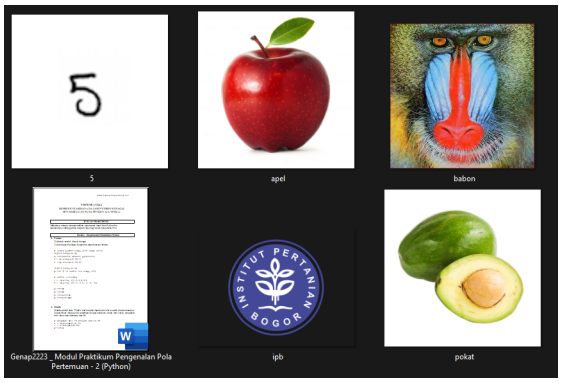
Representasi Data dan Pre-Proses Dasar Menggunakan Python dan *library* OpenCV, Numpy, Pandas dan Matplotlib

Wildan Fajri Alfarabi G64190060

Representasi Data

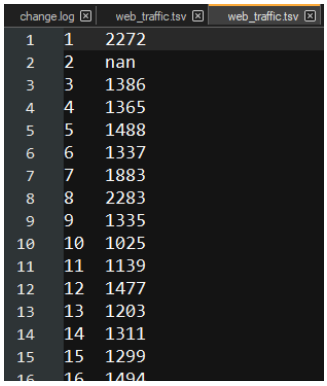
Data yang digunakan adalah data citra bertipe png dan jpg, serta data numeric web_traffic yang merupakan besaran data atau *byte* yang dikirimkan pengguna web dan diterima oleh server web. Berikut adalah *preview* dari data-data yang digunakan :

- Data citra



Data tersebut di import kedalam program python. Data-data tersebut memiliki representasi berbeda-beda pada program python. Untuk data citra, program python menyimpan data tersebut dengan representasi list/array yang bernilai *value* dari tiap piksel, untuk citra RGB memiliki array berukuran NxMx3 dimana NxM adalah dimensi panjang dan lebar citra serta 3 adalah dimensi untuk masing masing *colorspace* pada citra. Sementara untuk citra keabuan, memiliki dimensi array/list NxM.

- Data numerik



Untuk data numerik web_traffic, direpresentasikan dengan dataframe dari pandas, yang memiliki 2 kolom untuk nomor data, dan *byte* yang dikirim pengguna. Baris yang dimiliki dataframe ini sebanyak 743 baris sebagai *record* tiap data dari web_traffic.

Pre-Processing Data

- **Data Numerik, Nomina atau String**

Data numerik dibersihkan dengan cara menghilangkan nilai *MissingValue* atau NaN pada data, yang akan diisikan dengan nilai mean-nya yang bernilai 1962,165986.

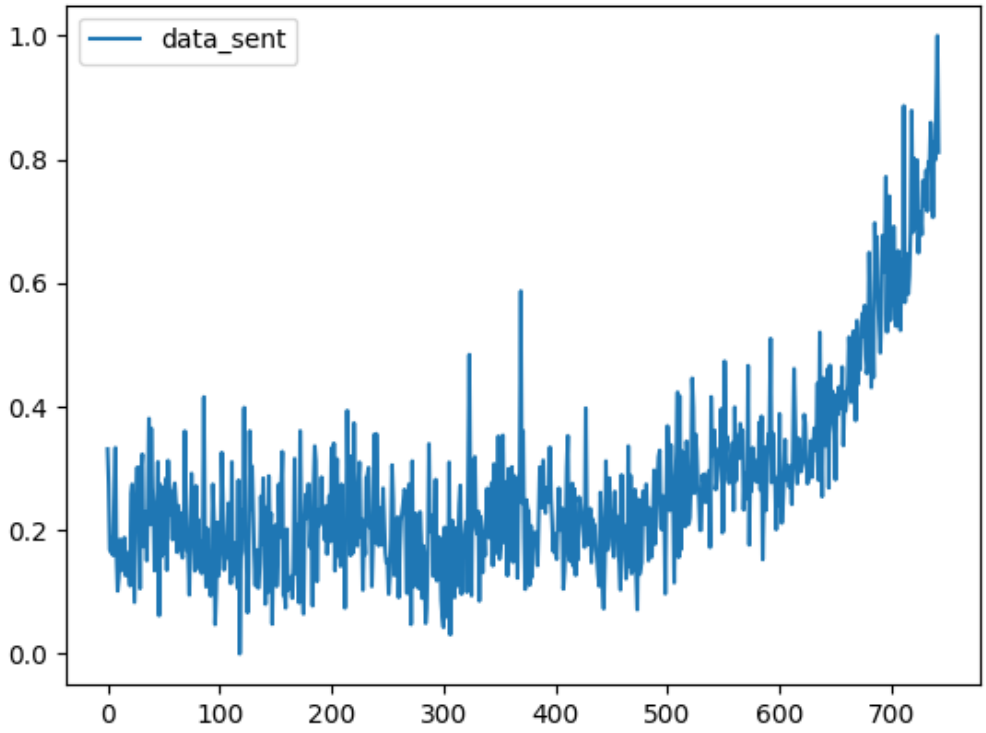
	times	data_sent
0	1	2272.000000
1	2	1962.165986
2	3	1386.000000
3	4	1365.000000
4	5	1488.000000
5	6	1337.000000
6	7	1883.000000
7	8	2283.000000
8	9	1335.000000
9	10	1025.000000

