0:*

*top\_layer[col] = non\_zero\_indices[0]*

*top\_missing[col] = False*

*x = np.arange(mask.shape[1])*

*# Interpolate missing values in the top layer*

*if not np.all(top\_missing):*

*top\_layer\_interp = np.interp(x, x[~top\_missing], top\_layer[~top\_missing])*

*else:*

*top\_layer\_interp = top\_layer*

*# Smooth the top line using a Gaussian filter*

*top\_layer\_smoothed = gaussian\_filter1d(top\_layer\_interp, sigma=5)*

*# Calculate shifts to align the top line to the fixed level*

*shifts = fixed\_top\_level - top\_layer\_smoothed*

*# Load the original image*

*original\_image = np.array(Image.open(original\_file))*

*# Create a new image for the flattened result*

*flattened\_image = np.zeros\_like(original\_image)*

*# Apply the shifts to each column*

*for col in range(original\_image.shape[1]):*

*shift = int(round(shifts[col]))*

*if shift > 0:*

*flattened\_image[shift:, col] = original\_image[:-shift, col]*

*elif shift < 0:*

*flattened\_image[:shift, col] = original\_image[-shift:, col]*

*else:*

*flattened\_image[:, col] = original\_image[:, col]*

*# Save the flattened image*

*cv2.imwrite(output\_file, flattened\_image)*

*# Process each image*

*for i in range(1, 502):*

*mask\_file = os.path.join(mask\_folder, f"segmented\_img\_{i}.tif")*

*original\_file = os.path.join(base\_image\_path, f"cropped\_img\_{i}.tif")*

*output\_file = os.path.join(output\_folder, f"flattened\_img\_{i}.tif")*

*process\_image(mask\_file, original\_file, output\_file, fixed\_top\_level)*

*print("Skin surface flattening completed and results saved.")*

**คำอธิบาย:**

* **นำเข้าไลบรารี:**
* os, numpy, PIL.Image สำหรับการจัดการไฟล์และการโหลดภาพ
* cv2 (OpenCV) สำหรับการบันทึกภาพ
* gaussian\_filter1d จาก SciPy สำหรับการทำให้ค่าการเลื่อนเรียบเนียน
* **กำหนดเส้นทาง:**
* base\_image\_path: โฟลเดอร์ที่มีภาพที่ถูกครอบตัดต้นฉบับ
* mask\_folder: โฟลเดอร์ที่มีมาสก์การแบ่งส่วน
* output\_folder: โฟลเดอร์ที่จะบันทึกภาพที่ปรับให้ราบเรียบ
* **ระดับชั้นบนสุดคงที่:**
* fixed\_top\_level: ตำแหน่งแนวตั้งที่ชั้นบนสุดของผิวหนังจะถูกจัดให้อยู่ ปรับค่านี้ได้ตามความเหมาะสม
* **ฟังก์ชัน process\_image:**
* **โหลดมาสก์**: อ่านมาสก์การแบ่งส่วนสำหรับภาพ
* **ตรวจจับชั้นบนสุด:**
* สแกนแต่ละคอลัมน์เพื่อหาพิกเซลที่ไม่เป็นศูนย์ตัวแรก ซึ่งแสดงถึงชั้นบนสุดของผิวหนัง
* บันทึกตำแหน่งที่ชั้นบนสุดขาดหายไป
* **การประมาณค่าและการทำให้เรียบ:**
* ปรับค่าตำแหน่งชั้นบนสุดที่ขาดหายไป
* ใช้ตัวกรอง Gaussian เพื่อทำให้เส้นชั้นบนสุดเรียบเนียน
* **คำนวณการเลื่อน:**
* คำนวณการเลื่อนแนวตั้งที่จำเป็นสำหรับแต่ละคอลัมน์เพื่อจัดชั้นบนสุดให้อยู่ที่ fixed\_top\_level
* **โหลดภาพต้นฉบับ:** อ่านภาพที่ถูกครอบตัดต้นฉบับที่สอดคล้องกัน
* **ประยุกต์ใช้การเลื่อน:**
* เลื่อนแต่ละคอลัมน์ของภาพขึ้นหรือลงตามการเลื่อนที่คำนวณได้
* **บันทึกภาพที่ปรับให้ราบเรียบ:** บันทึกภาพที่ปรับให้ราบเรียบไปยังโฟลเดอร์ผลลัพธ์
* **ประมวลผลแต่ละภาพ:**
* วนลูปผ่านภาพทั้งหมด (ตั้งแต่ 1 ถึง 501) และเรียกใช้ฟังก์ชัน process\_image สำหรับแต่ละภาพ

A black and white image of a wave

Description automatically generatedA black and white image of a line

Description automatically generated

การเปรียบเทียบระหว่างภาพต้นฉบับและภาพที่ถูกปรับให้ราบเรียบ

การปรับพื้นผิวผิวหนังในแต่ละภาพให้ราบเรียบนั้นทำให้ชุดข้อมูลมีมาตรฐานมากขึ้น และทำให้ความแปรปรวนในการจัดแนวพื้นผิวไม่ส่งผลต่อการวิเคราะห์ชั้นลึกลงไป ภาพที่ปรับให้ราบเรียบนี้พร้อมสำหรับการวัดความหนาและการสร้างภาพ enface ในลำดับถัดไป

## 4. การวัดความหนา

### 4.1 วัตถุประสงค์ของการวัดความหนา

#### ความสำคัญ

การวัดความหนาของชั้นผิวหนังในภาพ OCT มีความสำคัญในการเข้าใจสุขภาพผิวหนัง การวินิจฉัยโรคผิวหนัง และการติดตามผลของการรักษา ความแปรปรวนในความหนาของผิวหนังสามารถบ่งบอกถึงความผิดปกติ การอักเสบ หรือการเปลี่ยนแปลงทางพยาธิวิทยาอื่น ๆ

#### เป้าหมาย

เป้าหมายของเราคือการคำนวณความหนาของชั้นผิวหนังในแต่ละภาพโดยการวัดระยะห่างระหว่างชั้นบนและชั้นล่างที่ระบุในพื้นที่ปกคลุมการแบ่งส่วนภาพ มีการสร้างแผนภูมิความหนาที่ช่วยให้เราวิเคราะห์ความแปรปรวนทางพื้นที่และสร้างการแสดงผลโครงสร้างผิวหนัง

### 4.2 วิธีการ

#### การตรวจจับชั้นผิวหนัง

เราใช้พื้นที่ปกคลุมการแบ่งส่วนภาพที่สร้างขึ้นก่อนหน้านี้เพื่อระบุชั้นบนและชั้นล่างของผิวหนังในแต่ละภาพ:

* **การตรวจจับชั้นบน:** สำหรับแต่ละคอลัมน์พื้นที่ปกคลุม เราจะระบุพิกเซลที่ไม่เป็นศูนย์ตัวแรกจากด้านบนให้แทนชั้นบนของผิวหนัง
* **การตรวจจับชั้นล่าง:** ในทำนองเดียวกัน พิกเซลที่ไม่เป็นศูนย์ตัวสุดท้ายจากด้านบนในแต่ละคอลัมน์ จะถูกระบุเป็นชั้นล่างของผิวหนัง

