ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ №1

за курсом «Алгоритми аналізу та методи обробки зображень»

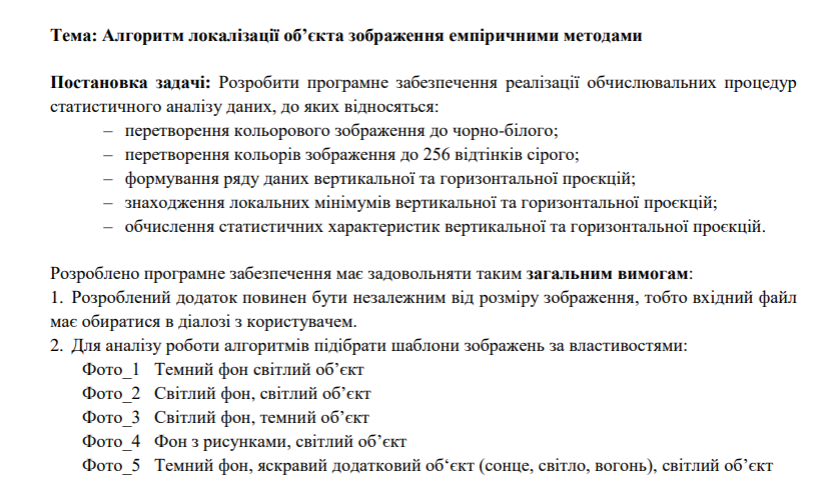
студента групи ПК-24М-1

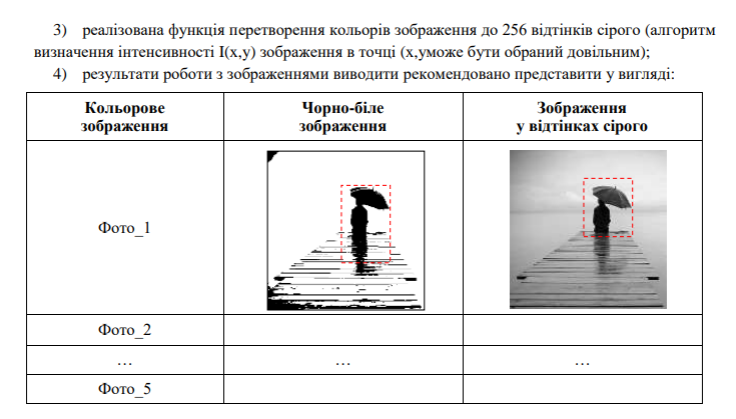
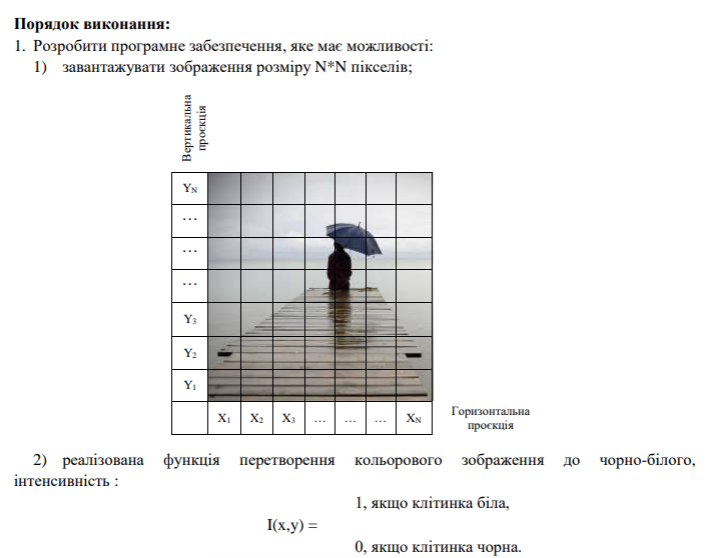
Афанасьєва Дениса

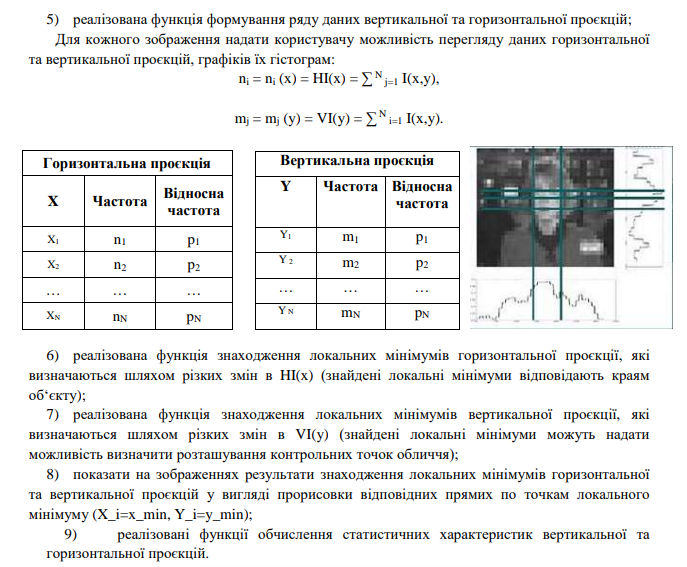
кафедра комп’ютерних технологій, ДНУ

2024/2025 н.р.

Постановка задачі







Опис розв’язку

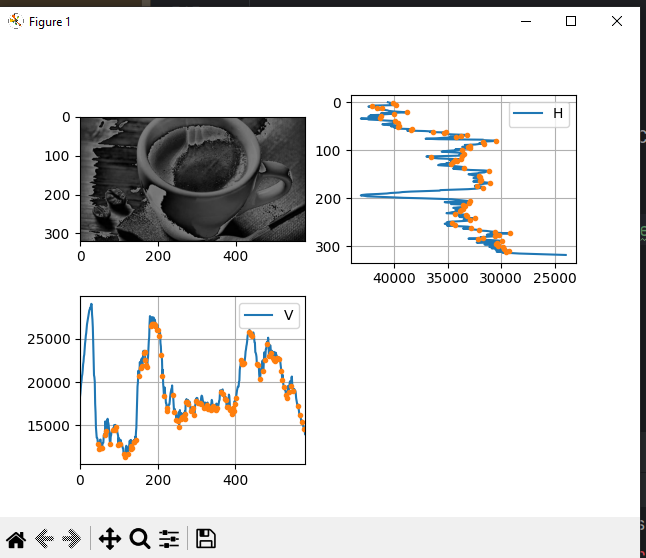
Використовував мову програмування Python, з використанням бібліотек OpenCV, NumPy, Matplotlib та SciPy.

Реалізував можливості:

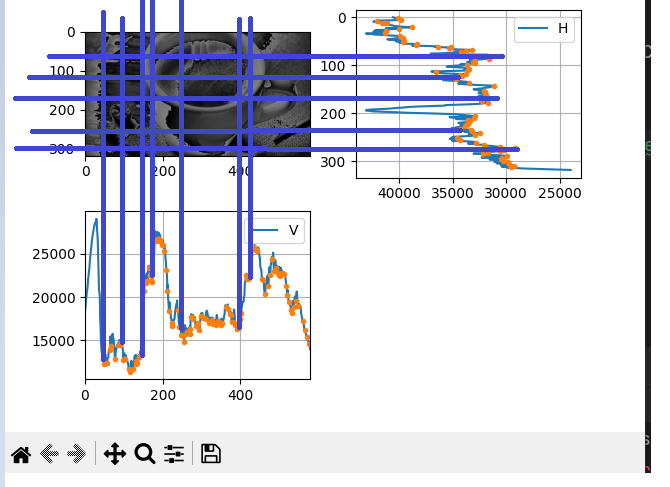
* Завантажувати зображення довільного розміру
* Функція перетворення кольорового зображення до чорно-білого
* Функція перетворення кольорів зображення до 256 відтінків сірого
* Функція формування ряду данних вертикальної та горизонтальної проекції
* Функція знаходження локальних мінімумов для вертикальної та горизонтальної проекції
* Функція обчислення статичних характеристик вертикальної та горизонтальної проекцій.

У результаті застосування програми до фото, отримав такі результати:

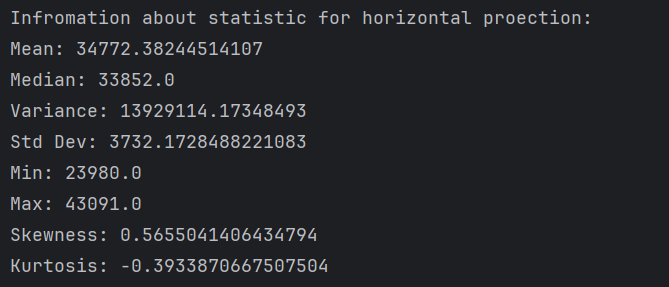
|  |  |  |
| --- | --- | --- |
| Кольорове  зображення | Чорно-біле  зображення | Зображення у відтінках сірого |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |



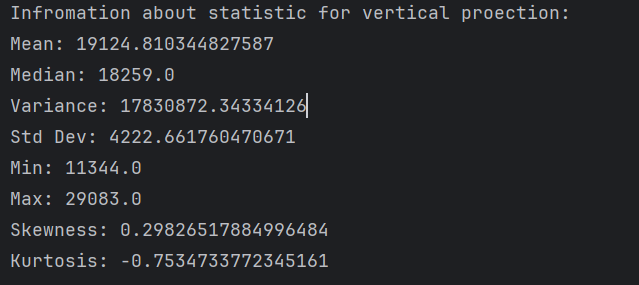
Зображення у сірих відтінках, графіки горизонтальної та вертикальної проекцій та локальні мінімуми на них.



Результати знаходження локальних мінімумов вертикальної та горизонтальної проекції



Результати обчислення статистичних характеристик горизонтальної проекції



Результати обчислення статистичних характеристик вертикальної проекції

Код програми

from \_\_future\_\_ import annotations  
from abc import ABC, abstractmethod  
  
import cv2  
import numpy as np  
from enum import Enum  
import matplotlib.pyplot as plt  
from scipy.stats import skew, kurtosis  
  
  
class MyFig():  
  
 def \_\_init\_\_(self, r: int, c: int):  
 *"""  
 :param r: count of rows on figure  
 :param c: count of colums on figure  
 """* self.fig, self.ax = plt.subplots(r, c, gridspec\_kw={'height\_ratios': [1 for i in range(r)],  
 'width\_ratios': [1 for i in range(c)]})  
  
 def get\_ax(self, i: int, j: int):  
 return self.ax[i, j]  
  
 def show(self):  
 plt.show()  
  
  
def open\_img(path: str = "") -> np.ndarray:  
 path\_to\_file = path or input("Enter path to image: \t")  
 img = cv2.imread(path\_to\_file, cv2.IMREAD\_COLOR)  
 if img is None:  
 print("Can't open image:(")  
 return None  
  
 return img  
  
  
def plot(dataset: np.ndarray, title: str = "", xlabel: str = "", ylabel: str = "") -> None:  
 plt.plot(range(dataset.size), dataset)  
 plt.title = title  
 plt.xlabel = xlabel  
 plt.ylabel = ylabel  
 plt.grid()  
 plt.show()  
  
  
def unite\_plot\_vp\_and\_hp(img: np.ndarray, hp: np.ndarray, vp: np.ndarray) -> None:  
 plt.subplot(2, 2, 1)  
 plt.imshow(img)  
  
 plt.subplot(2, 2, 2)  
 plt.plot(hp, range(hp.size))  
 plt.gca().invert\_xaxis()  
 plt.gca().invert\_yaxis()  
 # plt.plot(range(hp.size), hp)  
 plt.legend("Horizontal projection")  
 plt.grid()  
  
 plt.subplot(2, 2, 3)  
 plt.plot(range(vp.size), vp)  
 plt.legend("Vertical projection")  
 plt.xlim((0, vp.size))  
 plt.grid()  
  
 plt.show()  
  
  
def unite\_plot\_new(fig: MyFig, img: np.darray, hp: np.ndarray, vp: np.ndarray) -> MyFig:  
 ax = fig.get\_ax(0, 0)  
 ax.imshow(img)  
  
 ax = fig.get\_ax(0, 1)  
 ax.plot(hp, range(hp.size))  
 ax.invert\_xaxis()  
 ax.invert\_yaxis()  
 ax.legend("H")  
 ax.grid()  
  
 ax = fig.get\_ax(1, 0)  
 ax.plot(range(vp.size), vp)  
 ax.legend("Vertical projection")  
 ax.set\_xlim((0, vp.size))  
 ax.grid()  
  
 ax = fig.get\_ax(1, 1)  
 ax.axis("off")  
  
 return fig  
  
  
def add\_minimums\_to\_ax(fig: MyFig, array: np.ndarray, i: int, j: int, is\_inverted: bool = False) -> MyFig:  
 *"""  
 :param fig: figure on which data is shown  
 :param array: array of data to show  
 :param i: row num of subplot  
 :param j: column num of subplot  
 :param is\_inverted: True - if inverted, otherwise False  
 :return: MyFig  
 """* ax = fig.get\_ax(i, j)  
 nums = local\_minimum\_numlist(array)  
 list\_of\_values = [array[i] for i in nums]  
  
 if is\_inverted:  
 ax.plot(list\_of\_values, nums, linestyle='', marker='.')  
 else:  
 ax.plot(nums, list\_of\_values, linestyle='', marker='.')  
  
 return fig  
  
  
def color\_to\_bw(img: np.ndarray, color\_component: int = 0) -> np.ndarray:  
 *"""  
 color\_component:  
 0 - red  
 1 - green  
 2 - blue  
 """* img\_gray = img.copy()  
 match (color\_component):  
 case 0:  
 for i in range(img\_gray.shape[0]):  
 for j in range(img\_gray.shape[1]):  
 (\_, \_, r) = img\_gray[i, j]  
 gray = 0 if r <= 256 / 2 else 255  
 img\_gray[i, j] = gray  
 case 1:  
 for i in range(img\_gray.shape[0]):  
 for j in range(img\_gray.shape[1]):  
 (\_, g, \_) = img\_gray[i, j]  
 gray = 0 if g <= 256 / 2 else 255  
 img\_gray[i, j] = gray  
 case 2:  
 for i in range(img\_gray.shape[0]):  
 for j in range(img\_gray.shape[1]):  
 (b, \_, \_) = img\_gray[i, j]  
 gray = 0 if b <= 256 / 2 else 255  
 img\_gray[i, j] = gray  
  
 return img\_gray  
  
  
def color\_to\_gray(img: np.ndarray) -> np.ndarray:  
 img\_gray = img.copy()  
  
 for i in range(img\_gray.shape[0]):  
 for j in range(img\_gray.shape[1]):  
 (r, g, b) = img\_gray[i, j, ::-1]  
 gray = int((max(r, g, b) + min(r, g, b)) / 2)  
 img\_gray[i, j] = gray % 255  
  
 return img\_gray  
  
  
def compute\_statistics(projection: np.ndarray) -> dict:  
 stats = {  
 "Mean": np.mean(projection),  
 "Median": np.median(projection),  
 "Variance": np.var(projection),  
 "Std Dev": np.std(projection),  
 "Min": np.min(projection),  
 "Max": np.max(projection),  
 "Skewness": skew(projection),  
 "Kurtosis": kurtosis(projection)  
 }  
 return stats  
  
  
def vertical\_proection(img: np.ndarray) -> np.ndarray:  
 if img.size == img.shape[0] \* img.shape[1]:  
 v\_pr = np.zeros(img.shape[1])  
 for i in range(img.shape[1]):  
 for j in range(img.shape[0]):  
 v\_pr[i] += img[j, i]  
  
 v\_pr = v\_pr.astype(int)  
 else:  
 arr1 = vertical\_proection(img[:, :, 0])  
 arr2 = vertical\_proection(img[:, :, 1])  
 arr3 = vertical\_proection(img[:, :, 2])  
 v\_pr = np.zeros((arr1.size, 3))  
 for i in range(arr1.size):  
 v\_pr[i, 0] = arr1[i]  
 v\_pr[i, 1] = arr2[i]  
 v\_pr[i, 2] = arr3[i]  
  
 return v\_pr  
  
  
# produce matrix with rows HEIGHT (img.shape[1]) and cols (img.shape[2])  
# if its 3 colored, then colors will be in (b, g, r)  
def horizontal\_proection(img: np.ndarray) -> np.ndarray:  
 if img.size == img.shape[0] \* img.shape[1]:  
 h\_pr = np.zeros(img.shape[0])  
 for i in range(img.shape[0]):  
 for j in range(img.shape[1]):  
 h\_pr[i] += img[i, j]  
  
 h\_pr = h\_pr.astype(int)  
 else:  
 arr1 = horizontal\_proection(img[:, :, 0])  
 arr2 = horizontal\_proection(img[:, :, 1])  
 arr3 = horizontal\_proection(img[:, :, 2])  
 h\_pr = np.zeros((arr1.size, 3))  
 for i in range(arr1.size):  
 h\_pr[i, 0] = arr1[i]  
 h\_pr[i, 1] = arr2[i]  
 h\_pr[i, 2] = arr3[i]  
  
 return h\_pr  
  
  
# Send ONLY one dimensional array  
# Produce list of nums of minimum, that is less than average value  
def local\_minimum\_list(img: np.ndarray) -> list:  
 lminimums = list()  
 for i in range(1, img.size):  
 lminimums.append(img[i] - img[i - 1])  
  
 average = max(lminimums) - min(lminimums)  
 print("average\_val: ", average)  
  
 def lminimumsnum(average: float, lminimums: list) -> list:  
 lminimumsnum = list()  
 for i in range(len(lminimums)):  
 if lminimums[i] <= average:  
 lminimumsnum.append(i)  
  
 return lminimumsnum  
  
 return lminimumsnum(average, lminimums)  
  
  
def local\_minimum\_numlist(array: np.ndarray) -> list:  
 *"""  
 :param array: one dimensional array of any projection  
 :return: list with element nums of minimums  
 """* res = list()  
 for i in range(1, array.shape[0] - 1):  
 if array[i] < array[i - 1] and array[i] < array[i + 1]:  
 res.append(i)  
  
 return res  
  
  
class MODE(Enum):  
 TEST = 0  
 RUN = 1  
  
  
def client(mode: MODE, img\_name: str) -> None:  
 if mode == MODE.TEST:  
 img = open\_img(img\_name)  
 else:  
 img = open\_img()  
  
 if img is None:  
 return  
  
 if IS\_SHOW\_ORIGINAL\_IMG:  
 cv2.imshow("Your image", img)  
 cv2.waitKey(0)  
  
 bw\_img = color\_to\_bw(img, 2)  
 gray\_img = color\_to\_gray(img)  
  
 cv2.imwrite("bw\_img.png", bw\_img)  
 cv2.imwrite("gray\_img.png", gray\_img)  
  
 if IS\_SHOW\_PRODUCED\_IMG:  
 cv2.imshow("BW\_image (blue\_component)", bw\_img)  
 cv2.imshow("gray\_image", gray\_img)  
 cv2.waitKey(0)  
  
