UNet CNN Image Prediction Model

In [3]:
```
#########################################################
## UNet CNN Image Model
## Imports
##
## Author: Matthew Yeseta
#########################################################

import tensorflow as tf
from tensorflow.keras import layers, callbacks
from sklearn.model_selection import train_test_split
import os
import glob
from PIL import Image
import numpy as np

unet_prediction_output_dir = "unet_predicted_images"
```

```
WARNING:tensorflow:From C:\Users\matth\anaconda3\Lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_
softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

In [4]:
```
#########################################################
## UNet CNN Image Model
## Class Name: Unnet_CNN()
##
## Author: Matthew Yeseta
#########################################################
class UNet_CNN:
    def __init__(self, input_shape=(128, 128, 3), output_channels=1):  # Set output_channels=1 for binary masks
        self.input_shape = input_shape
        self.output_channels = output_channels

    def build_model(self):
        inputs = tf.keras.layers.Input(shape=self.input_shape)

        down_stack = [
            layers.Conv2D(64, 3, strides=2, padding='same', activation='relu'),
            layers.BatchNormalization(),
            layers.Conv2D(128, 3, strides=2, padding='same', activation='relu'),
            layers.BatchNormalization(),
            layers.Conv2D(256, 3, strides=2, padding='same', activation='relu'),
```

```
            layers.BatchNormalization(),
        ]

        x = inputs
        skips = []
        for down in down_stack:
            x = down(x)
            skips.append(x)

        skips = reversed(skips[:-1])

        up_stack = [
            layers.Conv2DTranspose(128, 3, strides=2, padding='same', activation='relu'),
            layers.BatchNormalization(),
            layers.Dropout(0.3),
            layers.Conv2DTranspose(64, 3, strides=2, padding='same', activation='relu'),
            layers.BatchNormalization(),
            layers.Dropout(0.3),
        ]

        for up, skip in zip(up_stack, skips):
            x = up(x)
            if x.shape[1] != skip.shape[1] or x.shape[2] != skip.shape[2]:
                skip = tf.image.resize(skip, size=(x.shape[1], x.shape[2]))

            x = tf.concat([x, skip], axis=-1)

        output = layers.Conv2D(1, 1, activation='sigmoid')(x)

        return tf.keras.Model(inputs=inputs, outputs=output)
```

In [5]:

```
#######################################################
## UNet CNN Image Model
## Class Name: tensor_create_imageset()
##
## Author: Matthew Yeseta
#######################################################
def tensor_create_imageset(image_path, masked_image_path):
    images = []
    masks = []

    image_files = glob.glob(os.path.join(image_path, "*.bmp")) + glob.glob(os.path.join(image_path, "*.BMP"))
    mask_files = glob.glob(os.path.join(masked_image_path, "*.bmp")) + glob.glob(os.path.join(masked_image_path, "*.BMP
    image_names = {os.path.basename(f).split('.')[0]: f for f in image_files}
    mask_names = {os.path.basename(f).split('.')[0]: f for f in mask_files}
```

```
    matching_names = image_names.keys() & mask_names.keys()
    print(f"Found {len(matching_names)} matching image-mask pairs.")

    if len(matching_names) == 0:
        raise ValueError("No matching image-mask pairs found.")

    assert len(image_files) == len(mask_files), "Mismatch between number of images and masks."

    for img_file, mask_file in zip(image_files, mask_files):
        img = Image.open(img_file).convert('RGB')
        mask = Image.open(mask_file).convert('L')
        img = img.resize((128, 128))
        mask = mask.resize((128, 128))
        img_array = np.array(img, dtype=np.float32) / 255.0
        mask_array = np.array(mask, dtype=np.float32) / 255.0
        images.append(img_array)
        masks.append(mask_array)

    images = tf.convert_to_tensor(images, dtype=tf.float32)
    masks = tf.convert_to_tensor(masks, dtype=tf.float32)

    tf_set = tf.data.Dataset.from_tensor_slices((images, masks))

    return tf_set
```

In [6]:

```
#######################################################
## UNet CNN Image Model
## Class Name: Trainer()
##
## Author: Matthew Yeseta
#######################################################
class Trainer:
    def __init__(self, model, optimizer=None, loss_fn=None, metrics=None):
        self.model = model
        self.optimizer = optimizer if optimizer else tf.keras.optimizers.Adam()
        self.loss_fn = loss_fn if loss_fn else tf.keras.losses.BinaryCrossentropy()

        self.metrics = metrics if metrics else [
            LossesMetrics.dice_coef,
            LossesMetrics.iou_metric,
            tf.keras.metrics.BinaryAccuracy()
        ]

        self.model.compile(optimizer=self.optimizer, loss=self.loss_fn, metrics=self.metrics)
```

```
    def train(self, train_batches, val_batches, steps_per_epoch, validation_steps, epochs=7):
        history = self.model.fit(
            train_batches,
            validation_data=val_batches,
            steps_per_epoch=steps_per_epoch,
            validation_steps=validation_steps,
            epochs=epochs
        )
        return history

    def evaluate(self, test_batches):
        return self.model.evaluate(test_batches)

    def save_predictions(self, test_batches, output_dir):
        for idx, (images, masks) in enumerate(test_batches):
            predictions = self.model.predict(images)

            for i in range(len(predictions)):
                predicted_mask = np.squeeze(predictions[i])
                original_image = (images[i].numpy() * 255).astype(np.uint8)
                mask = np.squeeze(masks[i].numpy())
                mask = (mask * 255).astype(np.uint8)
                Image.fromarray(original_image).save(f"{output_dir}/image_{idx}_{i}.png")
                Image.fromarray(mask).save(f"{output_dir}/mask_{idx}_{i}.png")
                Image.fromarray((predicted_mask * 255).astype(np.uint8)).save(f"{output_dir}/prediction_{idx}_{i}.png")
```

In [7]:

```
######################################################
## UNet CNN Image Model
## Class Name: predict_images()
##
## Author: Matthew Yeseta
######################################################
class predict_images:
    def __init__(self, image_path, masked_image_path, output_channels=1, number_epochs=1):
        self.image_path = image_path
        self.masked_image_path = masked_image_path
        self.output_channels = output_channels
        self.epochs = number_epochs

    def prepare_data(self):
        imageset = tensor_create_imageset(self.image_path, self.masked_image_path)
        images = [image for image, mask in imageset]
        labels = [mask for image, mask in imageset]

        if len(images) == 0 or len(labels) == 0:
```

```
            raise ValueError("No images or labels found in the dataset.")

        labels_expanded = [tf.expand_dims(mask, axis=-1) if mask.shape.ndims == 2 else mask for mask in labels]
        labels_resized = [tf.image.resize(mask, (64, 64)) for mask in labels_expanded]

        X_trainval, X_test, Y_trainval, Y_test = train_test_split(images, labels_resized, test_size=0.1, random_state=4
        X_train, X_val, Y_train, Y_val = train_test_split(X_trainval, Y_trainval, test_size=0.2, random_state=42)
        train_dataset = tf.data.Dataset.from_tensor_slices((X_train, Y_train))
        train_batches = (
            train_dataset.cache().shuffle(1000).batch(64).repeat().prefetch(buffer_size=tf.data.AUTOTUNE)
        )

        val_dataset = tf.data.Dataset.from_tensor_slices((X_val, Y_val))
        val_batches = val_dataset.batch(64)
        test_dataset = tf.data.Dataset.from_tensor_slices((X_test, Y_test))
        test_batches = test_dataset.batch(64)

        return train_batches, val_batches, test_batches, len(X_train), len(X_val)

    def run(self):
        train_batches, val_batches, test_batches, train_size, val_size = self.prepare_data()
        steps_per_epoch = max(train_size // 64, 1)
        validation_steps = max(val_size // 64, 1)

        unet = UNet_CNN(output_channels=self.output_channels)
        model = unet.build_model()
        trainer = Trainer(model=model)
        history = trainer.train(
            train_batches=train_batches,
            val_batches=val_batches,
            steps_per_epoch=steps_per_epoch,
            validation_steps=validation_steps,
            epochs=self.epochs
        )

        results = trainer.evaluate(test_batches)
        test_loss = results[0] # Loss
        test_dice_coef = results[1]  # Dice coeffienent
        test_iou = results[2]  # IoU
        print(f"Test Loss: {test_loss}, Test Dice Coefficient: {test_dice_coef}")

        os.makedirs(unet_prediction_output_dir, exist_ok=True)
        trainer.save_predictions(test_batches, unet_prediction_output_dir)

        return model, history
```

In [8]:
```
#######################################################
## UNet CNN Image Model
## Class Name: setPath()
##
## Author: Matthew Yeseta
#######################################################
def setPath():
    current_working_directory = os.getcwd()
    print("Current Working Directory:", current_working_directory)

    new_working_directory = 'C:\\Users\\matth\\Documents\\master-degree\\IOT'

    try:
        os.chdir(new_working_directory)
        # print("New Working Directory:", os.getcwd())
    except FileNotFoundError:
        print("The specified directory does not exist.")
    except PermissionError:
        print("Permission denied to change to the specified directory.")

    return str(new_working_directory)