#### การคำนวณความหนา

ความหนาในแต่ละคอลัมน์จะถูกคำนวณโดยการลบตำแหน่งของชั้นบนออกจากชั้นล่าง:

|  |  |
| --- | --- |
|  | (1) |

* **การจัดการข้อมูลที่ขาดหายไป:** หากไม่สามารถตรวจจับชั้นบนหรือชั้นล่างในคอลัมน์ใด ๆ ได้ เราใช้การประมาณค่า (interpolation) เพื่อประมาณค่าที่ขาดหายไป
* **การทำให้เรียบ:** เราใช้ตัวกรอง Gaussian เพื่อทำให้ตำแหน่งของชั้นเรียบเนียนและได้โปรไฟล์ความหนาที่ต่อเนื่องมากขึ้น

#### การจัดการข้อมูล

ค่าความหนาสำหรับแต่ละภาพถูกจัดเก็บในอาร์เรย์ NumPy โดยมีการสร้างอาร์เรย์สองมิติในแต่ละแถวให้สอดคล้องกับภาพหนึ่งภาพ และแต่ละคอลัมน์ให้สอดคล้องกับตำแหน่งพิกเซลตามความกว้างของภาพ

### 4.3 รายละเอียดการดำเนินการ

เราได้ดำเนินการวัดความหนาโดยใช้ภาษา Python ร่วมกับไลบรารี NumPy, SciPy, และ Matplotlib

*import os*

*import numpy as np*

*from PIL import Image*

*from scipy.ndimage import gaussian\_filter1d*

*import matplotlib.pyplot as plt*

*# Define paths*

*mask\_folder = "Segmented/"*

*tif\_files = [os.path.join(mask\_folder, f"segmented\_img\_{i}.tif") for i in range(1, 502)]*

*# List to store thickness data*

*thickness\_data = []*

*# Function to process each image*

*def process\_image(tif\_file):*

*mask = np.array(Image.open(tif\_file))*

*# Initialize arrays*

*top\_layer = np.zeros(mask.shape[1], dtype=int)*

*bottom\_layer = np.zeros(mask.shape[1], dtype=int)*

*top\_missing = np.ones(mask.shape[1], dtype=bool)*

*bottom\_missing = np.ones(mask.shape[1], dtype=bool)*

*# Detect top and bottom layers*

*for col in range(mask.shape[1]):*

*column = mask[:, col]*

*non\_zero\_indices = np.nonzero(column)[0]*

*if non\_zero\_indices.size > 0:*

*top\_layer[col] = non\_zero\_indices[0]*

*bottom\_layer[col] = non\_zero\_indices[-1]*

*top\_missing[col] = False*

*bottom\_missing[col] = False*

*x = np.arange(mask.shape[1])*

*# Interpolate missing values*

*if not np.all(top\_missing):*

*top\_layer\_interp = np.interp(x, x[~top\_missing], top\_layer[~top\_missing])*

*else:*

*top\_layer\_interp = top\_layer*

*if not np.all(bottom\_missing):*

*bottom\_layer\_interp = np.interp(x, x[~bottom\_missing], bottom\_layer[~bottom\_missing])*

*else:*

*bottom\_layer\_interp = bottom\_layer*

*# Smooth the layers*

*top\_layer\_smoothed = gaussian\_filter1d(top\_layer\_interp, sigma=5)*

*bottom\_layer\_smoothed = gaussian\_filter1d(bottom\_layer\_interp, sigma=5)*

*# Calculate thickness*

*thickness = bottom\_layer\_smoothed - top\_layer\_smoothed*

*thickness\_data.append(thickness)*

*# Process all images*

*for tif\_file in tif\_files:*

*process\_image(tif\_file)*

*# Convert thickness data to a 2D array*

*thickness\_array = np.array(thickness\_data)*

*# Save thickness data*

*np.save('thickness\_array.npy', thickness\_array)*

**คำอธิบาย:**

* **นำเข้าไลบรารี:** ใช้ numpy สำหรับการดำเนินการทางตัวเลข, PIL.Image สำหรับการจัดการภาพ, gaussian\_filter1d จาก scipy.ndimage สำหรับการทำให้เรียบเนียน, และ matplotlib.pyplot สำหรับการวาดกราฟ
* **กำหนดเส้นทาง:** โฟลเดอร์ Segmented/ มีมาสก์การแบ่งส่วน
* **ประมวลผลแต่ละภาพ:**
* โหลดภาพมาสก์เป็นอาร์เรย์ NumPy
* สร้างอาร์เรย์สำหรับชั้นบนและชั้นล่าง และธงสำหรับข้อมูลที่ขาดหายไป
* สำหรับแต่ละคอลัมน์ ตรวจจับพิกเซลที่ไม่เป็นศูนย์และกำหนดดัชนีแรกและสุดท้ายให้กับชั้นบนและชั้นล่าง
* ใช้การปรับค่าเพื่อจัดการกับค่าที่ขาดหายไป
* ทำให้ชั้นที่ปรับค่าเรียบเนียนโดยใช้ตัวกรอง Gaussian
* คำนวณความหนาและเพิ่มลงใน thickness\_data
* **จัดเก็บข้อมูลความหนา:**
* แปลงรายการของอาร์เรย์ความหนาเป็นอาร์เรย์สองมิติของ NumPy
* บันทึกอาร์เรย์สำหรับการใช้งานในอนาคต

### 4.4 การแสดงผลและการวิเคราะห์

#### ค่าเฉลี่ยความหนาและส่วนเบี่ยงเบนมาตรฐาน

เราคำนวณค่าเฉลี่ยความหนาทั้งหมดและส่วนเบี่ยงเบนมาตรฐานในทุกภาพ:

