TUTORIAL 6

Feature Extraction

Pengolahan Citra - Semester Gasal 2023/2024

1 Pendahuluan

Feature Extraction adalah teknik-teknik yang dapat digunakan untuk mengekstraksi fitur dari sebuah citra. Fitur-fitur ini selanjutnya dapat diproses untuk task tertentu, contohnya untuk melakukan klasifikasi citra [1]. Mahasiswa dapat meng-eksplor untuk fitur-fitur yang lain karena pada tutorial ini hanya subset dari banyak fitur.

2 Beberapa Contoh Features

2.1 Image Histogram

Kita dapat menggunakan piksel-piksel yang terdapat pada citra sebagai fitur. Untuk memudahkan bagaimana cara kita melihat bagaimana piksel-piksel dapat berfungsi sebagai fitur, kita dapat menggunakan histogram untuk meneliti bagaimana cara kita menggunakan fitur tersebut.

Misalkan kita ingin mengklasifikasikan apakah citra *grayscale* yang diberikan merupakan kucing hitam atau putih. Kita bisa menampilkan terlebih dahulu histogram untuk citra kucing hitam dan putih.

```
from skimage import io
import matplotlib.pyplot as plt

# Read image
black = io.imread('black.jpeg')
white = io.imread('white.jpeg')

# Show image and histogram
plt.figure(figsize=(20,10))
plt.subplot(2, 2, 1)
plt.title('Black Cat Image')
plt.imshow(black)
plt.axis('off')
plt.subplot(2, 2, 2)
plt.title('White Cat Image')
plt.imshow(white)
plt.axis('off')
plt.subplot(2, 2, 3)
plt.title('Black Cat Histogram')
plt.hist(black.flatten(), 256, range=(0,255))
plt.axis('off')
plt.subplot(2, 2, 4)
plt.title('White Cat Histogram')
plt.hist(white.flatten(), 256, range=(0,255))
plt.axis('off')
plt.show()
```

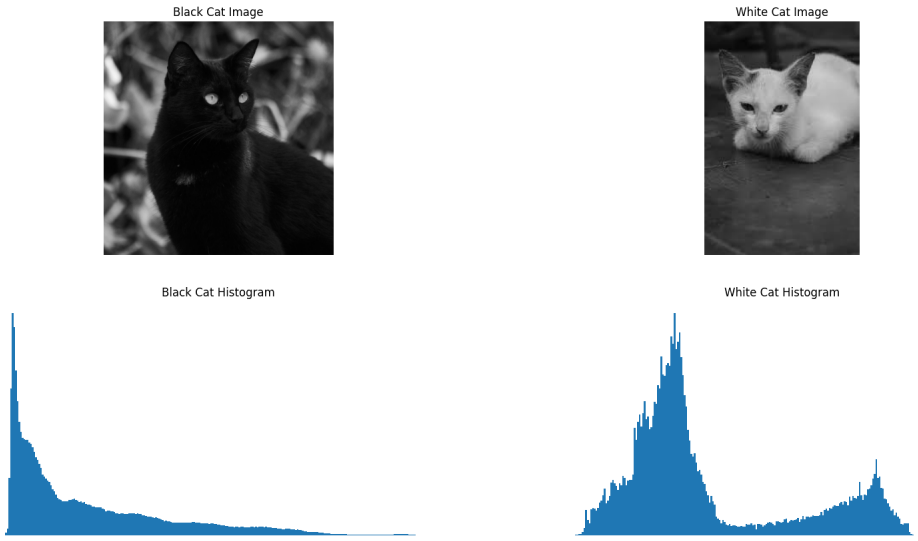

Figure 1: Perbandingan Kucing Hitam dan Putih

Dapat kita lihat dari hasil histogram yang ditampilkan di atas bahwa untuk citra kucing putih memiliki distribusi yang cenderung ke kanan (intensitas lebih tinggi) dibanding citra kucing hitam yang distribusi histogram cenderung ke kiri (intensitas lebih rendah).