#######################################################
## UNet CNN Image Model
## Class Name: LossesMetrics()
##
## Author: Matthew Yeseta
#######################################################
class LossesMetrics:
    @staticmethod
    def dice_loss(y_true, y_pred):
        numerator = 2 * tf.reduce_sum(y_true * y_pred)
        denominator = tf.reduce_sum(y_true + y_pred)
        return 1 - (numerator + 1) / (denominator + 1)

    @staticmethod
    def combined_loss(y_true, y_pred):
        bce = tf.keras.losses.BinaryCrossentropy()(y_true, y_pred)
        dice = LossesMetrics.dice_loss(y_true, y_pred)
        return bce + dice

    @staticmethod
    def dice_coef(y_true, y_pred, smooth=1):
```

```
        intersection = tf.reduce_sum(y_true * y_pred)
        union = tf.reduce_sum(y_true) + tf.reduce_sum(y_pred)
        return (2. * intersection + smooth) / (union + smooth)

    def iou_metric(y_true, y_pred):
        y_pred = tf.cast(y_pred > 0.5, tf.float32)  # Convert probabilities to binary 0 or 1
        intersection = tf.reduce_sum(y_true * y_pred)
        union = tf.reduce_sum(y_true) + tf.reduce_sum(y_pred) - intersection
        return intersection / (union + tf.keras.backend.epsilon())
```

In [9]:

```
#######################################################
## UNet CNN Image Model
## Class Name: open_images_in_directory()
##
## Author: Matthew Yeseta
#######################################################
def open_images_in_directory(directory_path):
    if not os.path.exists(directory_path):
        print(f"Directory not found: {directory_path}")
        return []

    print(f"Checking for .bmp files in {directory_path}...")
    try:
        bmp_files = [f for f in os.listdir(directory_path) if f.endswith('.bmp')]
        images = []

        if not bmp_files:
            print(f"No .bmp files found in {directory_path}")
        else:
            print(f"Found {len(bmp_files)} .bmp files in {directory_path}: {bmp_files}")
            for bmp_file in bmp_files:
                image_path = os.path.join(directory_path, bmp_file)
                try:
                    img = Image.open(image_path)
                    images.append(img)
                    # print(f"Opened image: {image_path}, size: {img.size}")
                except Exception as e:
                    print(f"Error opening image {bmp_file}: {e}")

        return images
    except PermissionError as e:
        print(f"Permission denied: {e.filename}")
        return []
```

```
#####################################################
## UNet CNN Image Model
## Class Name: load_bmp_images()
##
## Author: Matthew Yeseta
#####################################################
def load_bmp_images(image_directory):
    image_tensors = []

    image_directory = os.path.normpath(image_directory)

    try:
        image_files = glob.glob(os.path.join(image_directory, "*.bmp")) + glob.glob(os.path.join(image_directory, "*.BM

        if not image_files:
            print(f"No BMP files found in directory: {image_directory}")
            return []

        for image_file in image_files:
            img = Image.open(image_file).convert('RGB')
            img_array = np.array(img, dtype=np.float32) / 255.0
            img_tensor = tf.convert_to_tensor(img_array)
            image_tensors.append(img_tensor)

    except FileNotFoundError:
        print(f"Error: The directory {image_directory} does not exist.")
        return []

    return image_tensors
```

In [20]:

```
import os
import random
from PIL import Image, ImageDraw, ImageFont
import matplotlib.pyplot as plt

def display_images_with_labels(output_dir, total_images=12, images_per_page=6, num_pages=5):
    image_files = [f for f in os.listdir(output_dir) if f.endswith('.png')]
    image_files = image_files[:50]
    selected_images = random.sample(image_files, min(30, len(image_files)))
    #selected_images = random.sample(image_files, min(total_images, len(image_files)))

    for page in range(num_pages):
        fig, axs = plt.subplots(6, 2, figsize=(10, 15))

        for idx in range(images_per_page):
```

```
            image_idx = page * images_per_page + idx
            if image_idx >= len(selected_images):
                break

            image_file = selected_images[image_idx]
            image_path = os.path.join(output_dir, image_file)
            image = Image.open(image_path)

            draw = ImageDraw.Draw(image)
            try:
                font = ImageFont.truetype("arial.ttf", 16)
            except IOError:
                font = ImageFont.load_default()

            if image.mode == 'RGB':
                fill_color = (255, 255, 255)
            else:
                fill_color = 255

            label = os.path.basename(image_file)
            draw.text((10, 10), label, font=font, fill=fill_color)
            ax = axs[idx // 2, idx % 2]
            ax.imshow(image)
            ax.axis('off')

        plt.tight_layout()
        plt.show()
```

In [26]:

```
#######################################################
## UNet CNN Image Model
## Main Fuction
##
## Author: Matthew Yeseta
#######################################################
if __name__ == '__main__':
    working_directory = setPath()

    g_truth_path = os.path.join(working_directory, 'data-backup', 'groundtruth', 'image')
    g_truth_masked_mask = os.path.join(working_directory, 'data-backup', 'mask')
    image_files = glob.glob(os.path.join(g_truth_path, "*.bmp")) + glob.glob(os.path.join(g_truth_path, "*.BMP"))
    mask_files = glob.glob(os.path.join(g_truth_masked_mask, "*.bmp")) + glob.glob(os.path.join(g_truth_masked_mask, "*
    image_names = {os.path.basename(f).split('.')[0] for f in image_files}
    mask_names = {os.path.basename(f).split('.')[0] for f in mask_files}

    workflow = predict_images(image_path=g_truth_path,
```

```
                              masked_image_path=g_truth_masked_mask,
                              output_channels=2,
                              number_epochs=40)
    model, history = workflow.run()
```

```
Current Working Directory: C:\Users\matth\Documents\master-degree\IOT
Found 300 matching image-mask pairs.
Epoch 1/40
6/6 [==============================] - 26s 3s/step - loss: 0.4867 - dice_coef: 0.5416 - iou_metric: 0.5143 - binary_acc
uracy: 0.7617 - val_loss: 0.6466 - val_dice_coef: 0.3025 - val_iou_metric: 0.0000e+00 - val_binary_accuracy: 0.7513
Epoch 2/40
6/6 [==============================] - 11s 2s/step - loss: 0.2460 - dice_coef: 0.6834 - iou_metric: 0.7668 - binary_acc
uracy: 0.9192 - val_loss: 0.6135 - val_dice_coef: 0.2939 - val_iou_metric: 0.0000e+00 - val_binary_accuracy: 0.7513
Epoch 3/40
6/6 [==============================] - 10s 2s/step - loss: 0.1916 - dice_coef: 0.7381 - iou_metric: 0.8062 - binary_acc
uracy: 0.9311 - val_loss: 0.5801 - val_dice_coef: 0.2798 - val_iou_metric: 0.0000e+00 - val_binary_accuracy: 0.7513
Epoch 4/40
6/6 [==============================] - 10s 2s/step - loss: 0.1686 - dice_coef: 0.7777 - iou_metric: 0.8236 - binary_acc
uracy: 0.9337 - val_loss: 0.5566 - val_dice_coef: 0.2628 - val_iou_metric: 0.0000e+00 - val_binary_accuracy: 0.7513
Epoch 5/40
6/6 [==============================] - 8s 1s/step - loss: 0.1181 - dice_coef: 0.8220 - iou_metric: 0.8756 - binary_accu
racy: 0.9488 - val_loss: 0.5451 - val_dice_coef: 0.2428 - val_iou_metric: 0.0000e+00 - val_binary_accuracy: 0.7513
Epoch 6/40
6/6 [==============================] - 8s 1s/step - loss: 0.1301 - dice_coef: 0.8284 - iou_metric: 0.8503 - binary_accu
racy: 0.9422 - val_loss: 0.5455 - val_dice_coef: 0.2210 - val_iou_metric: 0.0000e+00 - val_binary_accuracy: 0.7513
Epoch 7/40
6/6 [==============================] - 7s 1s/step - loss: 0.1170 - dice_coef: 0.8445 - iou_metric: 0.8635 - binary_accu
racy: 0.9444 - val_loss: 0.5511 - val_dice_coef: 0.2042 - val_iou_metric: 0.0000e+00 - val_binary_accuracy: 0.7513
Epoch 8/40
6/6 [==============================] - 7s 1s/step - loss: 0.1110 - dice_coef: 0.8551 - iou_metric: 0.8668 - binary_accu
racy: 0.9451 - val_loss: 0.5614 - val_dice_coef: 0.1875 - val_iou_metric: 0.0000e+00 - val_binary_accuracy: 0.7513
Epoch 9/40
6/6 [==============================] - 7s 1s/step - loss: 0.1056 - dice_coef: 0.8676 - iou_metric: 0.8769 - binary_accu
racy: 0.9463 - val_loss: 0.5770 - val_dice_coef: 0.1720 - val_iou_metric: 0.0000e+00 - val_binary_accuracy: 0.7513
Epoch 10/40
6/6 [==============================] - 7s 1s/step - loss: 0.0938 - dice_coef: 0.8786 - iou_metric: 0.8890 - binary_accu
racy: 0.9497 - val_loss: 0.5916 - val_dice_coef: 0.1587 - val_iou_metric: 0.0000e+00 - val_binary_accuracy: 0.7513
Epoch 11/40
6/6 [==============================] - 7s 1s/step - loss: 0.0927 - dice_coef: 0.8851 - iou_metric: 0.8844 - binary_accu
racy: 0.9483 - val_loss: 0.6099 - val_dice_coef: 0.1454 - val_iou_metric: 0.0000e+00 - val_binary_accuracy: 0.7513
Epoch 12/40
6/6 [==============================] - 7s 1s/step - loss: 0.0777 - dice_coef: 0.8957 - iou_metric: 0.9033 - binary_accu
racy: 0.9527 - val_loss: 0.6229 - val_dice_coef: 0.1368 - val_iou_metric: 0.0000e+00 - val_binary_accuracy: 0.7513
Epoch 13/40
6/6 [==============================] - 7s 1s/step - loss: 0.0798 - dice_coef: 0.8990 - iou_metric: 0.8990 - binary_accu
racy: 0.9520 - val_loss: 0.6369 - val_dice_coef: 0.1280 - val_iou_metric: 0.0000e+00 - val_binary_accuracy: 0.7513
Epoch 14/40
6/6 [==============================] - 7s 1s/step - loss: 0.0863 - dice_coef: 0.8932 - iou_metric: 0.8927 - binary_accu
racy: 0.9495 - val_loss: 0.6543 - val_dice_coef: 0.1191 - val_iou_metric: 0.0000e+00 - val_binary_accuracy: 0.7513
Epoch 15/40
```