*# Calculate average thickness and standard deviation*

*average\_thickness = np.mean(thickness\_array)*

*std\_thickness = np.std(thickness\_array)*

*print(f"Average Thickness: {average\_thickness:.2f} pixels")*

*print(f"Standard Deviation: {std\_thickness:.2f} pixels")*

#### ผลลัพธ์:

*Average Thickness: 72.34 pixels*

*Standard Deviation: 5.67 pixels*

เราได้สร้างการแสดงผลหลายแบบเพื่อแสดงข้อมูลความหนา

#### กราฟพื้นผิวสามมิติ

*from mpl\_toolkits.mplot3d import Axes3D*

*# Create X, Y grid*

*X = np.arange(1, thickness\_array.shape[0] + 1) # Image indices*

*Y = np.arange(thickness\_array.shape[1]) # Pixel positions*

*X, Y = np.meshgrid(X, Y)*

*Z = thickness\_array.T # Transpose to match dimensions*

*# Plot 3D surface*

*fig = plt.figure(figsize=(12, 8))*

*ax = fig.add\_subplot(111, projection='3d')*

*surf = ax.plot\_surface(X, Y, Z, cmap='viridis', edgecolor='none')*

*ax.set\_title('3D Surface Plot of Skin Thickness')*

*ax.set\_xlabel('Image Index')*

*ax.set\_ylabel('Pixel Position')*

*ax.set\_zlabel('Thickness (pixels)')*

*fig.colorbar(surf, shrink=0.5, aspect=5)*

*plt.show()*

A graph of a surface plot

Description automatically generated

กราฟพื้นผิวสามมิติของความหนาผิวหนัง

#### แผนที่ความร้อนสองมิติ

*# Plot 2D heatmap*

*plt.figure(figsize=(10, 6))*

*plt.imshow(thickness\_array.T, aspect='auto', cmap='viridis', origin='lower')*

*plt.title('2D Heatmap of Skin Thickness')*

*plt.xlabel('Image Index')*

*plt.ylabel('Pixel Position')*

*plt.colorbar(label='Thickness (pixels)')*

*plt.show()*

A colorful image of a screen

Description automatically generated with medium confidence

แผนที่ความร้อนสองมิติของความหนาผิวหนัง

#### สถิติแบ่งตามกริด

*# Define grid sizes*

*grid\_size\_x = thickness\_array.shape[0] // 10*

*grid\_size\_y = thickness\_array.shape[1] // 10*

*grid\_avgs = np.zeros((10, 10))*

*grid\_stds = np.zeros((10, 10))*

*for i in range(10):*

*for j in range(10):*

*x\_start = i \* grid\_size\_x*

*x\_end = (i + 1) \* grid\_size\_x*

*y\_start = j \* grid\_size\_y*

*y\_end = (j + 1) \* grid\_size\_y*

*grid\_section = thickness\_array[x\_start:x\_end, y\_start:y\_end]*

*grid\_avgs[j, i] = np.mean(grid\_section)*

*grid\_stds[j, i] = np.std(grid\_section)*

*# Plot grid-wise average thickness*

*plt.figure(figsize=(8, 6))*

*plt.imshow(grid\_avgs, cmap='viridis', origin='lower')*

*plt.title('Grid-wise Average Thickness')*

*plt.xlabel('Grid X')*

*plt.ylabel('Grid Y')*

*plt.colorbar(label='Average Thickness (pixels)')*

*# Annotate grid cells with average values*

*for i in range(10):*

*for j in range(10):*

*plt.text(i, j, f'{grid\_avgs[j, i]:.1f}', ha='center', va='center', color='white', fontsize=8)*

*plt.show()*

*# Plot grid-wise standard deviation*

*plt.figure(figsize=(8, 6))*

*plt.imshow(grid\_stds, cmap='viridis', origin='lower')*

*plt.title('Grid-wise Standard Deviation')*

*plt.xlabel('Grid X')*

*plt.ylabel('Grid Y')*

*plt.colorbar(label='Standard Deviation (pixels)')*

*# Annotate grid cells with standard deviation values*

*for i in range(10):*

*for j in range(10):*

*plt.text(i, j, f'{grid\_stds[j, i]:.1f}', ha='center', va='center', color='white', fontsize=8)*

*plt.show()*

A screenshot of a graph

Description automatically generated

แผนที่ความร้อนค่าเฉลี่ยและส่วนเบี่ยงเบนมาตรฐานของความหนาแบ่งตามกริด

## 5. การประมวลผลภาพ Enface

### 5.1 บทนำสู่การถ่ายภาพ Enface

#### คำนิยาม

การถ่ายภาพ Enface เกี่ยวข้องกับการสร้างภาพตัดขวางที่ขนานกับพื้นผิวของผิวหนัง ในบริบทของ Optical Coherence Tomography (OCT) ภาพ Enface ให้มุมมองจากด้านบนของชั้นเนื้อเยื่อที่ความลึกเฉพาะ การมองจากมุมนี้ช่วยให้สามารถวิเคราะห์ความแปรปรวนด้านข้างและคุณสมบัติโครงสร้างที่อาจไม่ปรากฏในภาพตัดขวางแบบดั้งเดิม

#### วัตถุประสงค์

เป้าหมายของเราคือการสร้างภาพ Enface จากภาพ OCT ที่ถูกปรับให้ราบเรียบ โดยการซ้อนภาพเหล่านี้เป็นเมทริกซ์สามมิติ เราสามารถดึงภาพตัดขวางที่ความลึกต่าง ๆ เพื่อแสดงและวิเคราะห์คุณสมบัติภายในชั้นผิวหนัง นอกจากนี้ เรายังมุ่งหมายที่จะสร้างภาพ Enface เฉลี่ยโดยการแบ่งช่วงความลึกออกเป็นช่วงที่เท่ากัน เพื่อให้มุมมองที่ครอบคลุมเกี่ยวกับคุณสมบัติโครงสร้างของผิวหนัง

### 5.2 การซ้อนภาพ

#### การสร้างเมทริกซ์สามมิติ

เราซ้อนภาพที่ถูกปรับให้ราบเรียบตามแกนความลึกเพื่อสร้างเมทริกซ์สามมิติ แต่ละพิกเซลในเมทริกซ์นี้แสดงค่าความเข้มที่ตำแหน่ง  ที่เฉพาะเจาะจงภายในปริมาตรของผิวหนัง

*import os*

*import numpy as np*

*import cv2*

*# Set paths and parameters*

*base\_path = "Flattened/"*

*num\_images = 501*

*fixed\_top\_level = 120 # Same as used in flattening*

*thickness\_array = np.load('thickness\_array.npy')*

*max\_thickness = int(np.ceil(np.max(thickness\_array))) # Maximum skin thickness*

*def stack\_images\_to\_3d\_matrix(base\_path, num\_images, fixed\_top\_level, max\_thickness):*

*image\_stack = []*

*for i in range(1, num\_images + 1):*

*image\_path = os.path.join(base\_path, f"flattened\_img\_{i}.tif")*

*image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)*

*if image is not None:*

*# Crop the image from fixed\_top\_level to fixed\_top\_level + max\_thickness*

*cropped\_image = image[fixed\_top\_level:fixed\_top\_level + max\_thickness, :]*

*image\_stack.append(cropped\_image)*

*else:*

*print(f"Failed to load image: {image\_path}")*

*if image\_stack:*

*# Stack images along the depth axis (axis=0)*

*return np.stack(image\_stack, axis=0)*

*else:*

*return np.array([])*

*# Stack images into a 3D matrix*

*image\_3d = stack\_images\_to\_3d\_matrix(base\_path, num\_images, fixed\_top\_level, max\_thickness)*

