**BLM6105-2 Uzaktan Algılama Dersi Ödev 6**

**24501100 - Aleyna Nil Uzunoğlu**

**Giriş**

Bu çalışmada, üç farklı skimage görüntüsü üzerinde konvolüsyon filtreleri, gradyan analizi ve kenar tespiti algoritmaları uygulanmıştır. Kullanılan yöntemler görüntü işleme ve uzaktan algılama alanlarında sıklıkla tercih edilen tekniklerdir.  
Ödevin üç ana bölümünden oluşan yapısı sayesinde, hem temel filtreleme teknikleri hem de gradyan tabanlı ve model tabanlı kenar tespit yöntemleri karşılaştırmalı olarak incelenmiştir. Veri seti olarak scikit-image kütüphanesinden alınan gri seviye üç test görüntüsü: camera, astronaut, coins kullanılmıştır.

**Adımlar**

*(a) Konvolüsyon Filtreleri Uygulama (Kutu, Gauss, Laplacian)*

Bu filtreler için k=3 ve k=5 çekirdek boyutlarında simetrik ve yansımalı padding uygulanarak konvolüsyon gerçekleştirilmiştir.

*(b) Gradyan Vektörleri (Magnitude & Orientation)*

* Görüntüdeki yönsel değişim sobel filtresi ile hesaplandı.
* Her piksel için:
  + Magnitude: Kenarın gücünü (parlaklık değişim şiddeti)
  + Orientation: Kenarın yönünü (açı bilgisi, derece cinsinden)

hesaplanmıştır.

*(c) Kenar Tespit Yöntemleri*

Aşağıdaki 5 kenar tespit algoritması her görüntüye uygulanmış ve görsel olarak karşılaştırılmıştır:

* Roberts
* Prewitt
* Sobel
* LoG (Laplacian of Gaussian)
* Canny

Ek: main.py

import numpy as np

import matplotlib.pyplot as plt

from skimage import data, io, color, util

from skimage.filters import gaussian, laplace

from skimage.filters import roberts, prewitt, sobel, sobel\_v, sobel\_h

from skimage.feature import canny

from scipy import ndimage as ndi

import os

OUTPUT\_DIR = "outputs"

os.makedirs(OUTPUT\_DIR, exist\_ok=True)

def apply\_convolution(image, filter\_type='box', k\_size=3, padding\_mode='reflect', \*\*kwargs):

if image.ndim > 2:

image = color.rgb2gray(image)

image = util.img\_as\_float(image)

if filter\_type == 'box':

if k\_size % 2 == 0:

k\_size += 1

kernel = np.ones((k\_size, k\_size)) / (k\_size \* k\_size)

return ndi.convolve(image, kernel, mode=padding\_mode)

elif filter\_type == 'gaussian':

sigma = (k\_size - 1) / 6.0 if k\_size > 1 else 1

return gaussian(image, sigma=sigma, mode=padding\_mode, \*\*kwargs)

elif filter\_type == 'laplacian':

return laplace(image, ksize=3, \*\*kwargs)

else:

raise ValueError("Desteklenmeyen filtre tipi. 'box', 'gaussian' veya 'laplacian' kullanın.")

def calculate\_gradients(image):

if image.ndim > 2:

image = color.rgb2gray(image)

image = util.img\_as\_float(image)

grad\_x = sobel\_v(image)

grad\_y = sobel\_h(image)

magnitude = np.sqrt(grad\_x\*\*2 + grad\_y\*\*2)

max\_mag = np.max(magnitude)

if max\_mag > np.finfo(float).eps:

magnitude = magnitude / max\_mag

orientation = np.arctan2(grad\_y, grad\_x)

return grad\_x, grad\_y, magnitude, orientation

def apply\_edge\_detectors(image, log\_sigma=1.0, canny\_sigma=1.0):

if image.ndim > 2:

image = color.rgb2gray(image)

image = util.img\_as\_float(image)

edges = {}

edges['Roberts'] = roberts(image)

edges['Prewitt'] = prewitt(image)

edges['Sobel'] = sobel(image)

im\_gaussian = gaussian(image, sigma=log\_sigma)

edges['LoG (Sigma={})'.format(log\_sigma)] = laplace(im\_gaussian)

edges['Canny (Sigma={})'.format(canny\_sigma)] = canny(image, sigma=canny\_sigma)

return edges

def plot\_comparison(original, filtered\_dict, main\_title="Karşılaştırma", cmap='gray', save\_path=None):

num\_images = len(filtered\_dict) + 1

cols = 3

rows = (num\_images + cols - 1) // cols

fig, axes = plt.subplots(rows, cols, figsize=(cols \* 4, rows \* 4))

if num\_images <= 1:

ax = np.array([axes])

elif rows == 1 and cols == 1:

axes = np.array([[axes]])

elif rows == 1 or cols == 1:

axes = axes.reshape(rows, cols)

if axes is None:

print("Error: axes is None.")