Dari fitur ini kita bisa membuat sebuah model untuk memprediksi apakah citra yang diberikan merupakan kucing hitam atau putih. Berikut adalah contoh cara untuk memprediksi apakah citra merupakan kucing hitam atau putih.

```
import numpy as np

def predict(file_name):
    img = io.imread(file_name)
    frequency_count = np.bincount(img.flatten())
    cum_sum = np.cumsum(frequency_count)


  # lebih banyak intensitas rendah, prediksi sebagai kucing
    hitam
    if cum_sum[50] > (cum_sum[-1] - cum_sum[50]):
        return 'black'
    else:
        return 'white'

filename = 'black.jpeg'
print(f'image {filename} predicted as {predict(filename)} cat')
filename = 'white.jpeg'
print(f'image {filename} predicted as {predict(filename)} cat')
```

Apabila kode tersebut dijalankan akan didapatkan hasil sebagai berikut.

```
image black.jpeg predicted as black cat
```

```
image white.jpeg predicted as white cat
```

Contoh di atas adalah salah satu contoh yang menggunakan histogram pada citra *grayscale*. Tetapi penggunaan Image Histogram ini tidak hanya berfokus pada histogram *grayscale*. Terdapat banyak variasi histogram lain yang dapat digunakan seperti menggunakan color histogram RGB/HSI, histogram of gradients, dan lain sebagainya.

2.2 Image Segmentation

Dengan melakukan segmentasi terlebih dahulu pada citra. Maka kita bisa mendapatkan fitur-fitur berupa banyak objek, bentuk objek yang relevan, dan masih banyak lagi. Silakan lihat lab image segmentation untuk informasi detilnya.

2.3 Fourier Transform

Seperti yang telah dipelajari pada lab sebelumnya, kita dapat menggunakan frekuensi hasil fourier transform untuk dijadikan sebagai fitur. Silakan lihat lab image processing frequency domain untuk informasi detilnya.

2.4 Hough Transform

Hough Transformation adalah algoritma yang dapat mendeteksi bentuk-bentuk pada citra selama bentuk tersebut dapat direpresentasikan dalam garis lurus. Untuk melakukan Hough Transformation, kita harus mencari terlebih dahulu edge-edge pada citra tersebut. Hal ini bisa dicapai dengan menggunakan algoritma Canny.

```
import numpy as np
import cv2
from matplotlib import pyplot as plt
from skimage import io, color

img = io.imread('calendar.jpeg')
original_image = np.copy(img)

# convert image to gray
gray = (color.rgb2gray(img) * 255).astype(np.uint8)

# find all of the edges using canny
edges = cv2.Canny(gray, 50, 150, apertureSize=3)
# find all of the lines using hough transformation
lines = cv2.HoughLines(edges, 1, np.pi/180, 200)

# plotting line
for line in lines:
    for rho,theta in line:
        a = np.cos(theta)
        b = np.sin(theta)
```

```
        x0 = a*rho
        y0 = b*rho
        x1 = int(x0 + 1000*(-b))
        y1 = int(y0 + 1000*(a))
        x2 = int(x0 - 1000*(-b))
        y2 = int(y0 - 1000*(a))
        cv2.line(img, (x1,y1), (x2,y2), (255,0,0), 2)

# Show image
plt.figure(figsize=(20,10))
plt.subplot(1,2,1)
plt.title('Original Image')
plt.imshow(original_image)
plt.axis('off')
plt.subplot(1,2,2)
plt.title('After Hough Transfromation')
plt.imshow(img,)
```