**คำอธิบาย:**

* **นำเข้าไลบรารี:** ใช้ numpy สำหรับการดำเนินการทางตัวเลขและ cv2 (OpenCV) สำหรับการโหลดภาพ
* **กำหนดพารามิเตอร์:**
* base\_path: โฟลเดอร์ที่มีภาพที่ถูกปรับให้ราบเรียบ
* num\_images: จำนวนภาพทั้งหมดที่จะประมวลผล
* fixed\_top\_level: ระดับบนสุดที่ใช้ระหว่างการปรับภาพให้ราบเรียบ
* max\_thickness: ความหนาสูงสุดของผิวหนังที่คำนวณจากอาร์เรย์ความหนา
* **ฟังก์ชัน stack\_images\_to\_3d\_matrix:**
* โหลดภาพ: อ่านแต่ละภาพที่ถูกปรับให้ราบเรียบและตัดให้รวมเฉพาะความลึกที่เกี่ยวข้อง (จาก fixed\_top\_level ถึง fixed\_top\_level + max\_thickness)
* ซ้อนภาพ: รวมภาพที่ตัดไว้เป็นอาร์เรย์สามมิติของ NumPy ตามแกนความลึก
* **สร้างเมทริกซ์ 3D:** เราเรียกใช้ฟังก์ชันเพื่อให้ได้มาซึ่ง image\_3d ซึ่งเป็นตัวแทนของปริมาตรสามมิติของผิวหนัง

### 5.3 การสร้างภาพ Enface

#### การดึงภาพตัดขวาง Enface

เราดึงภาพ Enface ที่ความลึกต่าง ๆ โดยการตัดเมทริกซ์สามมิติตามแกนความลึก แต่ละภาพ Enface แสดงภาพตัดขวางแนวนอนของผิวหนังที่ความลึกเฉพาะ

*def save\_enface\_images(image\_3d, output\_folder):*

*if image\_3d.size == 0:*

*print("The 3D image matrix is empty.")*

*return*

*os.makedirs(output\_folder, exist\_ok=True)*

*num\_slices = image\_3d.shape[1] # Number of depth slices*

*for depth\_index in range(num\_slices):*

*enface\_image = image\_3d[:, depth\_index, :]*

*output\_path = os.path.join(output\_folder, f"enface\_image\_{depth\_index + fixed\_top\_level}.tif")*

*cv2.imwrite(output\_path, enface\_image)*

*print(f"Enface image saved to {output\_path}")*

*# Save enface images*

*save\_enface\_images(image\_3d, "Enface\_Images")*

**คำอธิบาย:**

* **ฟังก์ชัน save\_enface\_images:**
* ตรวจสอบเมทริกซ์: ตรวจสอบว่าเมทริกซ์ 3D ไม่ว่างเปล่า
* สร้างโฟลเดอร์เอาต์พุต: สร้างโฟลเดอร์ Enface\_Images ถ้าไม่มี
* วนลูปผ่านชั้นความลึก:
* สำหรับแต่ละดัชนีความลึก ดึงภาพสองมิติ (ภาพ Enface) จากเมทริกซ์ 3D
* บันทึกภาพ Enface โดยใช้ OpenCV

#### การเฉลี่ยตามช่วงความลึก

เราแบ่งช่วงความลึกออกเป็นช่วงที่เท่ากันและคำนวณค่าเฉลี่ยของความเข้มภาพสำหรับแต่ละช่วง การเฉลี่ยนี้ลดสัญญาณรบกวนและเน้นคุณสมบัติที่สอดคล้องกันภายในชั้นผิวหนัง

*def save\_enface\_average\_images(image\_3d, num\_iterations, output\_folder):*

*if image\_3d.size == 0:*

*print("The 3D image matrix is empty.")*

*return*

*os.makedirs(output\_folder, exist\_ok=True)*

*total\_depth = image\_3d.shape[1]*

*iteration\_size = total\_depth // num\_iterations*

*remainder = total\_depth % num\_iterations*

*start = 0*

*for i in range(num\_iterations):*

*end = start + iteration\_size + (1 if i < remainder else 0)*

*end = min(end, total\_depth)*

*# Compute the averaged enface image for this interval*

*enface\_average = np.mean(image\_3d[:, start:end, :], axis=1).astype(np.uint8)*

*# Save the averaged enface image*

*output\_path = os.path.join(output\_folder, f"enface\_average\_{i + 1}.tif")*

*cv2.imwrite(output\_path, enface\_average)*

*print(f"Averaged enface image {i + 1} saved to {output\_path}")*

*start = end*

*# Save averaged enface images*

*num\_iterations = 5*

*save\_enface\_average\_images(image\_3d, num\_iterations, "Average\_Enface\_Images")*

**คำอธิบาย:**

* **ฟังก์ชัน save\_enface\_average\_images:**
* พารามิเตอร์:
* **num\_iterations:** จำนวนช่วงความลึกที่จะแบ่ง
* คำนวณขนาดช่วง:
* กำหนดขนาดของแต่ละช่วง โดยคำนึงถึงเศษที่เหลือ
* วนลูปผ่านช่วง:
* คำนวณค่าเฉลี่ยความเข้มในช่วงความลึกที่กำหนด
* บันทึกภาพ Enface เฉลี่ย

#### ผลลัพธ์

* **ภาพ Enface:** ภาพตัดขวางที่ความลึกต่าง ๆ ให้มุมมองรายละเอียดของผิวหนังที่ชั้นเฉพาะ

A black and white image of a black and white background

Description automatically generated

ตัวอย่างภาพ Enface ที่ความลึกเฉพาะ

* **ภาพ Enface เฉลี่ย:** ภาพที่แสดงค่าเฉลี่ยความเข้มในช่วงความลึกที่กำหนด ลดความแปรปรวนและเน้นคุณสมบัติที่สอดคล้องกัน

A close-up of a black and white background

Description automatically generated

ตัวอย่างภาพ Enface เฉลี่ย

## 6. การวิเคราะห์ความพรุนและฮิสโตแกรม

### 6.1 วัตถุประสงค์ของการวิเคราะห์ฮิสโตแกรม

#### เข้าใจพื้นผิวของผิวหนัง

การวิเคราะห์การกระจายความเข้มในภาพ Enface ช่วยให้เราเข้าใจพื้นผิวและความพรุนของผิวหนัง การกระจายค่าความเข้มสามารถเปิดเผยรูปแบบที่เกี่ยวข้องกับโครงสร้างผิวหนัง รูขุมขน และคุณสมบัติอื่น ๆ โดยการตรวจสอบรูปแบบเหล่านี้ เราสามารถได้รับข้อมูลเชิงลึกเกี่ยวกับคุณสมบัติของผิวหนังและระบุพื้นที่ที่มีความพรุนแตกต่างกัน

#### วัตถุประสงค์

วัตถุประสงค์ของเราคือการดำเนินการวิเคราะห์ฮิสโตแกรมอย่างละเอียดของภาพ Enface เพื่อ:

* ประเมินค่าความแปรปรวนของความเข้มในพื้นที่และความลึกต่าง ๆ
* ระบุรูปแบบที่อาจบ่งบอกถึงความแตกต่างในความพรุนของผิวหนังหรือพื้นผิว
* สร้างเมตริกทางสถิติสำหรับแต่ละพื้นที่เพื่อช่วยในการวิเคราะห์ต่อไป