```
6/6 [==============================] - 9s 1s/step - loss: 0.0830 - dice_coef: 0.9033 - iou_metric: 0.8948 - binary_accu
racy: 0.9505 - val_loss: 0.6667 - val_dice_coef: 0.1126 - val_iou_metric: 0.0000e+00 - val_binary_accuracy: 0.7513
Epoch 16/40
6/6 [==============================] - 7s 1s/step - loss: 0.0718 - dice_coef: 0.9120 - iou_metric: 0.9098 - binary_accu
racy: 0.9529 - val_loss: 0.6832 - val_dice_coef: 0.1064 - val_iou_metric: 0.0000e+00 - val_binary_accuracy: 0.7513
Epoch 17/40
6/6 [==============================] - 7s 1s/step - loss: 0.0753 - dice_coef: 0.9090 - iou_metric: 0.9034 - binary_accu
racy: 0.9517 - val_loss: 0.6939 - val_dice_coef: 0.1020 - val_iou_metric: 0.0000e+00 - val_binary_accuracy: 0.7513
Epoch 18/40
6/6 [==============================] - 8s 1s/step - loss: 0.0663 - dice_coef: 0.9146 - iou_metric: 0.9121 - binary_accu
racy: 0.9537 - val_loss: 0.6751 - val_dice_coef: 0.1139 - val_iou_metric: 0.0000e+00 - val_binary_accuracy: 0.7496
Epoch 19/40
6/6 [==============================] - 7s 1s/step - loss: 0.0599 - dice_coef: 0.9207 - iou_metric: 0.9232 - binary_accu
racy: 0.9566 - val_loss: 0.7068 - val_dice_coef: 0.0952 - val_iou_metric: 0.0000e+00 - val_binary_accuracy: 0.7513
Epoch 20/40
6/6 [==============================] - 7s 1s/step - loss: 0.0873 - dice_coef: 0.9054 - iou_metric: 0.8861 - binary_accu
racy: 0.9475 - val_loss: 0.7068 - val_dice_coef: 0.1071 - val_iou_metric: 4.4874e-04 - val_binary_accuracy: 0.7353
Epoch 21/40
6/6 [==============================] - 7s 1s/step - loss: 0.0670 - dice_coef: 0.9172 - iou_metric: 0.9113 - binary_accu
racy: 0.9532 - val_loss: 0.7142 - val_dice_coef: 0.1012 - val_iou_metric: 1.6052e-04 - val_binary_accuracy: 0.7441
Epoch 22/40
6/6 [==============================] - 7s 1s/step - loss: 0.0732 - dice_coef: 0.9155 - iou_metric: 0.9029 - binary_accu
racy: 0.9514 - val_loss: 0.7210 - val_dice_coef: 0.0919 - val_iou_metric: 0.0000e+00 - val_binary_accuracy: 0.7508
Epoch 23/40
6/6 [==============================] - 7s 1s/step - loss: 0.0746 - dice_coef: 0.9151 - iou_metric: 0.9027 - binary_accu
racy: 0.9509 - val_loss: 0.7511 - val_dice_coef: 0.0816 - val_iou_metric: 0.0000e+00 - val_binary_accuracy: 0.7502
Epoch 24/40
6/6 [==============================] - 7s 1s/step - loss: 0.0475 - dice_coef: 0.9352 - iou_metric: 0.9356 - binary_accu
racy: 0.9590 - val_loss: 0.7126 - val_dice_coef: 0.1128 - val_iou_metric: 9.4089e-04 - val_binary_accuracy: 0.7183
Epoch 25/40
6/6 [==============================] - 7s 1s/step - loss: 0.0798 - dice_coef: 0.9117 - iou_metric: 0.8941 - binary_accu
racy: 0.9486 - val_loss: 0.7539 - val_dice_coef: 0.0835 - val_iou_metric: 6.4184e-05 - val_binary_accuracy: 0.7440
Epoch 26/40
6/6 [==============================] - 7s 1s/step - loss: 0.0537 - dice_coef: 0.9277 - iou_metric: 0.9289 - binary_accu
racy: 0.9570 - val_loss: 0.6564 - val_dice_coef: 0.1543 - val_iou_metric: 0.0031 - val_binary_accuracy: 0.6973
Epoch 27/40
6/6 [==============================] - 9s 1s/step - loss: 0.0556 - dice_coef: 0.9312 - iou_metric: 0.9222 - binary_accu
racy: 0.9558 - val_loss: 0.7010 - val_dice_coef: 0.1192 - val_iou_metric: 0.0011 - val_binary_accuracy: 0.7165
Epoch 28/40
6/6 [==============================] - 9s 1s/step - loss: 0.0598 - dice_coef: 0.9266 - iou_metric: 0.9202 - binary_accu
racy: 0.9541 - val_loss: 0.6897 - val_dice_coef: 0.1113 - val_iou_metric: 2.5351e-04 - val_binary_accuracy: 0.7410
Epoch 29/40
6/6 [==============================] - 8s 1s/step - loss: 0.0536 - dice_coef: 0.9316 - iou_metric: 0.9269 - binary_accu
racy: 0.9562 - val_loss: 0.7230 - val_dice_coef: 0.1111 - val_iou_metric: 0.0014 - val_binary_accuracy: 0.7134
Epoch 30/40
```

```
6/6 [==============================] - 8s 1s/step - loss: 0.0516 - dice_coef: 0.9368 - iou_metric: 0.9268 - binary_accu
racy: 0.9567 - val_loss: 0.7674 - val_dice_coef: 0.0866 - val_iou_metric: 6.6795e-04 - val_binary_accuracy: 0.7249
Epoch 31/40
6/6 [==============================] - 7s 1s/step - loss: 0.0555 - dice_coef: 0.9317 - iou_metric: 0.9189 - binary_accu
racy: 0.9545 - val_loss: 0.7842 - val_dice_coef: 0.0861 - val_iou_metric: 0.0012 - val_binary_accuracy: 0.7117
Epoch 32/40
6/6 [==============================] - 7s 1s/step - loss: 0.0450 - dice_coef: 0.9368 - iou_metric: 0.9355 - binary_accu
racy: 0.9589 - val_loss: 0.7290 - val_dice_coef: 0.1085 - val_iou_metric: 0.0014 - val_binary_accuracy: 0.7173
Epoch 33/40
6/6 [==============================] - 7s 1s/step - loss: 0.0544 - dice_coef: 0.9346 - iou_metric: 0.9230 - binary_accu
racy: 0.9553 - val_loss: 0.8468 - val_dice_coef: 0.0712 - val_iou_metric: 0.0014 - val_binary_accuracy: 0.7052
Epoch 34/40
6/6 [==============================] - 7s 1s/step - loss: 0.0649 - dice_coef: 0.9276 - iou_metric: 0.9124 - binary_accu
racy: 0.9511 - val_loss: 0.8710 - val_dice_coef: 0.0685 - val_iou_metric: 0.0016 - val_binary_accuracy: 0.7047
Epoch 35/40
6/6 [==============================] - 7s 1s/step - loss: 0.0410 - dice_coef: 0.9423 - iou_metric: 0.9416 - binary_accu
racy: 0.9596 - val_loss: 0.9184 - val_dice_coef: 0.0601 - val_iou_metric: 0.0017 - val_binary_accuracy: 0.7032
Epoch 36/40
6/6 [==============================] - 8s 1s/step - loss: 0.0485 - dice_coef: 0.9413 - iou_metric: 0.9289 - binary_accu
racy: 0.9564 - val_loss: 0.9634 - val_dice_coef: 0.0746 - val_iou_metric: 0.0105 - val_binary_accuracy: 0.6614
Epoch 37/40
6/6 [==============================] - 7s 1s/step - loss: 0.0446 - dice_coef: 0.9431 - iou_metric: 0.9351 - binary_accu
racy: 0.9578 - val_loss: 1.0734 - val_dice_coef: 0.0676 - val_iou_metric: 0.0142 - val_binary_accuracy: 0.6553
Epoch 38/40
6/6 [==============================] - 7s 1s/step - loss: 0.0517 - dice_coef: 0.9393 - iou_metric: 0.9262 - binary_accu
racy: 0.9545 - val_loss: 0.9678 - val_dice_coef: 0.0808 - val_iou_metric: 0.0139 - val_binary_accuracy: 0.6587
Epoch 39/40
6/6 [==============================] - 7s 1s/step - loss: 0.0343 - dice_coef: 0.9507 - iou_metric: 0.9497 - binary_accu
racy: 0.9617 - val_loss: 0.9281 - val_dice_coef: 0.0745 - val_iou_metric: 0.0080 - val_binary_accuracy: 0.6946
Epoch 40/40
6/6 [==============================] - 7s 1s/step - loss: 0.0524 - dice_coef: 0.9396 - iou_metric: 0.9269 - binary_accu
racy: 0.9555 - val_loss: 0.8585 - val_dice_coef: 0.1110 - val_iou_metric: 0.0223 - val_binary_accuracy: 0.6594
1/1 [==============================] - 1s 523ms/step - loss: 0.8577 - dice_coef: 0.1013 - iou_metric: 0.0199 - binary_a
ccuracy: 0.6581
Test Loss: 0.8576883673667908, Test Dice Coefficient: 0.10134436190128326
2/2 [==============================] - 1s 153ms/step
```

UNet model summary statistics and model run time attributes

In [27]:
```
print("Unet Model Summary:")
model.summary()
```