return

ax = axes.flat

ax[0].imshow(original, cmap='gray' if original.ndim == 2 else None)

ax[0].set\_title("Orijinal")

ax[0].axis('off')

i = 1

for title, img in filtered\_dict.items():

if i < len(ax):

img\_cmap = cmap if img.ndim == 2 else None

im = ax[i].imshow(img, cmap=img\_cmap)

ax[i].set\_title(title)

ax[i].axis('off')

if 'Orientation' in title:

fig.colorbar(im, ax=ax[i], fraction=0.046, pad=0.04)

i += 1

else:

break

for j in range(i, len(ax)):

ax[j].axis('off')

fig.suptitle(main\_title, fontsize=16)

plt.tight\_layout(rect=[0, 0.03, 1, 0.95])

if save\_path:

plt.savefig(save\_path, bbox\_inches='tight')

print(f"Grafik kaydedildi: {save\_path}")

plt.close(fig)

else:

plt.show()

image\_names = ['camera', 'moon', 'coins']

images\_to\_process = {name: getattr(data, name)() for name in image\_names}

# a

print("--- (a): Konvolüsyon Filtreleri")

for name, img in images\_to\_process.items():

print(f"\nİşlenen Görüntü: {name}")

filtered\_a = {}

for k in [3, 5]:

try:

filtered\_a[f'Box (k={k})'] = apply\_convolution(img, filter\_type='box', k\_size=k)

filtered\_a[f'Gaussian (k={k})'] = apply\_convolution(img, filter\_type='gaussian', k\_size=k)

except Exception as e:

print(f" Hata ({name}, k={k}): {e}")

try:

filtered\_a[f'Laplacian (k=3)'] = apply\_convolution(img, filter\_type='laplacian', k\_size=3)

except Exception as e:

print(f" Hata (Laplacian, {name}): {e}")

if filtered\_a:

output\_filename = os.path.join(OUTPUT\_DIR, f"{name}\_a\_convolution.png")

plot\_comparison(img, filtered\_a, f"{name} - Konvolüsyon Filtreleri", save\_path=output\_filename)

# (b): Gradyanlar

print("\n--- (b): Gradyanlar ---")

for name, img in images\_to\_process.items():

print(f"\nİşlenen Görüntü: {name}")

try:

gx, gy, mag, ori = calculate\_gradients(img)

filtered\_b = {

'Gradient X (Sobel\_V)': gx,

'Gradient Y (Sobel\_H)': gy,

'Magnitude': mag,

'Orientation': ori

}

cmaps = ['gray', 'gray', 'gray', 'hsv']

num\_images = len(filtered\_b) + 1

cols = 3

rows = (num\_images + cols - 1) // cols

fig, axes = plt.subplots(rows, cols, figsize=(cols \* 5, rows \* 4))

if num\_images <= 1: ax = np.array([axes])

elif rows == 1 and cols == 1: axes = np.array([[axes]])

elif rows == 1 or cols == 1: axes = axes.reshape(rows, cols)

if axes is None:

print("Error: axes is None.")

continue

ax = axes.flat

ax[0].imshow(img, cmap='gray' if img.ndim == 2 else None)

ax[0].set\_title("Orijinal")

ax[0].axis('off')

i = 1

for title, filtered\_img in filtered\_b.items():

if i < len(ax):

im = ax[i].imshow(filtered\_img, cmap=cmaps[i-1])

ax[i].set\_title(title)

ax[i].axis('off')

fig.colorbar(im, ax=ax[i], fraction=0.046, pad=0.04)

i += 1

else: break

for j in range(i, len(ax)): ax[j].axis('off')

fig.suptitle(f"{name} - Gradyanlar", fontsize=16)

plt.tight\_layout(rect=[0, 0.03, 1, 0.95])

output\_filename\_b = os.path.join(OUTPUT\_DIR, f"{name}\_b\_gradients.png")

plt.savefig(output\_filename\_b, bbox\_inches='tight')

print(f"Grafik kaydedildi: {output\_filename\_b}")

plt.close(fig)

except Exception as e:

print(f"Gradyan hesaplama/kaydetme hatası ({name}): {e}")

# (c): Kenar Belirleme Filtreleri

print("\n--- (c) Kenar Belirleme Filtreleri ---")

for name, img in images\_to\_process.items():

print(f"\nİşlenen Görüntü: {name}")

try:

edges\_c = apply\_edge\_detectors(img, log\_sigma=1.5, canny\_sigma=1.5)

output\_filename\_c = os.path.join(OUTPUT\_DIR, f"{name}\_c\_edge\_detection.png")

plot\_comparison(img, edges\_c, f"{name} - Kenar Belirleme Filtreleri", cmap='gray', save\_path=output\_filename\_c)

except Exception as e:

print(f"Kenar belirleme/kaydetme hatası ({name}): {e}")

print(f"\nİşlem Tamamlandı. Çıktılar '{OUTPUT\_DIR}' dizinine kaydedildi.")

Ek : Çıktılar

1. **a bölümü için**

metin, adam, insan, siyah beyaz, ekran görüntüsü içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Şekil 1.1) camera görüntüsüne uygulanan konvolüsyon filtreleri

ekran görüntüsü, kalıp, desen, düzen, dikdörtgen, kare içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Şekil 1.2) coins görüntüsüne uygulanan konvolüsyon filtreleri

ekran görüntüsü, doğa, krater içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Şekil 1.3) moon görüntüsüne uygulanan konvolüsyon filtreleri

1. b bölümü için

metin, ekran görüntüsü, diyagram, tasarım içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Şekil 2.1) camera görüntüsünün gradyanları

ekran görüntüsü, tasarım içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Şekil 2.2) coins görüntüsünün gradyanları

ekran görüntüsü içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Şekil 2.3) moon görüntüsünün gradyanları

1. c bölümü çıktıları

ekran görüntüsü içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Şekil 3.1) camera görüntüsü için kenar belirleme filtreleri

ekran görüntüsü, kalıp, desen, düzen, tasarım içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Şekil 3.2) coins görüntüsü için kenar belirleme filtreleri

ekran görüntüsü, Astronomi nesnesi, astronomi, evren, kainat içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Şekil 3.3) moon görüntüsü için kenar belirleme filtreleri