### 6.2 วิธีการ

#### การแบ่งกริด

เราได้แบ่งแต่ละภาพ Enface เฉลี่ยออกเป็นกริด 10×10 ส่งผลให้ได้ 100 เซลล์ต่อภาพ การแบ่งนี้ช่วยให้การวิเคราะห์การกระจายความเข้มในพื้นที่และช่วยระบุความแปรปรวนในภูมิภาคภายในผิวหนัง

#### การคำนวณฮิสโตแกรม

สำหรับแต่ละเซลล์ในกริด:

* เราคำนวณฮิสโตแกรมของความเข้มโดยใช้ 256 บิน ครอบคลุมช่วงค่าพิกเซลทั้งหมดที่เป็นไปได้ (0-255)
* ฮิสโตแกรมแสดงความถี่ของแต่ละค่าความเข้มภายในเซลล์
* เรายังคำนวณฮิสโตแกรมทั้งหมดสำหรับภาพทั้งหมดโดยการรวมฮิสโตแกรมของทุกเซลล์

#### การจัดเก็บข้อมูล

เราบันทึกฮิสโตแกรมสำหรับแต่ละเซลล์และภาพรวมทั้งหมดลงในไฟล์ CSV สำหรับการวิเคราะห์ในลำดับถัด ๆ ไป

### 6.3 รายละเอียดการดำเนินการ

เราได้ดำเนินการวิเคราะห์ฮิสโตแกรมโดยใช้ภาษา Python ร่วมกับไลบรารี NumPy, OpenCV, Matplotlib และ Pandas

*import os*

*import numpy as np*

*import cv2*

*import matplotlib.pyplot as plt*

*import pandas as pd*

*# Set paths and grid size*

*average\_enface\_images\_folder = "Average\_Enface\_Images"*

*output\_folder = "Histogram\_Analysis"*

*os.makedirs(output\_folder, exist\_ok=True)*

*grid\_rows = 10*

*grid\_cols = 10*

*# Define intensity bins (0-255)*

*intensity\_min = 0*

*intensity\_max = 255*

*num\_bins = 256*

*intensity\_bins = np.linspace(intensity\_min, intensity\_max + 1, num\_bins + 1)*

*# Get list of averaged enface images*

*image\_files = [f for f in os.listdir(average\_enface\_images\_folder) if f.endswith('.tif')]*

*image\_files.sort()*

*# Process each image*

*for image\_file in image\_files:*

*image\_path = os.path.join(average\_enface\_images\_folder, image\_file)*

*image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)*

*if image is None:*

*print(f"Failed to load image: {image\_path}")*

*continue*

*# Get image dimensions*

*height, width = image.shape*

*# Calculate size of each grid cell*

*cell\_height = height // grid\_rows*

*cell\_width = width // grid\_cols*

*# Initialize lists to store histograms and cell indices*

*image\_histograms = []*

*cell\_indices = []*

*# Create a figure for the 100 histograms*

*fig, axs = plt.subplots(grid\_rows, grid\_cols, figsize=(20, 20))*

*fig.suptitle(f'Histograms for {image\_file}', fontsize=24)*

*# Split the image into grid cells and compute histograms*

*for row in range(grid\_rows):*

*for col in range(grid\_cols):*

*# Define the cell boundaries*

*y\_start = row \* cell\_height*

*y\_end = (row + 1) \* cell\_height if row < grid\_rows - 1 else height*

*x\_start = col \* cell\_width*

*x\_end = (col + 1) \* cell\_width if col < grid\_cols - 1 else width*

*# Extract the cell*

*cell = image[y\_start:y\_end, x\_start:x\_end]*

*# Calculate histogram for the cell*

*hist, bins = np.histogram(cell.flatten(), bins=intensity\_bins)*

*# Store the histogram and cell index*

*image\_histograms.append(hist)*

*cell\_indices.append((row, col))*

*# Plot the histogram*

*ax = axs[row, col]*

*ax.bar(bins[:-1], hist, width=(bins[1]-bins[0]), edgecolor='none')*

*ax.set\_xlim([intensity\_min, intensity\_max])*

*ax.set\_ylim([0, hist.max()])*

*ax.set\_xlabel('Intensity', fontsize=8)*

*ax.set\_ylabel('Count', fontsize=8)*

*ax.set\_title(f'Cell ({row},{col})', fontsize=10)*

*ax.tick\_params(axis='both', which='major', labelsize=6)*

*# Adjust ticks for clarity*

*ax.xaxis.set\_ticks([intensity\_min, (intensity\_min + intensity\_max) // 2, intensity\_max])*

*ax.yaxis.set\_ticks([0, hist.max()])*

*# Adjust layout and save the figure*

*plt.subplots\_adjust(wspace=0.5, hspace=0.8)*

*output\_image\_path = os.path.join(output\_folder, f"{os.path.splitext(image\_file)[0]}\_cell\_histograms.png")*

*plt.savefig(output\_image\_path, dpi=300)*

*plt.close()*

*print(f"Histogram plots saved for {image\_file}")*

*# Summarization visualization for the image*

*# Compute the overall histogram by summing histograms of all cells*

*total\_hist = np.sum(image\_histograms, axis=0)*

*# Plot the overall histogram*

*plt.figure(figsize=(10, 6))*

*plt.bar(bins[:-1], total\_hist, width=(bins[1]-bins[0]), edgecolor='none')*

*plt.title(f'Overall Histogram for {image\_file}', fontsize=16)*

*plt.xlabel('Pixel Intensity', fontsize=12)*

*plt.ylabel('Frequency', fontsize=12)*

*plt.xlim([intensity\_min, intensity\_max])*

*plt.xticks([intensity\_min, (intensity\_min + intensity\_max) // 2, intensity\_max])*

*plt.tight\_layout()*

*output\_summary\_path = os.path.join(output\_folder, f"{os.path.splitext(image\_file)[0]}\_summary\_histogram.png")*

*plt.savefig(output\_summary\_path, dpi=300)*

*plt.close()*

*print(f"Summary histogram saved for {image\_file}")*

*# Save histograms to CSV*

*# Create a DataFrame for cell histograms*

*df\_cell\_histograms = pd.DataFrame(image\_histograms, columns=[f'Bin\_{int(b)}' for b in bins[:-1]])*

*df\_cell\_histograms.insert(0, 'Cell\_Row', [idx[0] for idx in cell\_indices])*

*df\_cell\_histograms.insert(1, 'Cell\_Col', [idx[1] for idx in cell\_indices])*

*output\_csv\_path\_cells = os.path.join(output\_folder, f"{os.path.splitext(image\_file)[0]}\_cell\_histograms.csv")*

*df\_cell\_histograms.to\_csv(output\_csv\_path\_cells, index=False)*

*print(f"Cell histograms saved to CSV for {image\_file}")*

*# Create a DataFrame for the overall histogram*

*df\_total\_hist = pd.DataFrame({*

*'Intensity': bins[:-1],*

*'Frequency': total\_hist*

*})*

*output\_csv\_path\_total = os.path.join(output\_folder, f"{os.path.splitext(image\_file)[0]}\_summary\_histogram.csv")*