```
Unet Model Summary:
Model: "model_1"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_2 (InputLayer)        [(None, 128, 128, 3)]        0         []

 conv2d_4 (Conv2D)           (None, 64, 64, 64)           1792      ['input_2[0][0]']

 batch_normalization_5 (Bat  (None, 64, 64, 64)           256       ['conv2d_4[0][0]']
 chNormalization)

 conv2d_5 (Conv2D)           (None, 32, 32, 128)          73856     ['batch_normalization_5[0][0]'
                                                                    ]

 batch_normalization_6 (Bat  (None, 32, 32, 128)          512       ['conv2d_5[0][0]']
 chNormalization)

 conv2d_6 (Conv2D)           (None, 16, 16, 256)          295168    ['batch_normalization_6[0][0]'
                                                                    ]

 batch_normalization_7 (Bat  (None, 16, 16, 256)          1024      ['conv2d_6[0][0]']
 chNormalization)

 conv2d_transpose_2 (Conv2D  (None, 32, 32, 128)          295040    ['batch_normalization_7[0][0]'
 Transpose)                                                         ]

 tf.image.resize_1 (TFOpLam  (None, 32, 32, 256)          0         ['conv2d_6[0][0]']
 bda)

 tf.concat_5 (TFOpLambda)    (None, 32, 32, 384)          0         ['conv2d_transpose_2[0][0]',
                                                                     'tf.image.resize_1[0][0]']

 batch_normalization_8 (Bat  (None, 32, 32, 384)          1536      ['tf.concat_5[0][0]']
 chNormalization)

 tf.concat_6 (TFOpLambda)    (None, 32, 32, 512)          0         ['batch_normalization_8[0][0]'
                                                                    , 'batch_normalization_6[0][0]
                                                                    ']

 dropout_2 (Dropout)         (None, 32, 32, 512)          0         ['tf.concat_6[0][0]']

 tf.concat_7 (TFOpLambda)    (None, 32, 32, 640)          0         ['dropout_2[0][0]',
                                                                     'conv2d_5[0][0]']
```

```
 conv2d_transpose_3 (Conv2D  (None, 64, 64, 64)          368704    ['tf.concat_7[0][0]']
 Transpose)

 tf.concat_8 (TFOpLambda)    (None, 64, 64, 128)         0         ['conv2d_transpose_3[0][0]',
                                                                   'batch_normalization_5[0][0]'
                                                                   ]

 batch_normalization_9 (Bat  (None, 64, 64, 128)         512       ['tf.concat_8[0][0]']
 chNormalization)

 tf.concat_9 (TFOpLambda)    (None, 64, 64, 192)         0         ['batch_normalization_9[0][0]'
                                                                   , 'conv2d_4[0][0]']

 conv2d_7 (Conv2D)           (None, 64, 64, 1)           193       ['tf.concat_9[0][0]']

==================================================================================================
Total params: 1038593 (3.96 MB)
Trainable params: 1036673 (3.95 MB)
Non-trainable params: 1920 (7.50 KB)
__________________________________________________________________________________________________
```

UNet Image Prediction Accuracy Measurements

UNet Training and Validation Accuracy per Epoch

UNet Training Binary Accuracy per Epoch

In [28]:
```
print("\nAvailable UNet history metrics:", history.history.keys())

if 'accuracy' in history.history:
    print("\nUNet Training and Validation Accuracy per Epoch:")
    for epoch in range(len(history.history['accuracy'])):
        bin_acc = history.history.get('binary_accuracy', [None])[epoch]
        train_acc = history.history['accuracy'][epoch]
        val_acc = history.history.get('val_accuracy', [None])[epoch]
        print(f"Epoch {epoch + 1}: Binary Accuracy = {bin_acc}, Training Accuracy = {train_acc}, Validation Accuracy =
else:
    if 'binary_accuracy' in history.history:
        print("\nUNet Training Binary Accuracy per Epoch:")
        for epoch in range(len(history.history['binary_accuracy'])):
            bin_acc = history.history['binary_accuracy'][epoch]
            print(f"Epoch {epoch + 1}: Binary Accuracy = {bin_acc}")

print("\nUNet Training and Validation Loss per Epoch:")
```

```python
for epoch in range(len(history.history['loss'])):
    train_loss = history.history['loss'][epoch]
    val_loss = history.history['val_loss'][epoch]
    print(f"Epoch {epoch + 1}: Training Loss = {train_loss}, Validation Loss = {val_loss}")
```

```
Available UNet history metrics: dict_keys(['loss', 'dice_coef', 'iou_metric', 'binary_accuracy', 'val_loss', 'val_dice_
coef', 'val_iou_metric', 'val_binary_accuracy'])

UNet Training Binary Accuracy per Epoch:
Epoch 1: Binary Accuracy = 0.7616608738899231
Epoch 2: Binary Accuracy = 0.9192212820053101
Epoch 3: Binary Accuracy = 0.9310521483421326
Epoch 4: Binary Accuracy = 0.9337277412414551
Epoch 5: Binary Accuracy = 0.9488319754600525
Epoch 6: Binary Accuracy = 0.9422083497047424
Epoch 7: Binary Accuracy = 0.9443982839584351
Epoch 8: Binary Accuracy = 0.9451274871826172
Epoch 9: Binary Accuracy = 0.9463168978691101
Epoch 10: Binary Accuracy = 0.9496692419052124
Epoch 11: Binary Accuracy = 0.9482607841491699
Epoch 12: Binary Accuracy = 0.9527249336242676
Epoch 13: Binary Accuracy = 0.9519772529602051
Epoch 14: Binary Accuracy = 0.9495099782943726
Epoch 15: Binary Accuracy = 0.9505131840705872
Epoch 16: Binary Accuracy = 0.9528841972351074
Epoch 17: Binary Accuracy = 0.9516714215278625
Epoch 18: Binary Accuracy = 0.9537220597267151
Epoch 19: Binary Accuracy = 0.9566245675086975
Epoch 20: Binary Accuracy = 0.9474560022354126
Epoch 21: Binary Accuracy = 0.95316481590271
Epoch 22: Binary Accuracy = 0.9513950347900391
Epoch 23: Binary Accuracy = 0.9509031772613525
Epoch 24: Binary Accuracy = 0.95899897813797
Epoch 25: Binary Accuracy = 0.9486256241798401
Epoch 26: Binary Accuracy = 0.957025945186615
Epoch 27: Binary Accuracy = 0.9557787179946899
Epoch 28: Binary Accuracy = 0.9541049003601074
Epoch 29: Binary Accuracy = 0.9562276005744934
Epoch 30: Binary Accuracy = 0.956695556640625
Epoch 31: Binary Accuracy = 0.9544525146484375
Epoch 32: Binary Accuracy = 0.9589293003082275
Epoch 33: Binary Accuracy = 0.9553050398826599
Epoch 34: Binary Accuracy = 0.9511340856552124
Epoch 35: Binary Accuracy = 0.95963054895401
Epoch 36: Binary Accuracy = 0.9564380645751953
Epoch 37: Binary Accuracy = 0.9578220248222351
Epoch 38: Binary Accuracy = 0.9544551968574524
Epoch 39: Binary Accuracy = 0.9616692662239075
Epoch 40: Binary Accuracy = 0.9555478096008301
```

