Kode

1 Code

```
import cv2
import os
import copy
import numpy as np


class RectangleExtractor:
    def __init__(self, image_path):
        self.image_path = image_path
        self.img = cv2.imread(image_path)
        if self.img.shape[:2] > (800, 800):
            self.img = cv2.resize(self.img, (600, 600))
        self.original_img = copy.deepcopy(self.img)
        self.drawing = False
        self.ix, self.iy = -1, -1
        self.extracted_areas = []

    def draw_rectangle(self, event, x, y, flags, param):
        if event == cv2.EVENT_LBUTTONDOWN:
            self.drawing = True
            self.ix, self.iy = x, y

        elif event == cv2.EVENT_LBUTTONUP:
            if self.drawing:
                cv2.rectangle(self.img, (self.ix, self.iy), (x, y),
 (0, 255, 0), 1)

                if self.ix > x:
                    self.ix, x = x, self.ix
                if self.iy > y:
                    self.iy, y = y, self.iy

                extracted_area = self.img[self.iy + 1 : y - 1, self
.ix + 1 : x - 1]
                self.extracted_areas.append(extracted_area)
                self.drawing = False
                self.original_img = copy.deepcopy(self.img)

        elif event == cv2.EVENT_MOUSEMOVE:
            if self.drawing:
                temp_img = copy.deepcopy(self.original_img)
                cv2.rectangle(temp_img, (self.ix, self.iy), (x, y),
```

```
     (0, 255, 0), 1)
                self.img = temp_img

    def start_extraction(self):
        cv2.namedWindow("image")
        cv2.setMouseCallback("image", self.draw_rectangle)

        while True:
            cv2.imshow("image", self.img)
            k = cv2.waitKey(1) & 0xFF
            if k == 27:  # Press 'Esc' to exit
                break

        cv2.destroyAllWindows()
        return self.extracted_areas


def create_dataset(pixels):
    dataset = []
    for i in range(len(pixels)):
        pixels[i] = pixels[i].reshape(-1, 3)
        pixels[i] = np.concatenate(
            (np.ones((pixels[i].shape[0], 1)) * (i + 1), pixels[i])
    , axis=1
        )
        dataset.extend(pixels[i])
    return np.array(dataset)


def estimate_pixels_apriori(pixels):
    probs = []
    for i in range(np.int64(np.max(pixels[:, 0], axis=0))):
        prob = np.sum(pixels[:, 0] == (i + 1)) / pixels.shape[0]
        probs.append(prob)
    return np.array(probs)


def estimate_pixels_mean(pixels):
    means = []
    for i in range(np.int64(np.max(pixels[:, 0], axis=0))):
        est_mean = pixels[pixels[:, 0] == (i + 1)].mean(axis=0)
        means.append(est_mean)
    return np.array(means)


def estimate_pixels_cov(pixels, pixel_class_means):
    covs = []
    for i in range(np.int64(np.max(pixels[:, 0], axis=0))):
        N_class = np.sum(pixels[:, 0] == (i + 1))
        class_dev = pixels[pixels[:, 0] == (i + 1), 1:] -
    pixel_class_means[i, 1:]
        class_cov = (class_dev.T @ class_dev) / (N_class - 1)
        covs.append(class_cov)

    return np.array(covs)


```

```python
def pixels_discriminants(pixel_means, pixel_covs, pixel_apriori):
    discriminants = []
    for i in range(len(pixel_means)):
        discriminants.append(
            class_discriminant(
                pixel_means[i], pixel_covs[i], pixel_apriori[i], pixels=True
            )
        )
    return discriminants


def class_discriminant(class_mean, class_cov, a_priori_prob, pixels=False):
    W = -(1 / 2) * np.linalg.inv(class_cov)

    w = np.linalg.inv(class_cov) @ class_mean

    det_cov = np.log(np.linalg.det(class_cov))
    det_cov = det_cov if det_cov > 1e-5 else 0

    w_0 = (
        -(1 / 2) * class_mean @ np.linalg.inv(class_cov) @ class_mean
        - (1 / 2) * det_cov
        + np.log(a_priori_prob)
    )
    if pixels:
        return lambda test_obs: test_obs.T @ W @ test_obs + test_obs @ w + w_0
    else:
        return (
            lambda test_obs: np.sum(test_obs @ W * test_obs, axis=1)
            + test_obs @ w
            + w_0
        )


def segment_image(image_path, discriminants):
    img = cv2.imread(image_path)
    if img.shape[:2] > (800, 800):
        img = cv2.resize(img, (600, 600))
    seg_img = np.zeros_like(img)
    colors = np.array(
        [
            [255, 0, 0],
            [0, 255, 0],
            [0, 0, 255],
            [255, 255, 0],
            [255, 0, 255],
            [0, 255, 255],
            [128, 0, 128],
            [255, 165, 0],
            [0, 128, 0],
            [128, 128, 128],
        ]
```

```
    )
    for x in range(img.shape[0]):
        for y in range(img.shape[1]):
            cl = np.argmax([disc(img[x, y]) for disc in
    discriminants])
            seg_img[x, y] = colors[cl]
    return seg_img


if __name__ == "__main__":
    train_image_path = os.path.dirname(os.path.dirname(__file__)) +
     "/data/Bilde1.png"
    test_image_path = os.path.dirname(os.path.dirname(__file__)) +
    "/data/Bilde1.png"
    extractor = RectangleExtractor(train_image_path)
    extracted_areas = extractor.start_extraction()

    dataset = create_dataset(extracted_areas)
    # norm_dataset = normalize_dataset(dataset)
    probs = estimate_pixels_apriori(dataset)
    means = estimate_pixels_mean(dataset)
    covs = estimate_pixels_cov(dataset, means)
    discs = pixels_discriminants(means[:, 1:], covs, probs)

    seg_img = segment_image(test_image_path, discs)
    combined_img = cv2.hconcat([extractor.img, seg_img])
    cv2.imwrite(
        os.path.dirname(os.path.dirname(__file__)) + "/data/
    Bilde1_segmentert.png",
        combined_img,
    )
    while True:
        cv2.imshow("image", combined_img)
        k = cv2.waitKey(1) & 0xFF
        if k == 27:  # Press 'Esc' to exit
            break

    cv2.destroyAllWindows()
    # disc1 = class_discriminant(means[0, 1:], covs[0], probs[0])

    # for i, area in enumerate(extracted_areas):
    #     cv2.imshow(f"Area {i}", area)
    #     cv2.waitKey(0)
    #     cv2.destroyAllWindows()
    #
```