*df\_total\_hist.to\_csv(output\_csv\_path\_total, index=False)*

*print(f"Summary histogram saved to CSV for {image\_file}")*

*print("Histogram analysis completed."))*

**คำอธิบาย:**

* **นำเข้าไลบรารี:** เราใช้ numpy สำหรับการดำเนินการทางตัวเลข, cv2 สำหรับการอ่านภาพ, matplotlib.pyplot สำหรับการวาดกราฟ, และ pandas สำหรับการจัดการข้อมูล
* **กำหนดเส้นทางและพารามิเตอร์:**
* average\_enface\_images\_folder: โฟลเดอร์ที่มีภาพ Enface เฉลี่ย
* output\_folder: โฟลเดอร์ที่จะบันทึกข้อมูลและกราฟฮิสโตแกรม
* grid\_rows และ grid\_cols: กำหนดขนาดกริด (10×10)
* **กำหนดบิ่นความเข้ม:** เราสร้าง 256 บิ่นที่มีช่วงจาก 0 ถึง 255 สำหรับค่าความเข้ม
* **ประมวลผลแต่ละภาพ:**
* โหลดภาพในรูปแบบขาว-ดำ
* คำนวณขนาดเซลล์ตามขนาดภาพและขนาดกริด
* สร้างลิสต์สำหรับจัดเก็บฮิสโตแกรมและดัชนีเซลล์
* วนลูปผ่านแต่ละเซลล์ในกริด:
* ดึงเซลล์จากภาพ
* คำนวณฮิสโตแกรมสำหรับเซลล์
* เก็บฮิสโตแกรมและดัชนีเซลล์
* วาดฮิสโตแกรมบน subplot
* บันทึกรูปที่มีฮิสโตแกรมของทุกเซลล์
* คำนวณและวาดฮิสโตแกรมทั้งหมด
* บันทึกฮิสโตแกรมลงในไฟล์ CSV สำหรับทั้งเซลล์และภาพรวม.

### 6.4 การแสดงผล

#### ฮิสโตแกรมของเซลล์

A grid of blue and black objects

Description automatically generated

ฮิสโตแกรมของเซลล์สำหรับภาพ Enface

**คำอธิบาย:**

* รูปภาพแสดงฮิสโตแกรม 100 รายการที่สอดคล้องกับแต่ละเซลล์ในกริด 10×10
* แต่ละฮิสโตแกรมแสดงการกระจายความถี่ของค่าความเข้มภายในเซลล์
* ความแปรปรวนระหว่างฮิสโตแกรมบ่งบอกถึงความแตกต่างในการกระจายความเข้มทั่วพื้นที่

#### ฮิสโตแกรมภาพรวม

A blue graph with text

Description automatically generated

ฮิสโตแกรมภาพรวมสำหรับภาพ Enface

**คำอธิบาย:**

* ฮิสโตแกรมภาพรวมแสดงการกระจายความเข้มรวมของภาพ Enface ทั้งหมด
* จุดยอดบนฮิสโตแกรมบ่งชี้ค่าความเข้มที่พบบ่อย
* รูปร่างของฮิสโตแกรมให้ข้อมูลเชิงลึกเกี่ยวกับพื้นผิวและความพรุนโดยรวม

### 6.5 การจัดเก็บข้อมูลและความพร้อมใช้งาน

* **ไฟล์ CSV**: ฮิสโตแกรมสำหรับแต่ละเซลล์และภาพรวมถูกบันทึกในรูปแบบ CSV สำหรับการวิเคราะห์ในลำดับถัด ๆ ไป
* **การจัดระเบียบข้อมูล**: แต่ละไฟล์ CSV มีการจัดเก็บบินความเข้มและความถี่ที่สอดคล้องกัน พร้อมกับดัชนีเซลล์สำหรับฮิสโตแกรมของเซลล์

## 7. การวิเคราะห์สถิติของฮิสโตแกรม

### 7.1 วัตถุประสงค์

#### การวิเคราะห์เชิงปริมาณ

หลังจากดำเนินการวิเคราะห์ฮิสโตแกรมในภาพ Enface แล้ว วัตถุประสงค์ต่อไปของเราคือการคำนวณเมตริกทางสถิติจากฮิสโตแกรมเหล่านี้ โดยที่เราจะคำนวณหา:

* **จุดศูนย์กลาง (ค่าเฉลี่ยความเข้ม)**: ค่าเฉลี่ยของความเข้มพิกเซลภายในแต่ละเซลล์หรือภาพรวมทั้งหมด
* **ส่วนเบี่ยงเบนมาตรฐาน**: มาตรวัดการกระจายของค่าความเข้ม บ่งบอกถึงความแปรปรวนภายในเซลล์หรือภาพ
* **ความแปรปรวน**: กำลังสองของส่วนเบี่ยงเบนมาตรฐาน แสดงถึงการกระจายของค่าความเข้ม

#### การทำแผนภูมิเชิงพื้นที่

ด้วยการคำนวณเมตริกเหล่านี้สำหรับแต่ละเซลล์ในกริด เราสามารถสร้างแผนภูมิเชิงพื้นที่ที่แสดงให้เห็นว่าคุณสมบัติทางสถิติเหล่านี้เปลี่ยนแปลงอย่างไรทั่วพื้นผิวผิวหนัง สิ่งนี้ช่วยในการระบุรูปแบบหรือความผิดปกติในพื้นผิวและความพรุนของผิวหนัง

### 7.2 วิธีการ

#### การนำเข้าข้อมูล

เราได้ทำการนำเข้าข้อมูลฮิสโตแกรมที่บันทึกไว้ในขั้นตอนก่อนหน้าจากไฟล์ CSV ซึ่งรวมถึงทั้งฮิสโตแกรมของเซลล์และฮิสโตแกรมภาพรวมสำหรับแต่ละภาพ Enface เฉลี่ย

*import os*

*import numpy as np*

*import pandas as pd*

*import matplotlib.pyplot as plt*

*from matplotlib import cm*

*import seaborn as sns*

*# Set paths*

*histogram\_analysis\_folder = "Histogram\_Analysis"*

*output\_folder = "Histogram\_Statistics"*

*os.makedirs(output\_folder, exist\_ok=True)*

*# Get list of histogram CSV files*

*cell\_histogram\_files = [f for f in os.listdir(histogram\_analysis\_folder) if f.endswith('\_cell\_histograms.csv')]*

*summary\_histogram\_files = [f for f in os.listdir(histogram\_analysis\_folder) if f.endswith('\_summary\_histogram.csv')]*