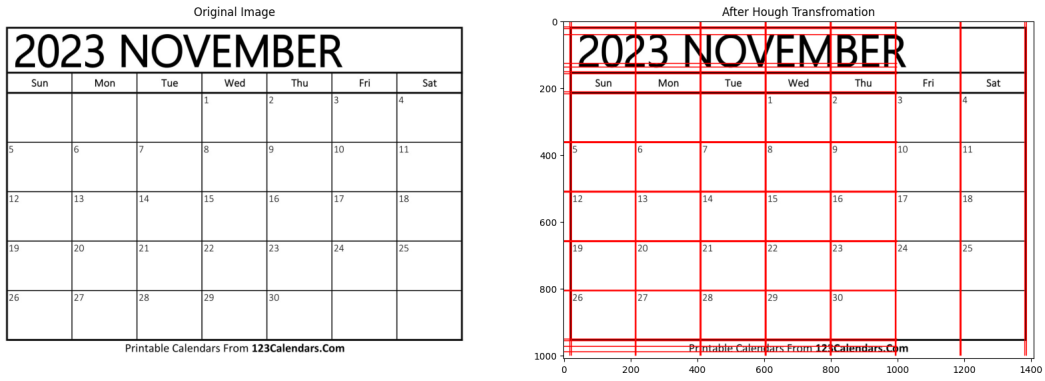

Figure 2: Hough Transform

Hasil dari hough transform dengan library OpenCV di atas adalah daftar garis atau edge yang dideteksi dari citra. Dengan informasi ini, kita dapat mengekstrak nilai-nilai fitur sesuai kebutuhan, misal jumlah garis/edge, bentuk edge, manipulasi aritmetika dari edge (misal hitung luas, walau ini lebih mudah dengan segmentasi region), dan lainnya. Pemilihan nilai yang diekstrak bergantung pada karakteristik yang dibutuhkan dan task yang dikerjakan (misal klasifikasi citra) dan perlu dicoba secara empiris.

2.5 Image Morphology

Mathematical morphology dapat digunakan juga untuk melakukan ekstraksi fitur dari sebuah gambar. Image morphology dapat dilakukan dengan dilasi, erosi, dan lain-lain seperti yang telah dibahas pada lab morphological image processing.

2.6 Template Matching

Template matching adalah metode untuk mencari dan menemukan bagian kecil dari sebuah citra sesuai dengan template yang diinginkan. Template tersebut akan berperan sebagai jendela yang akan digeser sepanjang citra (konvolusi) dan mencocokkan template tersebut dengan bagian citra yang tercover oleh template. Berikut contoh template matching menggunakan OpenCV.

```
from skimage import io, util, color
from cv2 import matchTemplate, TM_CCOEFF_NORMED, rectangle
import numpy as np
import matplotlib.pyplot as plt

# Read image
image = util.img_as_ubyte(io.imread("pertandingan.jpeg"))
template = util.img_as_ubyte(io.imread("bola.jpg"))

# To gray
image_gray = util.img_as_ubyte(color.rgb2gray(image))
template_gray = util.img_as_ubyte(color.rgb2gray(template))

# Get height and width of template
h, w = template_gray.shape

# Match template using cv2.matchTemplate
result = matchTemplate(image_gray, template_gray,
    TM_CCOEFF_NORMED)

# Copy image for bounding box
matched = image.copy()

# Thresholding
threshold = 0.5
loc = np.where(result >= threshold)

# Create bounding box
for point in zip(*loc[::-1]):
    rectangle(matched, point, (point[0] + w, point[1] + h),
        (255,0,0), 2)

# Show image
plt.figure(figsize=(20,10))
plt.subplot(131),plt.imshow(image)
plt.title('Original'), plt.xticks([]), plt.yticks([])
plt.subplot(132),plt.imshow(template)
plt.title('Template'), plt.xticks([]), plt.yticks([])
```

```
plt.subplot(133),plt.imshow(matched)
plt.title('Template Matched'), plt.xticks([]), plt.yticks([])
plt.yticks([])
plt.show()
```

Figure 3: Template Matching

References

[1] A.M. Arymurthy and L. Rahadianti. Image transform and mathematical morphology. PPT Pengcit.