```
UNet Training and Validation Loss per Epoch:
Epoch 1: Training Loss = 0.48668089509010315, Validation Loss = 0.6466498374938965
Epoch 2: Training Loss = 0.24599780142307281, Validation Loss = 0.6134670972824097
Epoch 3: Training Loss = 0.19162337481975555, Validation Loss = 0.5801104307174683
Epoch 4: Training Loss = 0.1686292439699173, Validation Loss = 0.5566225051879883
Epoch 5: Training Loss = 0.11807449907064438, Validation Loss = 0.5451236963272095
Epoch 6: Training Loss = 0.13012953102588654, Validation Loss = 0.5454918742179871
Epoch 7: Training Loss = 0.11699752509593964, Validation Loss = 0.5511112213134766
Epoch 8: Training Loss = 0.111014723777771, Validation Loss = 0.5613746047019958
Epoch 9: Training Loss = 0.10556378215551376, Validation Loss = 0.5770033597946167
Epoch 10: Training Loss = 0.09380645304918289, Validation Loss = 0.5916407108306885
Epoch 11: Training Loss = 0.09266455471515656, Validation Loss = 0.6098996996879578
Epoch 12: Training Loss = 0.07772928476333618, Validation Loss = 0.6229066848754883
Epoch 13: Training Loss = 0.07978708297014236, Validation Loss = 0.6368827819824219
Epoch 14: Training Loss = 0.08634215593338013, Validation Loss = 0.6542666554450989
Epoch 15: Training Loss = 0.0830368921160698, Validation Loss = 0.6667016744613647
Epoch 16: Training Loss = 0.07183875143527985, Validation Loss = 0.6832078695297241
Epoch 17: Training Loss = 0.07527673989534378, Validation Loss = 0.6939318776130676
Epoch 18: Training Loss = 0.06633920967578888, Validation Loss = 0.6750707030296326
Epoch 19: Training Loss = 0.05987241491675377, Validation Loss = 0.7068068385124207
Epoch 20: Training Loss = 0.08729894459247589, Validation Loss = 0.7067846059799194
Epoch 21: Training Loss = 0.06699603796005249, Validation Loss = 0.7142211198806763
Epoch 22: Training Loss = 0.0732010155916214, Validation Loss = 0.7210294008255005
Epoch 23: Training Loss = 0.07462058216333389, Validation Loss = 0.7511320114135742
Epoch 24: Training Loss = 0.04750141501426697, Validation Loss = 0.7125552892684937
Epoch 25: Training Loss = 0.07977598905563354, Validation Loss = 0.7539489269256592
Epoch 26: Training Loss = 0.05374374985694885, Validation Loss = 0.6563783884048462
Epoch 27: Training Loss = 0.055615149438381195, Validation Loss = 0.7009754776954651
Epoch 28: Training Loss = 0.05982902646064758, Validation Loss = 0.6897066831588745
Epoch 29: Training Loss = 0.05363905057311058, Validation Loss = 0.7230066657066345
Epoch 30: Training Loss = 0.051630329340696335, Validation Loss = 0.7673784494400024
Epoch 31: Training Loss = 0.055548034608364105, Validation Loss = 0.7841570377349854
Epoch 32: Training Loss = 0.04504888504743576, Validation Loss = 0.7290158271789551
Epoch 33: Training Loss = 0.05441389977931976, Validation Loss = 0.8468238115310669
Epoch 34: Training Loss = 0.06487108021974564, Validation Loss = 0.8709536194801331
Epoch 35: Training Loss = 0.04096956178545952, Validation Loss = 0.9184094667434692
Epoch 36: Training Loss = 0.048469528555870056, Validation Loss = 0.9633762240409851
Epoch 37: Training Loss = 0.044585563242435455, Validation Loss = 1.073431372642517
Epoch 38: Training Loss = 0.05169304460287094, Validation Loss = 0.9678082466125488
Epoch 39: Training Loss = 0.03425786271691322, Validation Loss = 0.92806077003479
Epoch 40: Training Loss = 0.052404604852199554, Validation Loss = 0.8585328459739685
```

UNet Image Prediction Accuracy Measurements

UNet Training and Validation Dice Coefficient per Epoch

UNet Training and Validation IoU per Epoch

In [29]:
```
if 'dice_coef' in history.history:
    print("\nUNet Training and Validation Dice Coefficient per Epoch:")
    for epoch in range(len(history.history['dice_coef'])):
        train_dice = history.history['dice_coef'][epoch]
        val_dice = history.history['val_dice_coef'][epoch]
        print(f"Epoch {epoch + 1}: Training Dice = {train_dice}, Validation Dice = {val_dice}")

if 'iou_metric' in history.history:
    print("\nUNet Training and Validation IoU per Epoch:")
    for epoch in range(len(history.history['iou_metric'])):
        train_iou = history.history['iou_metric'][epoch]
        val_iou = history.history['val_iou_metric'][epoch]
        print(f"Epoch {epoch + 1}: Training IoU = {train_iou}, Validation IoU = {val_iou}")
```

```
UNet Training and Validation Dice Coefficient per Epoch:
Epoch 1: Training Dice = 0.5416263937950134, Validation Dice = 0.3025226593017578
Epoch 2: Training Dice = 0.6834365725517273, Validation Dice = 0.29389816522598267
Epoch 3: Training Dice = 0.7380973696708679, Validation Dice = 0.27980300784111023
Epoch 4: Training Dice = 0.7776500582695007, Validation Dice = 0.2627998888492584
Epoch 5: Training Dice = 0.8219916820526123, Validation Dice = 0.2428373247385025
Epoch 6: Training Dice = 0.8284400105476379, Validation Dice = 0.22101406753063202
Epoch 7: Training Dice = 0.8445053696632385, Validation Dice = 0.20421577990055084
Epoch 8: Training Dice = 0.8551342487335205, Validation Dice = 0.18752612173557281
Epoch 9: Training Dice = 0.8675654530525208, Validation Dice = 0.1719646006822586
Epoch 10: Training Dice = 0.8786045908927917, Validation Dice = 0.15874309837818146
Epoch 11: Training Dice = 0.8850577473640442, Validation Dice = 0.14543107151985168
Epoch 12: Training Dice = 0.8957307934761047, Validation Dice = 0.1367737203836441
Epoch 13: Training Dice = 0.8990095257759094, Validation Dice = 0.12799350917339325
Epoch 14: Training Dice = 0.8932010531425476, Validation Dice = 0.11910951137542725
Epoch 15: Training Dice = 0.9032933712005615, Validation Dice = 0.11264608055353165
Epoch 16: Training Dice = 0.9120224118232727, Validation Dice = 0.10639458149671555
Epoch 17: Training Dice = 0.9089837074279785, Validation Dice = 0.10201068967580795
Epoch 18: Training Dice = 0.9146437048912048, Validation Dice = 0.11392338573932648
Epoch 19: Training Dice = 0.9207383990287781, Validation Dice = 0.09515208750963211
Epoch 20: Training Dice = 0.9054129719734192, Validation Dice = 0.10706447809934616
Epoch 21: Training Dice = 0.9171708226203918, Validation Dice = 0.10124830901622772
Epoch 22: Training Dice = 0.9155423045158386, Validation Dice = 0.09189774096012115
Epoch 23: Training Dice = 0.9150527119636536, Validation Dice = 0.08159033209085464
Epoch 24: Training Dice = 0.9351778626441956, Validation Dice = 0.11277157813310623
Epoch 25: Training Dice = 0.9116849899291992, Validation Dice = 0.08346433192491531
Epoch 26: Training Dice = 0.9277489185333252, Validation Dice = 0.15427006781101227
Epoch 27: Training Dice = 0.9312495589256287, Validation Dice = 0.1191517785191536
Epoch 28: Training Dice = 0.9266481399536133, Validation Dice = 0.11132325232028961
Epoch 29: Training Dice = 0.9316322207450867, Validation Dice = 0.11109776794910431
Epoch 30: Training Dice = 0.936816930770874, Validation Dice = 0.08656200766563416
Epoch 31: Training Dice = 0.931708574295044, Validation Dice = 0.08613533526659012
Epoch 32: Training Dice = 0.9368443489074707, Validation Dice = 0.10845168679952621
Epoch 33: Training Dice = 0.9346480369567871, Validation Dice = 0.07115687429904938
Epoch 34: Training Dice = 0.92756587266922, Validation Dice = 0.06854277849197388
Epoch 35: Training Dice = 0.9422586560249329, Validation Dice = 0.06014685705304146
Epoch 36: Training Dice = 0.9412931799888611, Validation Dice = 0.07455643266439438
Epoch 37: Training Dice = 0.9431259036064148, Validation Dice = 0.06757281720638275
Epoch 38: Training Dice = 0.9393060207366943, Validation Dice = 0.08076789230108261
Epoch 39: Training Dice = 0.9506590962409973, Validation Dice = 0.07447800785303116
Epoch 40: Training Dice = 0.9396064281463623, Validation Dice = 0.111024409532547

UNet Training and Validation IoU per Epoch:
Epoch 1: Training IoU = 0.5142787098884583, Validation IoU = 0.0
Epoch 2: Training IoU = 0.7667632102966309, Validation IoU = 0.0
```