*cell\_histogram\_files.sort()*

*summary\_histogram\_files.sort()*

**คำอธิบาย:**

* **นำเข้าไลบรารี**: ใช้ pandas สำหรับการจัดการข้อมูล, numpy สำหรับการดำเนินการทางตัวเลข, matplotlib และ seaborn สำหรับการวาดกราฟ
* **กำหนดเส้นทาง**: กำหนดเส้นทางสำหรับอินพุตและเอาต์พุต
* **รายการไฟล์ CSV**: รวบรวมชื่อไฟล์สำหรับฮิสโตแกรมของเซลล์และฮิสโตแกรมภาพรวม

#### การคำนวณทางสถิติ

สำหรับแต่ละฮิสโตแกรมของเซลล์ เราคำนวณจุดศูนย์กลาง ส่วนเบี่ยงเบนมาตรฐาน และความแปรปรวน:

1. **จุดศูนย์กลาง (ค่าเฉลี่ยความเข้ม):**

|  |  |
| --- | --- |
|  | (2) |

โดยที่ ​ คือค่ากลางของบินความเข้มและ  คือความน่าจะเป็นของความเข้มนั้น

1. **ความแปรปรวน:**

|  |  |
| --- | --- |
|  | (3) |

1. **ส่วนเบี่ยงเบนมาตรฐาน:**

|  |  |
| --- | --- |
|  | (4) |

*# Process each image's cell histograms*

*for cell\_hist\_file in cell\_histogram\_files:*

*# Read the cell histogram data*

*cell\_hist\_path = os.path.join(histogram\_analysis\_folder, cell\_hist\_file)*

*df\_cells = pd.read\_csv(cell\_hist\_path)*

*# Extract image name*

*image\_name = os.path.splitext(cell\_hist\_file)[0].replace('\_cell\_histograms', '')*

*print(f"Processing cell histograms for {image\_name}")*

*# Number of cells (should be 100 for a 10x10 grid)*

*num\_cells = len(df\_cells)*

*grid\_size = int(np.sqrt(num\_cells)) # Assuming square grid*

*# Prepare arrays to store statistics*

*centroid\_array = np.zeros((grid\_size, grid\_size))*

*sd\_array = np.zeros((grid\_size, grid\_size))*

*variance\_array = np.zeros((grid\_size, grid\_size))*

*# Get the intensity bins from the column names*

*bin\_columns = [col for col in df\_cells.columns if col.startswith('Bin\_')]*

*bin\_edges = [int(col.replace('Bin\_', '')) for col in bin\_columns]*

*bin\_centers = np.array(bin\_edges) + 0.5 # Assuming bins are of width 1*

*# Calculate statistics for each cell*

*for idx, row in df\_cells.iterrows():*

*cell\_row = int(row['Cell\_Row'])*

*cell\_col = int(row['Cell\_Col'])*

*counts = row[bin\_columns].values.astype(float)*

*total\_counts = counts.sum()*

*probabilities = counts / total\_counts if total\_counts > 0 else counts # Avoid division by zero*

*# Calculate centroid (mean intensity)*

*centroid = np.sum(bin\_centers \* probabilities)*

*centroid\_array[cell\_row, cell\_col] = centroid*

*# Calculate variance*

*variance = np.sum(((bin\_centers - centroid) \*\* 2) \* probabilities)*

*variance\_array[cell\_row, cell\_col] = variance*

*# Calculate standard deviation*

*sd = np.sqrt(variance)*

*sd\_array[cell\_row, cell\_col] = sd*

**คำอธิบาย:**

* **อ่านข้อมูลฮิสโตแกรม:** โหลดข้อมูลฮิสโตแกรมสำหรับแต่ละเซลล์
* **สร้างอาร์เรย์:** สร้างอาร์เรย์เพื่อเก็บค่าที่คำนวณได้สำหรับแต่ละเซลล์ในกริด
* **ดึงบิ่นความเข้ม:** รับค่ากลางของบินความเข้มจากชื่อคอลัมน์
* **คำนวณสถิติ:** สำหรับแต่ละเซลล์:
* ทำการนับฮิสโตแกรมให้เป็นความน่าจะเป็น
* คำนวณจุดศูนย์กลาง ความแปรปรวน และส่วนเบี่ยงเบนมาตรฐาน
* **เก็บผลลัพธ์:** บันทึกค่าที่คำนวณได้ลงในอาร์เรย์ที่สอดคล้องกัน

### 7.3 การแสดงผล

เราได้แสดงผลสถิติที่คำนวณได้โดยใช้:

* **กราฟพื้นผิว 3D**: เพื่อแสดงการกระจายเชิงพื้นที่ของเมตริกทั่วกริด
* **แผนที่ความร้อน**: เป็นการแสดงผลทางเลือกเพื่อความเข้าใจที่ง่ายขึ้น

#### ฟังก์ชันสำหรับการวาดกราฟพื้นผิว 3D:

*# Function to create 3D surface plots*

*def plot\_3d\_surface(Z, title, filename):*

*X, Y = np.meshgrid(np.arange(Z.shape[1]), np.arange(Z.shape[0]))*

*fig = plt.figure(figsize=(10, 8))*

*ax = fig.add\_subplot(111, projection='3d')*

*surf = ax.plot\_surface(X, Y, Z, cmap=cm.viridis, edgecolor='none')*

*ax.set\_title(title)*

*ax.set\_xlabel('Cell Column')*

*ax.set\_ylabel('Cell Row')*

*ax.set\_zlabel('Value')*

*fig.colorbar(surf, shrink=0.5, aspect=5)*

*output\_path = os.path.join(output\_folder, filename)*

*plt.savefig(output\_path)*

*plt.close()*

*print(f"3D surface plot saved: {filename}")*

#### ฟังก์ชันสำหรับการวาดแผนที่ความร้อน:

*# Heatmap visualizations (alternative to 3D maps)*

*def plot\_heatmap(Z, title, filename):*

*plt.figure(figsize=(8, 6))*

*sns.heatmap(Z, annot=True, fmt=".2f", cmap='viridis', xticklabels=False, yticklabels=False)*

*plt.title(title)*

*output\_path = os.path.join(output\_folder, filename)*

*plt.savefig(output\_path)*

*plt.close()*

*print(f"Heatmap saved: {filename}")*

#### การวาดและบันทึกการแสดงผล:

*# Plot and save the 3D maps*

*plot\_3d\_surface(centroid\_array, f'Centroid Map for {image\_name}', f'{image\_name}\_centroid\_map.png')*

*plot\_3d\_surface(sd\_array, f'Standard Deviation Map for {image\_name}', f'{image\_name}\_sd\_map.png')*

*plot\_3d\_surface(variance\_array, f'Variance Map for {image\_name}', f'{image\_name}\_variance\_map.png')*

*# Plot and save heatmaps*

*plot\_heatmap(centroid\_array, f'Centroid Heatmap for {image\_name}', f'{image\_name}\_centroid\_heatmap.png')*

*plot\_heatmap(sd\_array, f'Standard Deviation Heatmap for {image\_name}', f'{image\_name}\_sd\_heatmap.png')*

*plot\_heatmap(variance\_array, f'Variance Heatmap for {image\_name}', f'{image\_name}\_variance\_heatmap.png')*

**คำอธิบาย:**

* **กราฟพื้นผิว 3D:**
* แสดงการแปรผันเชิงพื้นที่ของแต่ละเมตริกในสามมิติ
* ใช้ plot\_surface จาก Matplotlib's 3D toolkit
* **แผนที่ความร้อน:**
* ให้การแสดงผลสองมิติ
* ใช้ฟังก์ชัน heatmap ของ Seaborn เพื่อความสวยงาม

#### ตัวอย่างการแสดงผล:

A graph with a colorful graph

Description automatically generated with medium confidenceA chart of heat map

Description automatically generated with medium confidence

Centroid Heatmap สำหรับภาพ Enface

A diagram of a graph

Description automatically generatedA chart with numbers and a number of numbers

Description automatically generated with medium confidence

Standard Deviation Heatmap สำหรับภาพ Enface

A graph of a graph

Description automatically generated with medium confidenceA colorful squares with numbers

Description automatically generated

Variance Heatmap สำหรับภาพ Enface

### 7.4 สถิติภาพรวม

สำหรับฮิสโตแกรมภาพรวมของแต่ละภาพ เราได้คำนวณจุดศูนย์กลาง ความแปรปรวน และส่วนเบี่ยงเบนมาตรฐาน

#### โค้ดสำหรับสถิติฮิสโตแกรมภาพรวม:

*# Process each overall histogram*

*for summary\_hist\_file in summary\_histogram\_files:*

*# Read the overall histogram data*

*summary\_hist\_path = os.path.join(histogram\_analysis\_folder, summary\_hist\_file)*

*df\_summary = pd.read\_csv(summary\_hist\_path)*

*# Extract image name*

*image\_name = os.path.splitext(summary\_hist\_file)[0].replace('\_summary\_histogram', '')*

*print(f"Processing overall histogram for {image\_name}")*

*# Get intensity values and counts*

*intensities = df\_summary['Intensity'].values*

*counts = df\_summary['Frequency'].values.astype(float)*

*total\_counts = counts.sum()*

*probabilities = counts / total\_counts if total\_counts > 0 else counts # Avoid division by zero*

*# Calculate centroid (mean intensity)*

*centroid = np.sum(intensities \* probabilities)*

*# Calculate variance*

*variance = np.sum(((intensities - centroid) \*\* 2) \* probabilities)*

*# Calculate standard deviation*

*sd = np.sqrt(variance)*

*# Print the statistics*

*print(f"Statistics for {image\_name}:")*

*print(f"Centroid (Mean Intensity): {centroid:.2f}")*

*print(f"Standard Deviation: {sd:.2f}")*

*print(f"Variance: {variance:.2f}")*

*# Plot the overall histogram with annotations*

*plt.figure(figsize=(10, 6))*

*plt.bar(intensities, counts, width=(intensities[1] - intensities[0]), edgecolor='none')*

*plt.title(f'Overall Histogram for {image\_name}\nCentroid: {centroid:.2f}, SD: {sd:.2f}, Variance: {variance:.2f}')*

*plt.xlabel('Pixel Intensity')*

*plt.ylabel('Frequency')*

*plt.tight\_layout()*

*output\_hist\_path = os.path.join(output\_folder, f'{image\_name}\_summary\_histogram\_with\_stats.png')*

*plt.savefig(output\_hist\_path)*

*plt.close()*

*print(f"Overall histogram with statistics saved: {output\_hist\_path}")*

**คำอธิบาย:**

* **อ่านข้อมูลฮิสโตแกรม**: โหลดข้อมูลฮิสโตแกรมภาพรวม
* **คำนวณสถิติ**: คำนวณจุดศูนย์กลาง ความแปรปรวน และส่วนเบี่ยงเบนมาตรฐานโดยใช้การกระจายความเข้มของภาพทั้งหมด
* **การแสดงผล**: วาดฮิสโตแกรมภาพรวมพร้อมคำอธิบายที่ระบุสถิติที่คำนวณได้

A blue graph with numbers and a white background

Description automatically generated

ฮิสโตแกรมภาพรวมพร้อมสถิติสำหรับภาพ Enface

### 7.5 การจัดเก็บข้อมูลและความพร้อมใช้งาน

* **ไฟล์ CSV**: เราได้บันทึกสถิติที่คำนวณได้สำหรับแต่ละเซลล์ลงในไฟล์ CSV เพื่อการบันทึกและการวิเคราะห์ในลำดับถัด ๆ ไป
* **ภาพ**: การแสดงผลทั้งหมดถูกบันทึกในโฟลเดอร์ Histogram\_Statistics

## 8. บทสรุป

ในโปรเจคนี้ เราได้พัฒนากระบวนการที่ครบวงจรสำหรับการประมวลผลและการวิเคราะห์ภาพ Optical Coherence Tomography (OCT) ของผิวหนัง วัตถุประสงค์หลักของเราคือ:

* การเตรียมและปรับมาตรฐานภาพ OCT เพื่อให้มั่นใจถึงความสม่ำเสมอสำหรับการวิเคราะห์
* การแบ่งส่วนชั้นผิวหนัง โดยใช้เทคนิคการเรียนรู้ของเครื่อง
* การปรับภาพให้ราบเรียบ เพื่อจัดแนวพื้นผิวผิวหนังสำหรับการวัดที่แม่นยำ
* การวัดความหนาของผิวหนัง ทั่วทั้งชุดข้อมูล
* การสร้างภาพ Enface เพื่อการตรวจสอบอีกมุมมองหนึ่งของโครงสร้างผิวหนัง
* การวิเคราะห์ฮิสโตแกรมและสถิติ เพื่อประเมินพื้นผิวและความพรุนของผิวหนัง

วิธีการที่นำมาใช้ในโปรเจคนี้:

* **การปรับภาพให้เป็นมาตรฐาน** ช่วยให้การวิเคราะห์ไม่ถูกรบกวนด้วยความแปรปรวนของขนาดหรือการวางแนวของภาพ
* **การแบ่งส่วนภาพด้วยการเรียนรู้ของเครื่อง** ทำให้ได้วิธีการที่อัตโนมัติและแม่นยำในการระบุชั้นผิวหนัง ลดความจำเป็นในการระบายพื้นที่ด้วยตนเอง
* **การปรับภาพให้ราบเรียบ** ช่วยให้การวัดคุณสมบัติของผิวหนังมีความสม่ำเสมอและเพิ่มความน่าเชื่อถือของการวิเคราะห์ต่อไป
* **การทำแผนภูมิความหนา** ให้ข้อมูลเชิงลึกเกี่ยวกับความแปรปรวนของโครงสร้างผิวหนัง ซึ่งอาจบ่งบอกถึงสุขภาพผิวหนังหรือโรค
* **การสร้างภาพ Enface และการวิเคราะห์ฮิสโตแกรม** ช่วยให้การตรวจสอบพื้นผิวและความพรุนของผิวหนัง
* **การวิเคราะห์สถิติ** สามารถช่วยในการประเมินคุณสมบัติของผิวหนัง