Setelah MissingValue dihilangkan, selanjutnya dilakukan normalisasi dengan MinMax Normalization agar rentang menjadi [0-1], berikut adalah rumus dan juga hasilnya.

```
table["data_sent"] = (table["data_sent"] -
          table["data_sent"].min()) / (
                 table["data_sent"].max()
          -table["data_sent"].min())
```

	times	data_sent
0	1	0.331248
1	2	0.274230
2	3	0.168200
3	4	0.164336
4	5	0.186971
5	6	0.159183
6	7	0.259661
7	8	0.333272
8	9	0.158815
9	10	0.101767

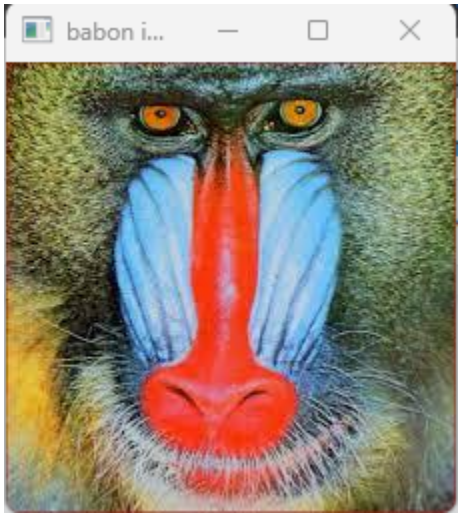
Setelah itu, data yang telah dilakukan praproses dapat divisualisasikan dengan histogram.



Dapat dilihat semakin berjalannya waktu (axis-x) besaran data yang dikirimkan pengguna ke server semakin tinggi.