 # Расчитать ряды данных вертикальных и горизонатльных проекций  
  
 hp\_res\_for\_gray = horizontal\_proection(gray\_img)  
  
 vp\_res\_for\_gray = vertical\_proection(gray\_img)  
  
 if IS\_SHOW\_DISTINCT\_GRAPHICS\_FOR\_PROJECTIONS:  
 plt.plot(hp\_res\_for\_gray[:, 0], linestyle='--', marker='.')  
 plt.title("HP")  
 nums = local\_minimum\_numlist(hp\_res\_for\_gray[:, 0])  
 vals = [hp\_res\_for\_gray[i, 0] for i in nums]  
 plt.plot(nums, vals, linestyle='', marker='.')  
 plt.show()  
  
 plt.plot(vp\_res\_for\_gray[:, 0], linestyle='--', marker='.')  
 nums = local\_minimum\_numlist(vp\_res\_for\_gray[:, 0])  
 vals = [vp\_res\_for\_gray[i, 0] for i in nums]  
 plt.plot(nums, vals, linestyle='', marker='.')  
 plt.title("VP")  
 plt.show()  
  
 # Знаходження локальних мінімумов горизонтальної та вертикальної проекцій  
  
 # use local\_minimum\_numlist  
  
 # Отрисовка графиков проеций и изображения  
 fig = MyFig(2, 2)  
  
 unite\_plot\_new(fig, gray\_img, hp\_res\_for\_gray[:, 0], vp\_res\_for\_gray[:, 0])  
 add\_minimums\_to\_ax(fig, hp\_res\_for\_gray[:, 0], 0, 1, True)  
 add\_minimums\_to\_ax(fig, vp\_res\_for\_gray[:, 0], 1, 0)  
  
 fig.show()  
  
 # Обчислення характериситик  
  
 statistic\_for\_hp = compute\_statistics(hp\_res\_for\_gray[:, 0])  
 statistic\_for\_vp = compute\_statistics(vp\_res\_for\_gray[:, 0])  
  
 print("Infromation about statistic for horizontal proection:")  
 for i in statistic\_for\_hp:  
 print(f"{i}: {statistic\_for\_hp.get(i)}")  
  
 print()  
 print("Infromation about statistic for vertical proection:")  
 for i in statistic\_for\_vp:  
 print(f"{i}: {statistic\_for\_vp.get(i)}")  
  
  
class Tests:  
 @staticmethod  
 def is\_eq(m1, m2):  
 for i in range(len(m1)):  
 if m1[i] != m2[i]:  
 return False  
  
 return True  
  
 @staticmethod  
 def test\_hp():  
 matrix = np.array([  
 [159, 223, 248, 15],  
 [224, 118, 178, 192],  
 [246, 8, 94, 66],  
 [166, 66, 147, 209],  
 [40, 94, 79, 137]  
 ])  
 res = np.array([645, 712, 414, 588, 350])  
 t1 = horizontal\_proection(matrix)  
 assert Tests.is\_eq(t1, res) == True  
 print("hp Test passed")  
  
 @staticmethod  
 def test\_img():  
 img = open\_img("cofee.png")  
 print(f"Length: {img.shape[0]}; \n Width: {img.shape[1]}")  
  
 @staticmethod  
 def test\_ndarray():  
 pass  
  
 @staticmethod  
 def test\_localminimum():  
 testdata = np.array([15, 4, 3, 12, 15, 18, 4, 13])  
 resdata = [3, 4]  
 assert resdata == [testdata[i] for i in local\_minimum\_numlist(testdata)]  
  
  
IS\_SHOW\_ORIGINAL\_IMG = False  
IS\_SHOW\_PRODUCED\_IMG = False  
IS\_SHOW\_DISTINCT\_GRAPHICS\_FOR\_PROJECTIONS = False  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 client(MODE.TEST, "cofee.png")

Висновок

У ході виконання лабораторної роботи навчився працювати з бібліотеками opencv, numpy, scypy, matplotlib для мови програмування Python. Також навчився перетворювати повноколірне зображення у зображення у сірих тонах, та у чорно-біле зображення. Реалізував функції для знаходження горизонтальної та вертикальної проекцій та їх локальних мінімумов, а також застосував свою програму до декількох картинок та перевірив її працездатністью. Та реалізував функції для знаходження статичних показників проекцій.