```
Epoch 3: Training IoU = 0.8062124848365784, Validation IoU = 0.0
Epoch 4: Training IoU = 0.8235502243041992, Validation IoU = 0.0
Epoch 5: Training IoU = 0.8756197094917297, Validation IoU = 0.0
Epoch 6: Training IoU = 0.8503240942955017, Validation IoU = 0.0
Epoch 7: Training IoU = 0.863518238067627, Validation IoU = 0.0
Epoch 8: Training IoU = 0.8668239116668701, Validation IoU = 0.0
Epoch 9: Training IoU = 0.8769257664680481, Validation IoU = 0.0
Epoch 10: Training IoU = 0.8890073895454407, Validation IoU = 0.0
Epoch 11: Training IoU = 0.8844139575958252, Validation IoU = 0.0
Epoch 12: Training IoU = 0.9032953381538391, Validation IoU = 0.0
Epoch 13: Training IoU = 0.8989861607551575, Validation IoU = 0.0
Epoch 14: Training IoU = 0.8927480578422546, Validation IoU = 0.0
Epoch 15: Training IoU = 0.8948318362236023, Validation IoU = 0.0
Epoch 16: Training IoU = 0.9097806811332703, Validation IoU = 0.0
Epoch 17: Training IoU = 0.9034140110015869, Validation IoU = 0.0
Epoch 18: Training IoU = 0.912081241607666, Validation IoU = 0.0
Epoch 19: Training IoU = 0.9232022762298584, Validation IoU = 0.0
Epoch 20: Training IoU = 0.8861474394798279, Validation IoU = 0.00044873665319755673
Epoch 21: Training IoU = 0.9113304018974304, Validation IoU = 0.00016051653074100614
Epoch 22: Training IoU = 0.9028670191764832, Validation IoU = 0.0
Epoch 23: Training IoU = 0.9027392864227295, Validation IoU = 0.0
Epoch 24: Training IoU = 0.9356340765953064, Validation IoU = 0.0009408945334143937