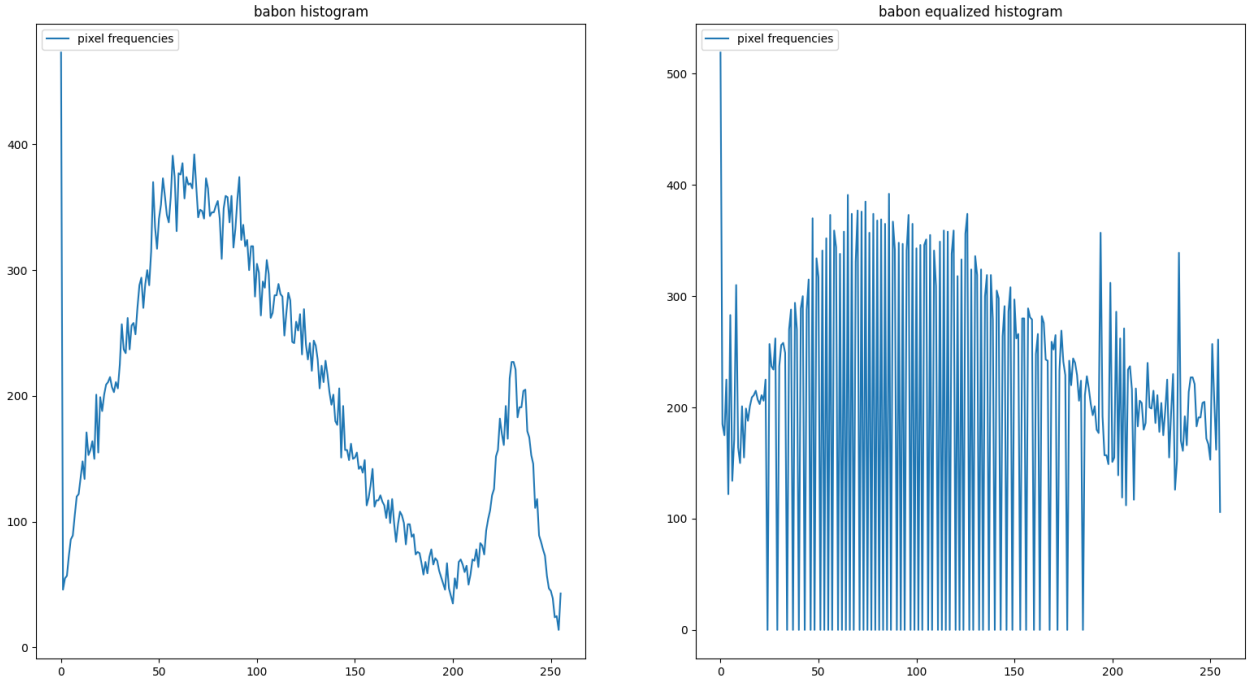
- **Data Citra**

Data citra di import menggunakan opencv lalu dipreview untuk menunjukan apakah data citra berhasil di import ke dalam variabel atau tidak.

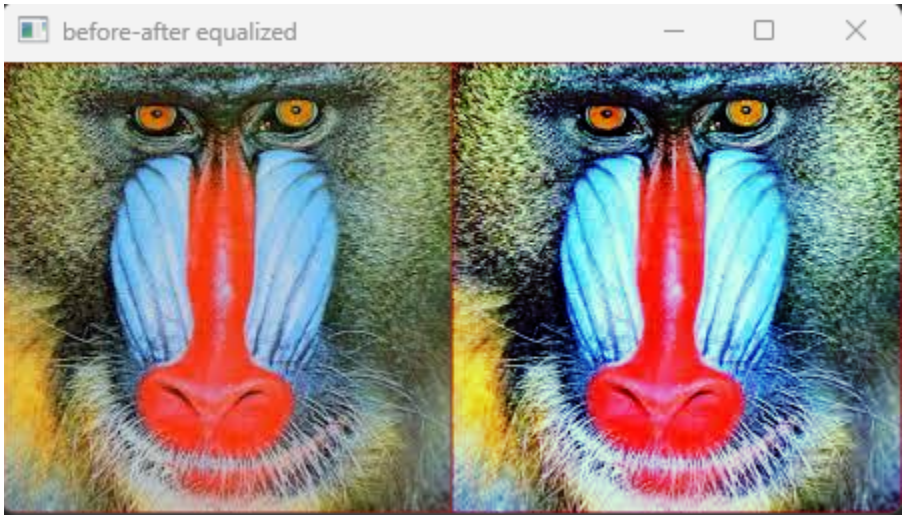


Setelah diimport, dapat dilakukan berbagai pra-proses data contohnya adalah equalisasi histogram citra dan juga segmentasi objek dari latar belakang citra.

Equalisasi citra dilakukan agar citra yang awalnya tidak kontras menjadi lebih kontas dengan melebarkan histogram citranya. Berikut ini adalah histogram citra sebelum dan sesudah equalisasi.



Histogram citra setelah dilakukan equalisasi menjadi lebih lebar, yang awalnya pada rentang nilai 150-220 memiliki frequensi yang sedikit, menjadi lebih seimbang frequensinya. Sehingga citra yang dihasilkan menjadi lebih kontras, berikut adalah hasil citra setelah dilakukan equalisasi.



Selanjutnya adalah segmentasi objek dari latar belakang citra, tahapan yang dilakukan adalah *split* masing-masing *colorspace*, lalu pilih salah satu *colorspace* yang paling kontras menunjukan perbedaan objek dengan latar belakang. Sebelum melakukan *splitting*, dapat dilakukan transformasi ke *colorspace* yang lain seperti CMYK apabila dirasa *colorspace* awal tidak

menunjukan perbedaan kontras objek dengan latar belakang citra. *Colorspace* yang baik untuk segmentasi adalah *colorspace* yang memiliki pola histogram objek berbeda dengan latar citra. Pada citra apel, *colorspace* merah memiliki pola yang signifikan antara objek dengan latar, karena apel sendiri mayoritas memiliki warna merah.

Oleh karena itu dipilih *colorspace* merah. Setelah itu dilakukan inverse binary thresholding untuk mengubah nilai pixel yang kurang dari threshold menjadi 1, dalam kasus ini adalah objek apel. Setelah dilakukan thresholding terhadap seluruh pixel citra, didapatkan mask untuk apel. Apabila mask masih memiliki lubang didalam objek, dapat dilakukan transformasi morfologi closing dan opening terhadap objek. Tahap terakhir adalah melakukan operasi bitwise AND masing masing pixel citra asli dengan pixel mask untuk mensegmentasi objek apel dengan latar belakang. Citra ini apabila disimpan kedalam tipe png, maka latar citra akan menjadi transparan.

Code Program :

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd


def importImage(imageName,title="Image"):
    data = cv2.imread(imageName)
    cv2.imshow(title, data)
    cv2.waitKey(0)
    return data


def line_chart(data, data_classes):
    fig, (ax) = plt.subplots(1, 1)
    ax.plot(data[data_classes], label=data_classes)
    ax.legend(loc="upper left")
    plt.show()

def numeric_preprocessing(table):
    table.columns = ["times", "data_sent"]

    mean_data_send = np.mean(table["data_sent"])
    table.data_sent[np.isnan(table["data_sent"])] =
mean_data_send
    print(table.head(10), "\n")

    table["data_sent"] = (table["data_sent"] -
table["data_sent"].min()) / (
                table["data_sent"].max()
-table["data_sent"].min())

    print(table.head(10))
```

```
    line_chart(table, "data_sent")

def img_hist_eq(img):
    R, G, B = cv2.split(img)
    output1_R = cv2.equalizeHist(R)
    output1_G = cv2.equalizeHist(G)
    output1_B = cv2.equalizeHist(B)
    img_equalized = cv2.merge((output1_R, output1_G,
output1_B))

    res = np.hstack((img, img_equalized))

    cv2.imshow("before-after equalized", res)
    cv2.waitKey(0)

    babon_hist = cv2.calcHist([img],[0],None,[256],[0,256])
    babon_equ_hist = cv2.calcHist([img_equalized], [0], None,
[256], [0, 256])

    fig, (ax1,ax2) = plt.subplots(1, 2)
    ax1.plot(babon_hist, label="pixel frequencies")
    ax1.set_title("babon histogram")
    ax1.legend(loc="upper left")
    ax2.plot(babon_equ_hist, label="pixel frequencies")
    ax2.set_title("babon equalized histogram")
    ax2.legend(loc="upper left")
    plt.show()

def img_bg_segmentation(img):
    R, G, B = cv2.split(img)
    images = [R, G, B]
    titles = ["red", "green", "blue"]
    for i in range(3):
        plt.subplot(1, 3, i + 1), plt.imshow(images[i], 'gray',
```

```
vmin=0, vmax=255)
        plt.title(titles[i])
        plt.xticks([]), plt.yticks([])
    plt.show()

    ret, mask = cv2.threshold(R, 240, 255,
cv2.THRESH_BINARY_INV)

    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3,
3))
    resultR = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel,
iterations=10)

    image_segmented_from_bg = cv2.bitwise_and(img, img,
mask=resultR)
    cv2.imshow("segmented", image_segmented_from_bg)
    cv2.waitKey(0)

if __name__ == "__main__":
    #Numeric preprocessing
    webtraffic = pd.read_table("web_traffic.tsv", header=None)
    numeric_preprocessing(webtraffic)

    #Image Hist Equalization
    img_hist = importImage("babon.jpg","babon image")
    img_hist_eq(img_hist)

    #IMG BG SEGMENTATION
    img_segmentation = importImage("apel.jpg","apel image")
    img_bg_segmentation(img_segmentation)
```