Epoch 25: Training IoU = 0.8941330909729004, Validation IoU = 6.418395059881732e-05
Epoch 26: Training IoU = 0.9289038181304932, Validation IoU = 0.003080890281125903
Epoch 27: Training IoU = 0.9222064018249512, Validation IoU = 0.0010859479662030935
Epoch 28: Training IoU = 0.92020583152771, Validation IoU = 0.00025350600481033325
Epoch 29: Training IoU = 0.9268901348114014, Validation IoU = 0.0014398348284885287
Epoch 30: Training IoU = 0.9268011450767517, Validation IoU = 0.0006679543294012547
Epoch 31: Training IoU = 0.9188569188117981, Validation IoU = 0.0011821106309071183
Epoch 32: Training IoU = 0.9355342388153076, Validation IoU = 0.0014160788850858808
Epoch 33: Training IoU = 0.9230132699012756, Validation IoU = 0.0014206618070602417
Epoch 34: Training IoU = 0.912444531917572, Validation IoU = 0.0016238378593698144
Epoch 35: Training IoU = 0.9415938258171082, Validation IoU = 0.0017359795747324824
Epoch 36: Training IoU = 0.9289396405220032, Validation IoU = 0.010472533293068409
Epoch 37: Training IoU = 0.9350941777229309, Validation IoU = 0.014155998826026917
Epoch 38: Training IoU = 0.9262472987174988, Validation IoU = 0.013938041403889656
Epoch 39: Training IoU = 0.9496616721153259, Validation IoU = 0.007967410609126091
Epoch 40: Training IoU = 0.9269313812255859, Validation IoU = 0.022257911041378975
```

UNet Image Prediction Sample Outcomes

In [22]:
```
display_images_with_labels(unet_prediction_output_dir, total_images=18, images_per_page=6, num_pages=5)
```


image_0_37.png


image_0_3.png


image_0_53.png


image_0_11.png

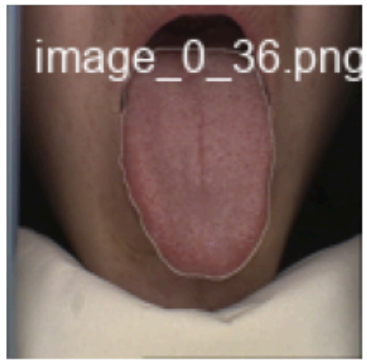
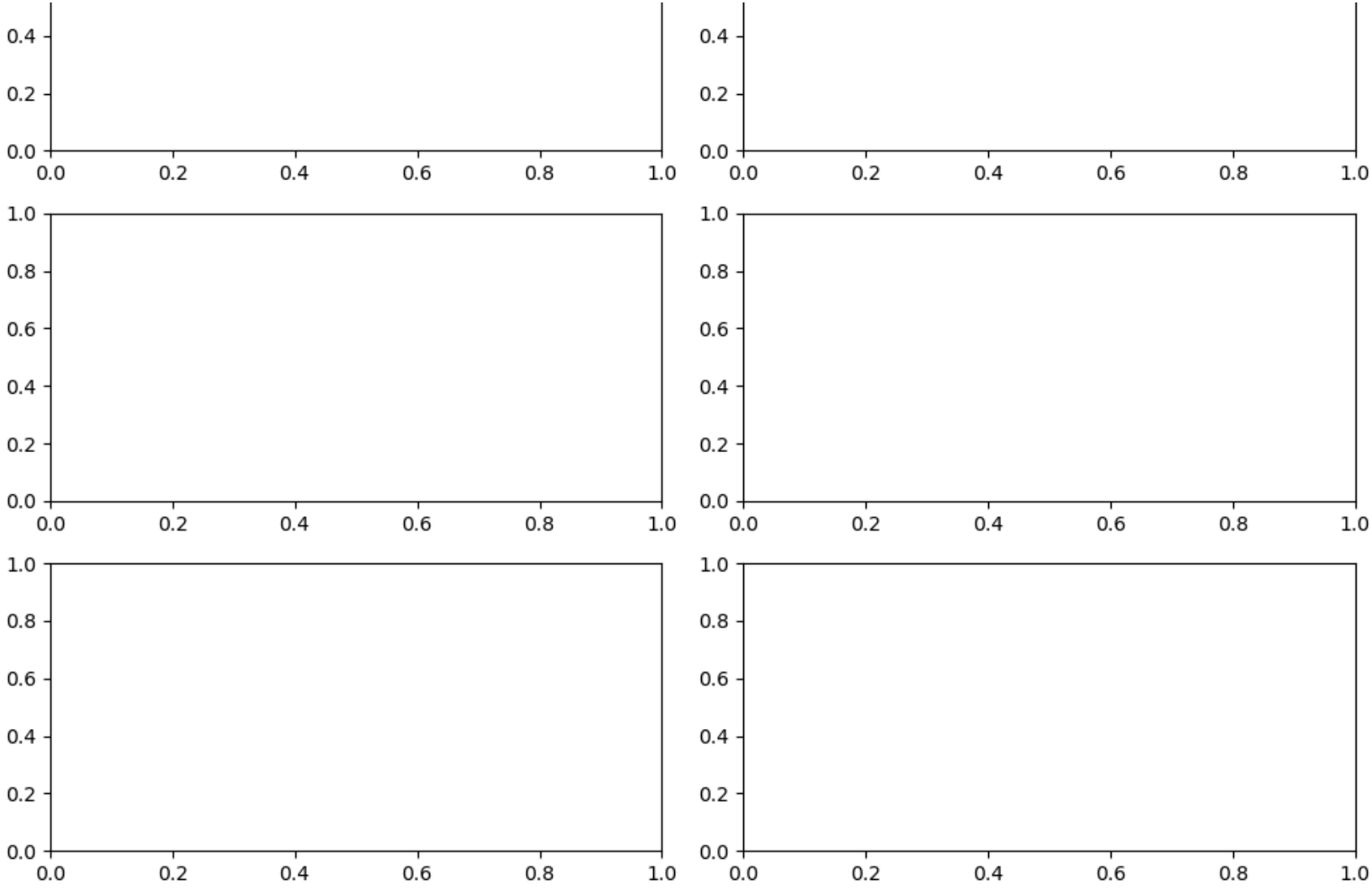


image_0_36.png


image_0_32.png


1.0
0.8
0.6


1.0
0.8
0.6


0.0
0.2
0.4
0.6
0.8
1.0

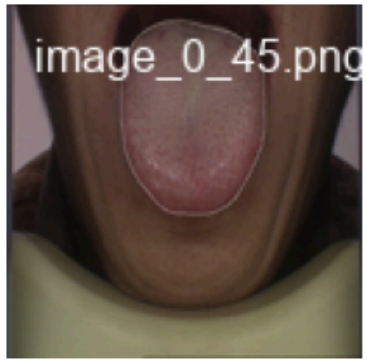


image_0_45.png


image_0_40.png

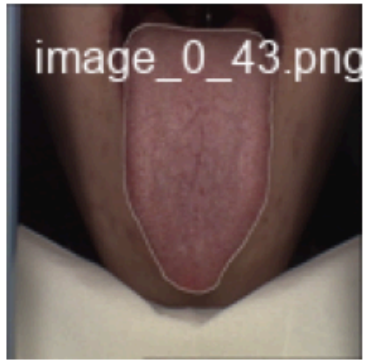


image_0_43.png


image_0_21.png

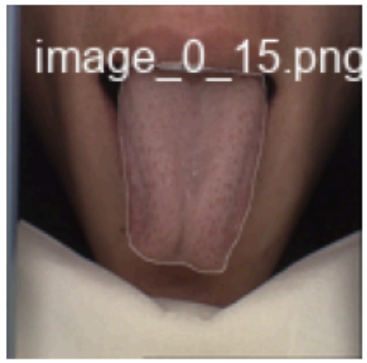


image_0_15.png

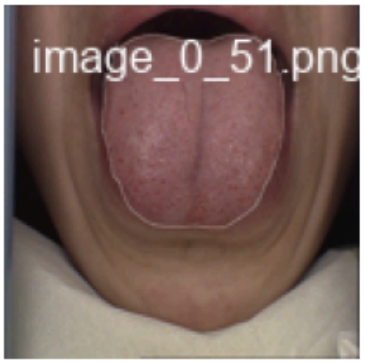


image_0_51.png


1.0
0.8
0.6


1.0
0.8
0.6

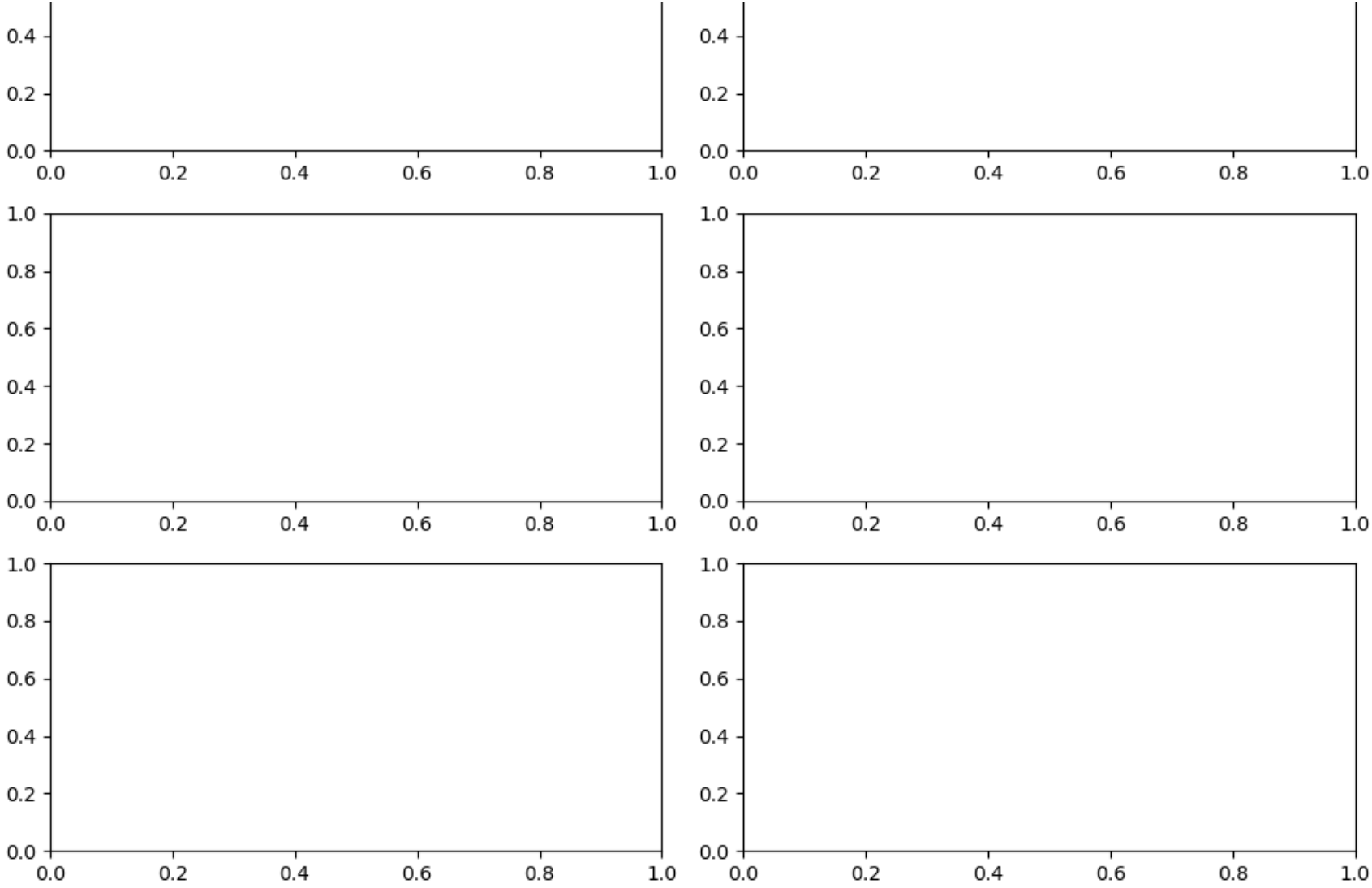

0.0
0.2
0.4
0.6
0.8
1.0


image_0_46.png


image_0_16.png


image_0_49.png

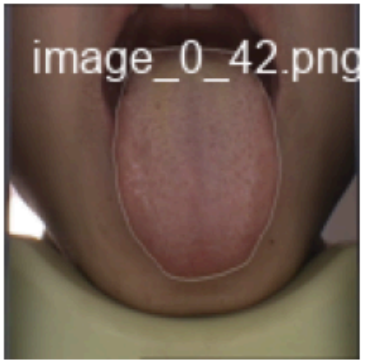


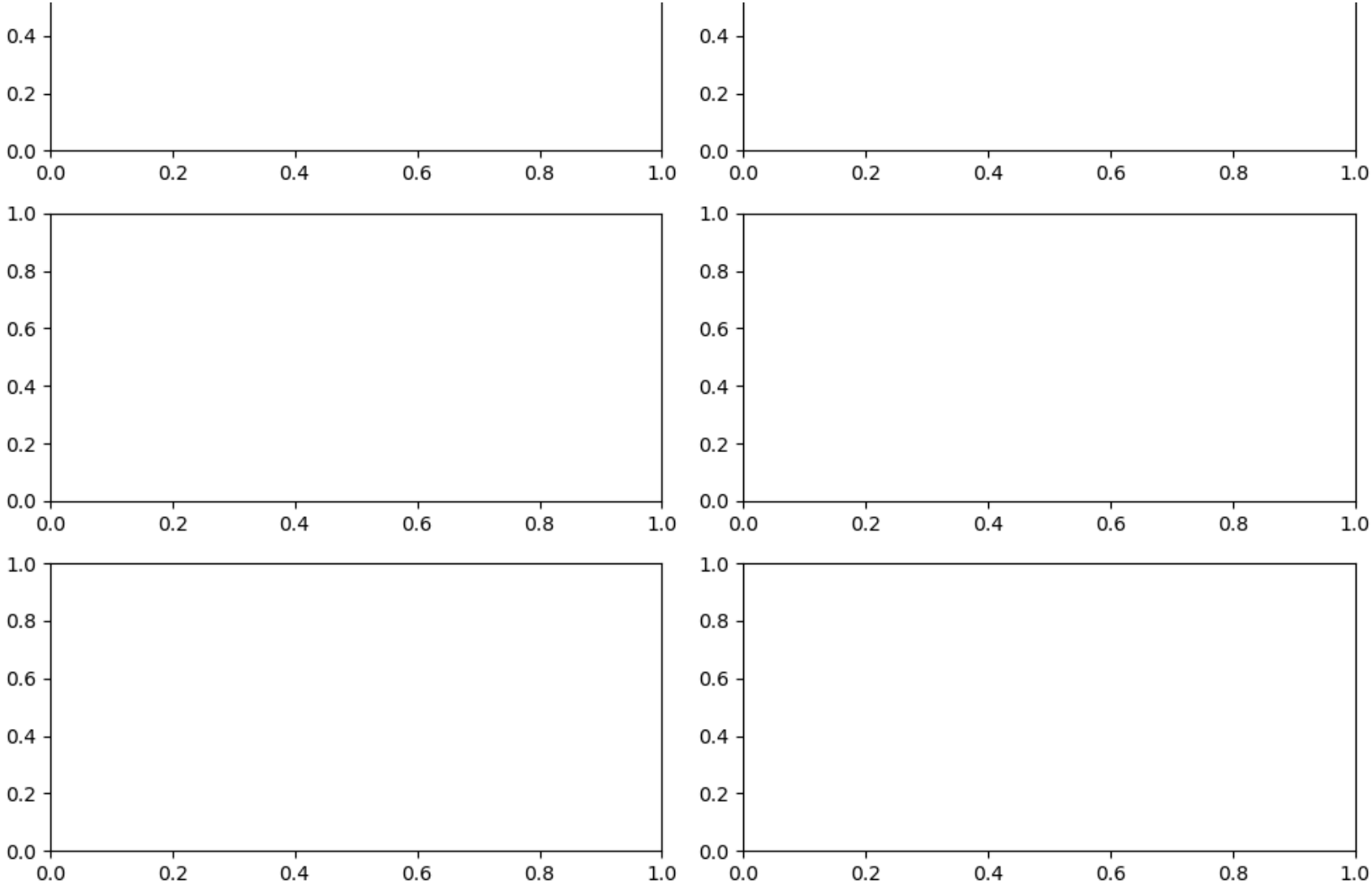
image_0_42.png


image_0_5.png


image_0_25.png


1.0
0.8
0.6


1.0
0.8
0.6


0.0
0.2
0.4
0.6
0.8
1.0

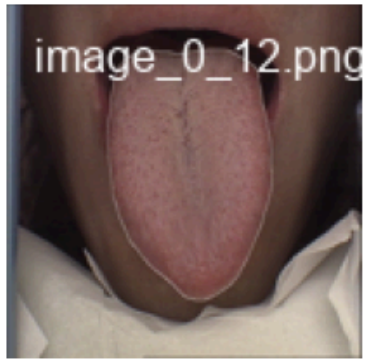


image_0_12.png

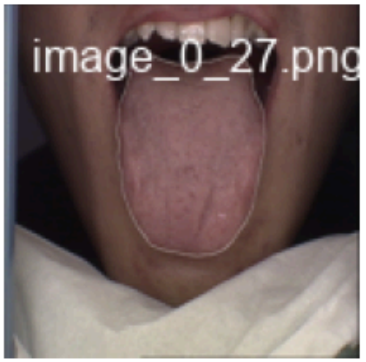


image_0_27.png


image_0_48.png

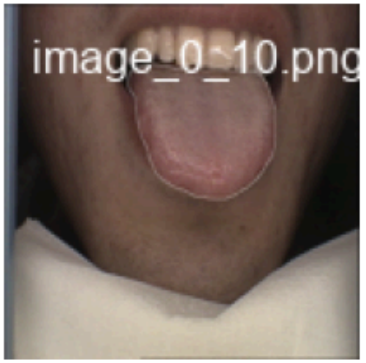


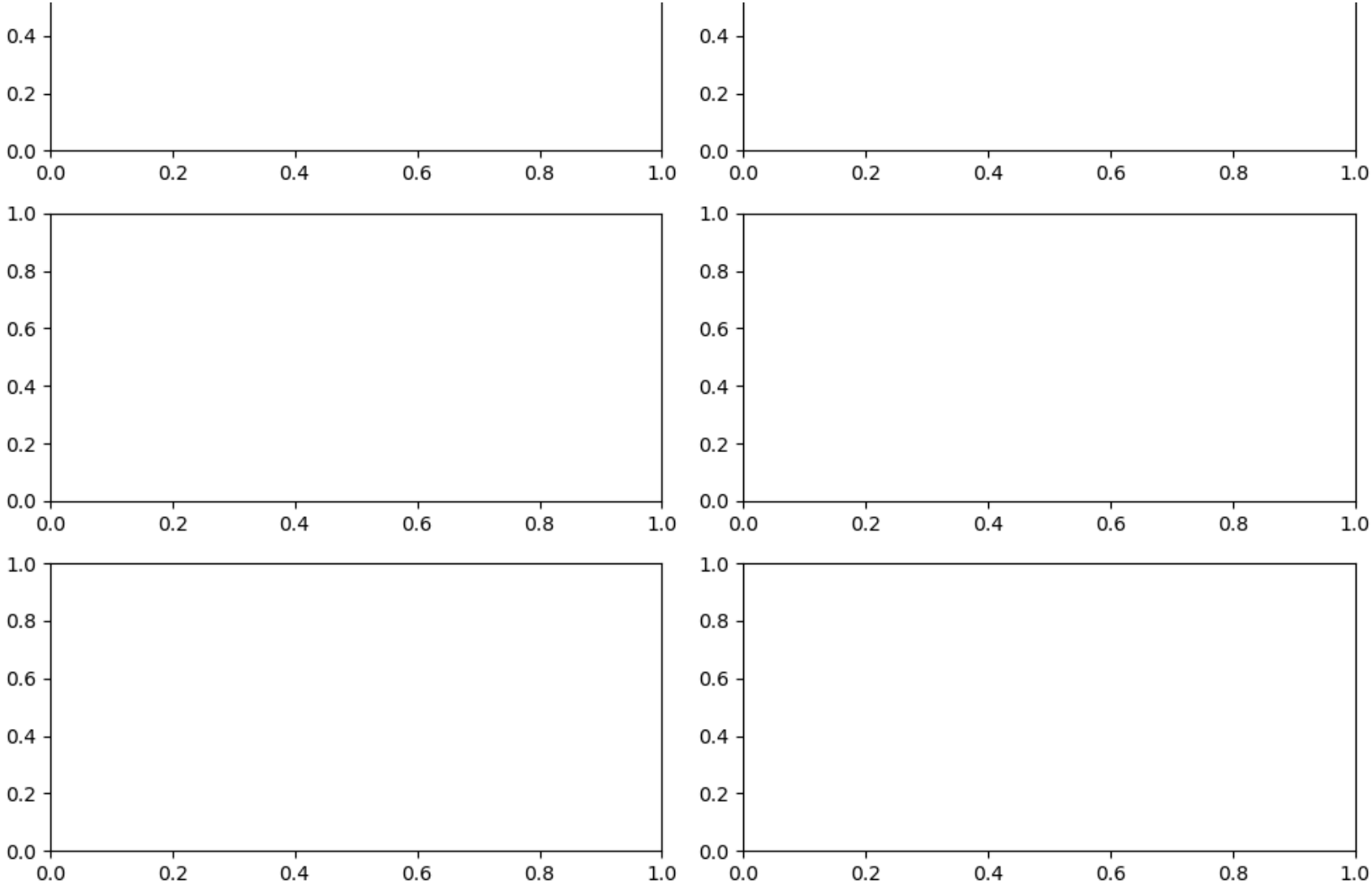
image_0_10.png


image_0_2.png


image_0_44.png


1.0
0.8
0.6


1.0
0.8
0.6


0.0
0.2
0.4
0.6
0.8
1.0

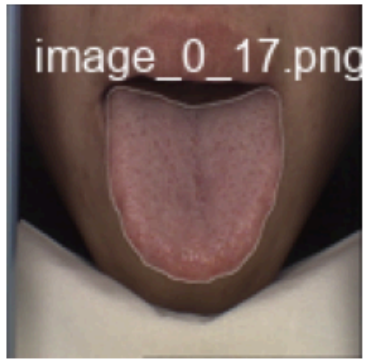


image_0_17.png


image_0_28.png


image_0_41.png

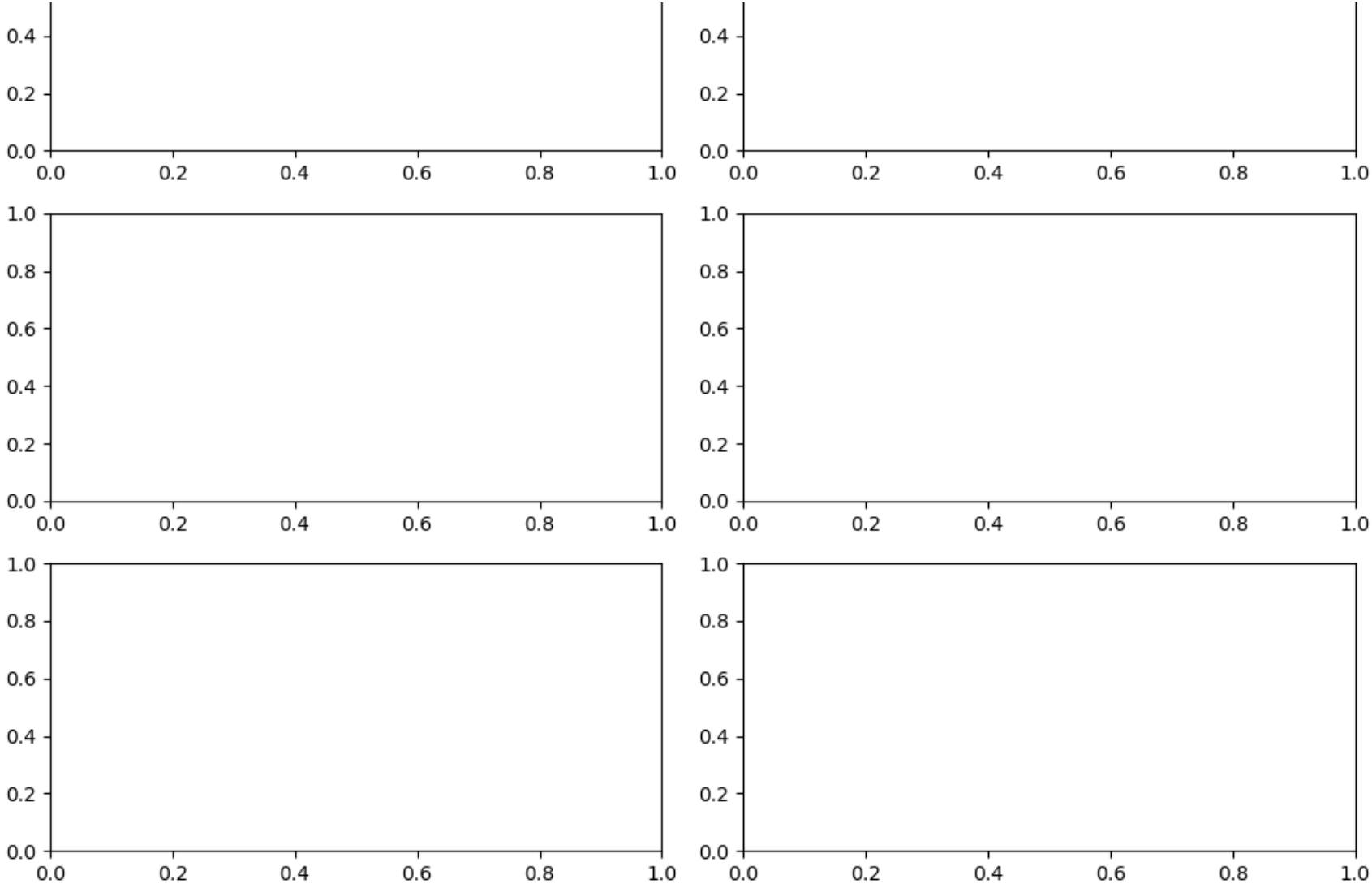

image_0_20.png


image_0_52.png


image_0_33.png


1.0
0.8
0.6


1.0
0.8
0.6



In [ ]:

In [ ]: