ЛАБОРАТОРНА РОБОТА № 8
ДОСЛІДЖЕННЯ МЕТОДІВ КОМП'ЮТЕРНОГО ЗОРУ

***Мета*:** використовуючи спеціалізовані бібліотеки та мову програмування Python навчитися обробляти зображення за допомогою бібліотеки OpenCV.

Хід роботи:

Завдання 1

Лістинг програми:

```
from google.colab import drive
drive.mount('/content/drive')

!pip install pyngrok==5.0.5
from pyngrok import ngrok

import cv2
from IPython.display import display, Javascript, HTML
from google.colab.output import eval_js
from base64 import b64decode
from google.colab.patches import cv2_imshow

# Функція для захоплення відео з вебкамери та відображення його в Colab
def webcam():
    js = Javascript('''
        async function create() {
            div = document.createElement('div');
            document.body.appendChild(div);

            video = document.createElement('video');
            video.setAttribute('playsinline', '');

            div.appendChild(video);

            stream = await navigator.mediaDevices.getUserMedia({video:
{facingMode: "environment"}});
            video.srcObject = stream;
```

					ДУ «Житомирська політехніка».23.122.4.000 – Лр8		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Дяченко В.В.			Звіт з лабораторної роботи	Літ.	Арк. 1 / Аркушів
Перевір.		.					
Керівник						ФІКТ Гр. КН-20-1(1)	
Н. контр.							
Зав. каф.							

```
                await video.play();


                // Змінити розмір виводу для відповідності елементу відео.
google.colab.output.setIframeHeight(document.documentElement.scrollHeight,
true);

                return;
            }

            async function capture() {
                const canvas = document.createElement('canvas');
                canvas.width = video.videoWidth;
                canvas.height = video.videoHeight;
                canvas.getContext('2d').drawImage(video, 0, 0);
                img = canvas.toDataURL('image/jpeg', 0.8);
                return img;
            }
        ''')

        display(js)
        eval_js('create()')

        capturing = True

        # Захоплення зображення при натисканні 'q' або коли користувач
вводить 'q' в текстовому полі
        display(HTML("Введіть 'q' у текстовому полі нижче, щоб припинити
захоплення."))
        text_input = ""
        while capturing:
            img_data = eval_js('capture()')
            img_binary = b64decode(img_data.split(',')[1])

            # Зберегти зображення
            with open("DIACHENKO.jpg", "wb") as f:
                f.write(img_binary)

            # Перевірити, чи користувач ввів 'q' в текстовому полі
            if 'text_input' in locals() and text_input.lower() == 'q':
                capturing = False
            else:
                # Відображення текстового поля для введення користувачем
                text_input = input("Введіть 'q', щоб зупинити захоплення: ")

        # Звільнення вебкамери
        eval_js('stream.getTracks().forEach(track => track.stop());')
```

```
# OpenCV код для відображення захопленого зображення
webcam()

# Завантаження та відображення захопленого зображення
img = cv2.imread("DIACHENKO.jpg")
cv2_imshow(img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

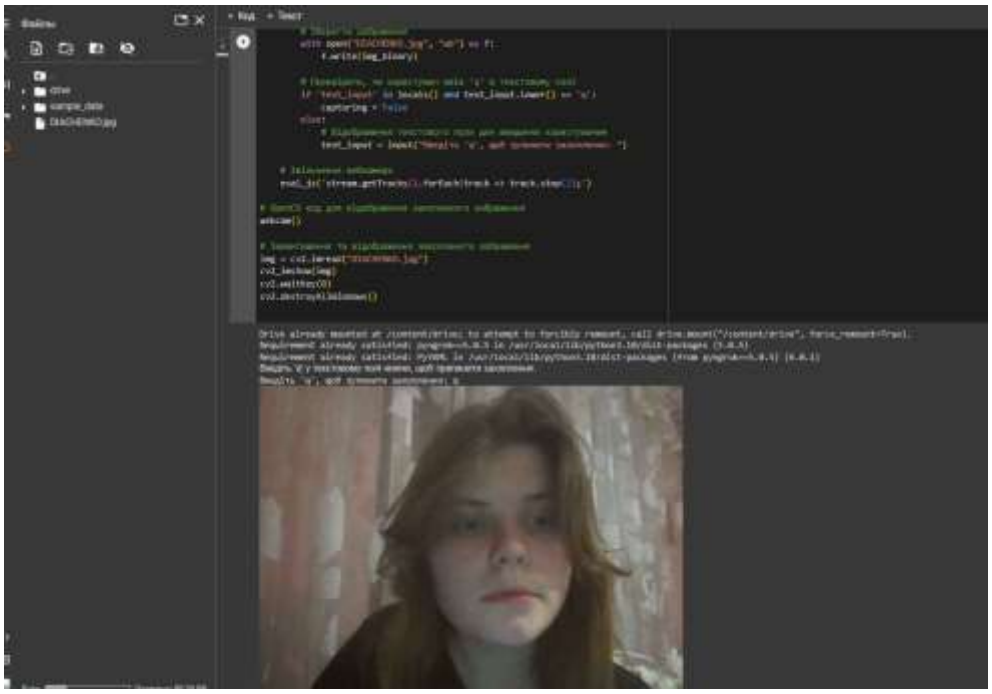

Рис.1. Результат роботи програми та отримане зображення.

Висновки: Початковий код, який використовує бібліотеку OpenCV (cv2.VideoCapture), може не працювати в середовищі Google Colab через обмеження доступу до фізичних пристроїв.

Модифікований код в свою чергу використовує для взаємодії navigator.mediaDevices.getUserMedia - стандартну JavaScript-функцію, яка використовується для отримання доступу до мультимедійних пристроїв. Також даний код він може бути більш гнучким для використання в інтерактивних середовищах, таких як Google Colab.

Для отримання результату ми використали механізм захоплення та збереження зображень. Етапи виконуються в циклі, поки користувач не введе 'q'.

Завдання 2

Лістинг програми:

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow

img = cv2.imread("DIACHENKO.jpg")
kernel = np.ones((5,5),np.uint8)
imgGray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
imgBlur = cv2.GaussianBlur(imgGray,(7,7),0)
imgCanny = cv2.Canny(img,150,200)
imgDialation = cv2.dilate(imgCanny,kernel,iterations=1)
imgEroded = cv2.erode(imgDialation,kernel,iterations=1)

cv2_imshow(imgGray)
cv2_imshow(imgBlur)
cv2_imshow(imgCanny)
cv2_imshow(imgDialation)
cv2_imshow(imgEroded)
```

Рис.2. Отримані imgGray та imgBlur

Рис.3. Отримані imgCanny, imgDialation та imgEroded.

Висновки: виконавше дане завдання ми отримали низку редагованих зображень.

- cvtColor (колірне просторове перетворення): використовується для зміни колірного простору зображення. Результат застосування cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) в коді - це перетворення кольорового зображення в відтінки сірого (grayscale), де кожен піксель представлений одним значенням яскравості (інтенсивності).
- GaussianBlur (гауссівське розмиття): використовується для розмиття зображення, що дозволяє зменшити шум та видалити деталі. Результат застосування cv2.GaussianBlur(imgGray, (7,7), 0) в коді - це отримання розмитого зображення, де значення кожного пікселя обчислюється на основі значень його сусідів згідно з гауссівським розподілом.
- Canny (виявлення країв): використовується для виявлення країв на зображенні.Результат застосування cv2.Canny(img, 150, 200) в коді - це отримання зображення, на якому відзначені контури та краї об'єктів.
- dilate (розширення): використовується для розширення областей яскравих пікселів на зображенні. Результат застосування cv2.dilate(imgCanny, kernel, iterations=1) в коді - це підсилення та розширення виявлених країв на зображенні.
- erode (ерозія): використовується для зменшення областей яскравих пікселів на зображенні. Результат застосування cv2.erode(imgDialation, kernel, iterations=1) в коді - це зменшення ширини виявлених країв та видалення невеликих деталей на зображенні.

Завдання 3

Лістинг програми:

```
import cv2
from google.colab.patches import cv2_imshow
import numpy as np

img = cv2.imread("DIACHENKO.jpg")
print(img.shape)
```

```
imgResize = cv2.resize(img, (1000, 500))
print(imgResize.shape)

imgCropped = img[46:119, 352:495]

# Вивід оригінального зображення
cv2_imshow(img)
print("\n")
# Вивід зміненого розміру зображення
#cv2_imshow(imgResize)
#print("\n")
# Вивід вирізаного зображення
cv2_imshow(imgCropped)
print("\n")
cv2.waitKey(0)
```

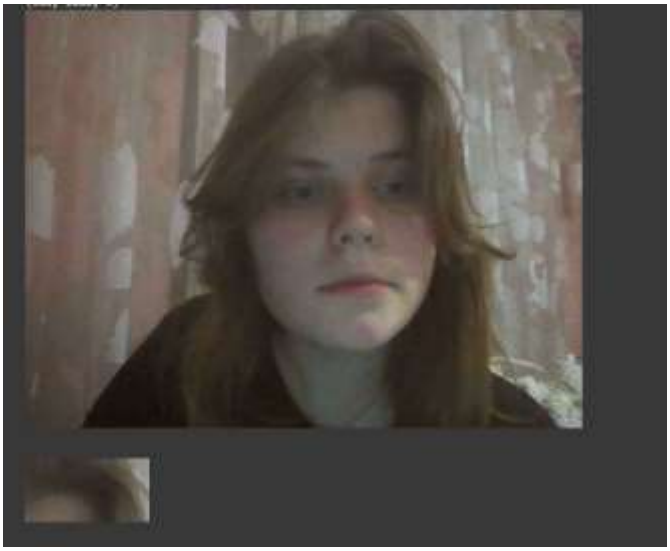

Рис.4. Результат вирізання частини зображення.

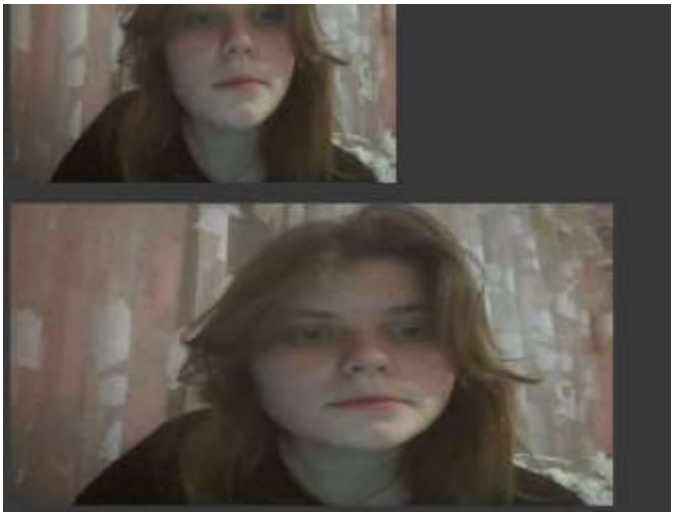

Рис.5. Результат розтягування зображення.

Завдання 4

Лістинг програми:

```
# Імпорт бібліотек та модулів
import cv2
from google.colab.patches import cv2_imshow

# Ініціалізація класифікатора для виявлення обличчя
faceCascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")

# Зчитування зображення з файлу
img = cv2.imread('DIACHENKO.jpg')

# Перетворення зображення у відтінки сірого
imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Виявлення обличчя на зображенні
faces = faceCascade.detectMultiScale(imgGray, 1.1, 4)

# Обробка та виділення областей, де знайдено обличчя
for (x, y, w, h) in faces:
    cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)

# Виведення зображення з виділеними областями обличчя
cv2_imshow(img)

# Очікування натискання клавіші для завершення відображення
cv2.waitKey(0)
```

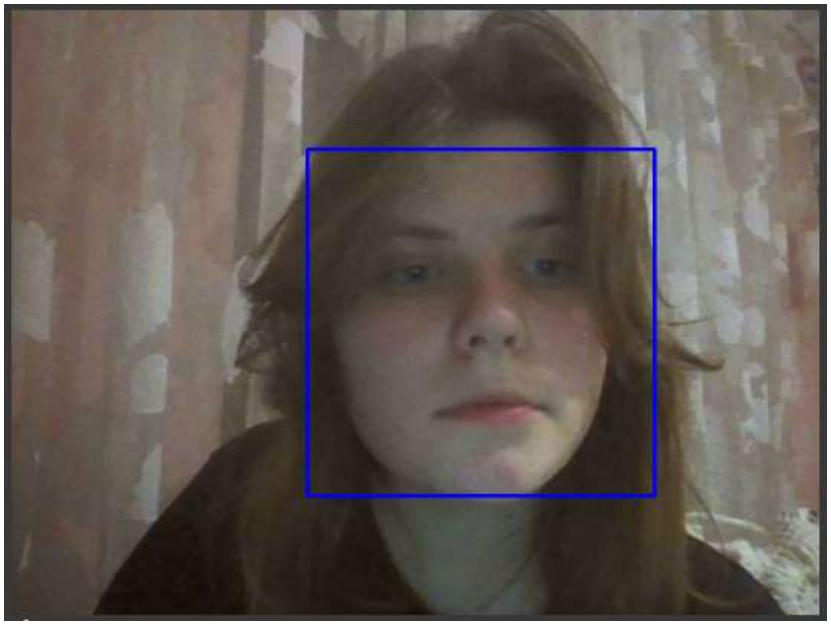

Рис.6. Результат роботи програми.

Висновки: виконавши дане завдання ми виявили обличчя на зображенні за допомогою методу машинного навчання Haar Cascade. Спочатку створюється екземпляр cv2.CascadeClassifier, який ініціалізується з файлом XML, що містить попередньо навчену модель. Зображення (DIACHENKO.jpg) зчитується за допомогою OpenCV, та потім перетворюється у відтінки сірого (imgGray), що допомагає полегшити обробку. Використовується метод detectMultiScale для виявлення зображення. Отримані координати та розміри обличчя зберігаються у змінній faces. Для виділення результату використовується прямокутник (синій на зображенні).

Завдання 5

Лістинг програми:

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

img = cv.imread('messi_full.JPG', 0)
img2 = img.copy()
template = cv.imread('messi_face.JPG', 0)
w, h = template.shape[::-1]

# All the 6 methods for comparison in a list
methods = ['cv.TM_CCOEFF', 'cv.TM_CCOEFF_NORMED', 'cv.TM_CCORR',
           'cv.TM_CCORR_NORMED', 'cv.TM_SQDIFF', 'cv.TM_SQDIFF_NORMED']

for meth in methods:
    img = img2.copy()
    method = eval(meth)

    # Apply template Matching
    res = cv.matchTemplate(img, template, method)
    min_val, max_val, min_loc, max_loc = cv.minMaxLoc(res)

    # If the method is TM_SQDIFF or TM_SQDIFF_NORMED, take minimum
    if method in [cv.TM_SQDIFF, cv.TM_SQDIFF_NORMED]:
        top_left = min_loc
    else:
        top_left = max_loc

    bottom_right = (top_left[0] + w, top_left[1] + h)
    cv.rectangle(img, top_left, bottom_right, 255, 2)
```

```
        plt.subplot(121), plt.imshow(res, cmap='gray')
        plt.title('Matching Result'), plt.xticks([]), plt.yticks([])
        plt.subplot(122), plt.imshow(img, cmap='gray')
        plt.title('Detected Point'), plt.xticks([]), plt.yticks([])
        plt.suptitle(meth)
        plt.show()
```

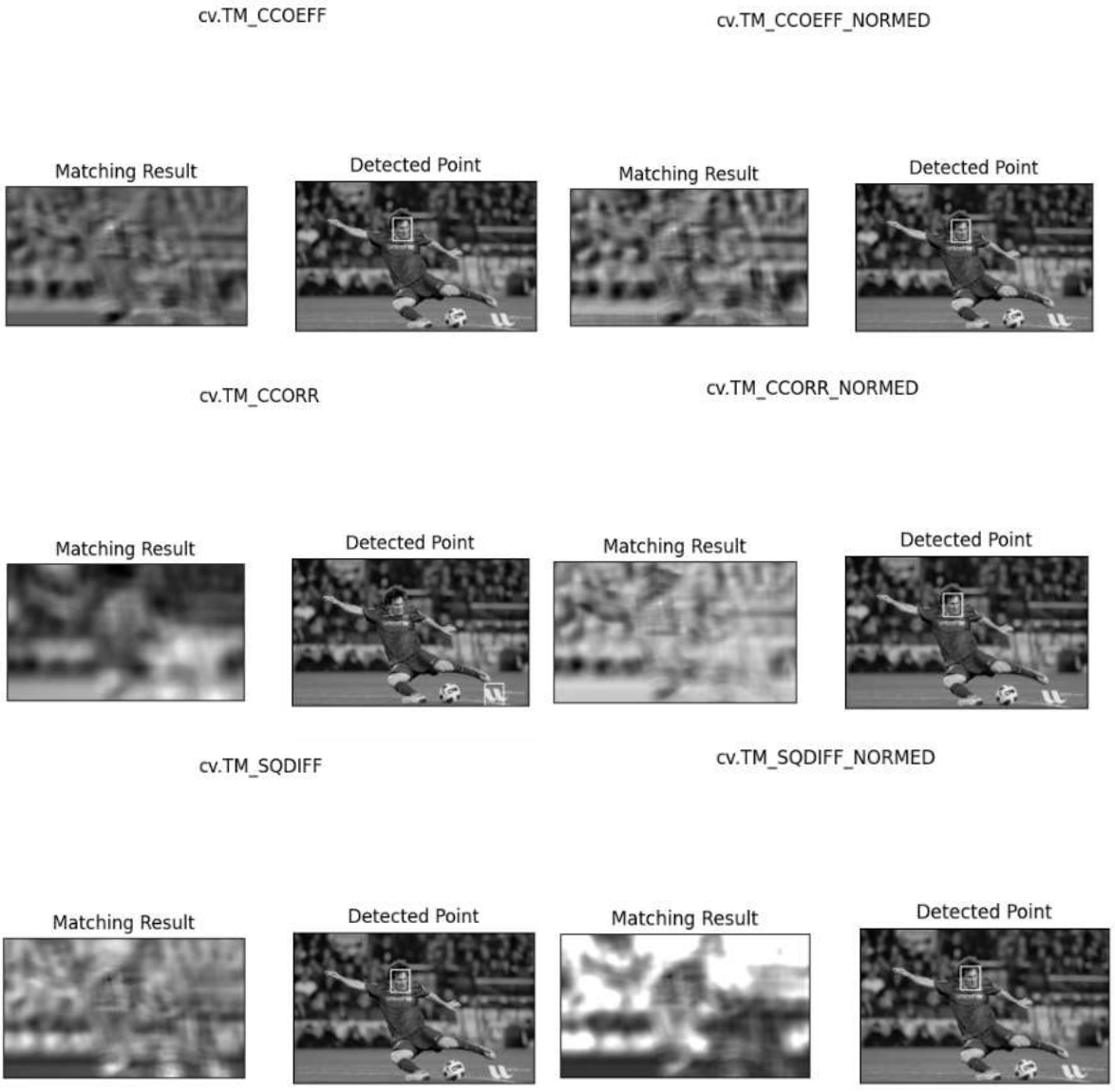

Рис.7. Результат розпізнавання Messi.

Рис.8. Результат розпізнавання нашого зображення.

Висновки: виконавши дане завдання ми здійснили пошук та знаходження розташування зображення шаблону на більшому. Для цього було використано різні методи:

- cv.TM_CCOEFF: використовує кореляцію (коефіцієнт кореляції) як показник подібності між шаблоном і зображенням. Більше значення

вказує на сильніший збіг. Метод не нормалізує результати, тобто значення може бути від'ємним.

- cv.TM_CCOEFF_NORMED: Цей метод також використовує кореляцію, але нормалізований від 0 до 1. Найвище значення вказує на найкращий збіг.
- cv.TM_CCORR: Використовує взаємокореляцію як показник подібності. Більше значення вказує на сильніший збіг.
- cv.TM_CCORR_NORMED: Використовує нормалізовану взаємокореляцію. Найвище значення вказує на найкращий збіг.
- cv.TM_SQDIFF: Використовує суму квадратів різниці як показник подібності. Найменше значення вказує на найкращий збіг.
- cv.TM_SQDIFF_NORMED: Використовує нормалізовану суму квадратів різниці. Найменше значення вказує на найкращий збіг.

У задачі з Мессі ми бачимо точку з найбільшим збігом, хоча і не дуже чітко (через велику к-ть шуму на зображеннях). В другій задачі результат краще. Усі методи дають приблизно однакові візуально, окрім TM_CCORR. В обох задачах даний метод має найгірші результати (в другій задачі взагалі виділена не область з шаблоном).

Завдання 6

Лістинг програми:

```
import numpy as np
import cv2
from matplotlib import pyplot as plt
from google.colab.patches import cv2_imshow

img = cv2.imread('coins.jpg')
cv2_imshow(img)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)
cv2_imshow(thresh)
# видалення шуму
kernel = np.ones((3, 3), np.uint8)
opening = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel, iterations=2)
```

```
    # певна фонова область
    sure_bg = cv2.dilate(opening, kernel, iterations=3)

    # Пошук впевненої області переднього плану
    dist_transform = cv2.distanceTransform(opening, cv2.DIST_L2, 5)
    ret, sure_fg = cv2.threshold(dist_transform, 0.7 * dist_transform.max(),
255, 0)

    # Пошук невідомого регіону
    sure_fg = np.uint8(sure_fg)
    unknown = cv2.subtract(sure_bg, sure_fg)
    cv2_imshow(opening)

    # Маркування міток
    ret, markers = cv2.connectedComponents(sure_fg)

    # Додайте один до всіх міток, щоб впевнений фон був не 0, а 1
    markers = markers + 1

    # Тепер позначте область невідомого нулем
    markers[unknown == 255] = 0
    markers = cv2.watershed(img, markers)
    img[markers == -1] = [255, 0, 0]
    cv2_imshow(img)
```

Рис.9. Початкове зображення.

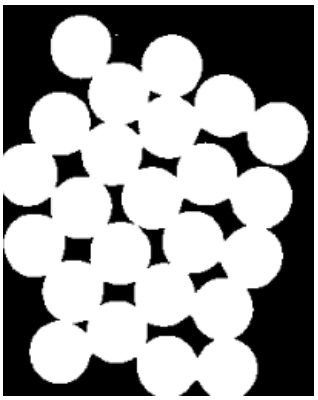

Рис.10. Результат пошуку приблизної оцінки монет.

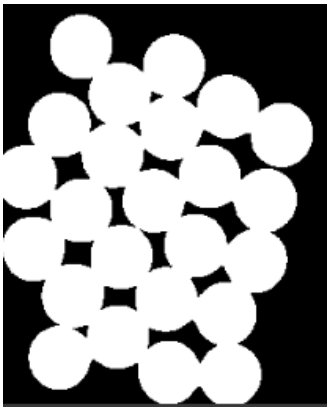

Рис.11. Результат видалення шумів та пошуку областей**.**

Рис.12. Результат**.**

Висновки: виконавши дане завдання ми зробили сегментацію зображення алгоритмом водо-розподілу (Watershed). На попередньо обробленому зображенні виділяються фонові і передніх області, після чого використовується алгоритм водорозподільного перетинання для визначення границь між ними.

Алгоритм може бути не коректним у випадках, коли об'єкти торкаються або перекриваються. Це відбувається через основний принцип його роботи, який полягає в "заповненні" областей водою від локальних максимумів.

Завдання 7

Лістинг програми:

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

# Зчитування зображення
img = cv2.imread('coins.jpg')
```

```
# Додавання контрастності
img = cv2.convertScaleAbs(img, alpha=0.8, beta=20)

# Перетворення в градації сірого
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Бінаризація та видалення шуму
ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)

# Знаходження контурів монет
contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

# Визначення мінімальної площі контуру для фільтрації
min_contour_area = 30

# Фільтрація за розміром монет
filtered_contours = [cnt for cnt in contours if cv2.contourArea(cnt) >
min_contour_area]

# Визначення діаметрів
diameters = []

# Прохід по кожному контуру і визначення діаметра
for contour in filtered_contours:
    # Отримання прямокутника, що обрамлює контур
    x, y, w, h = cv2.boundingRect(contour)

    # Визначення діаметра (може бути адаптовано до реального розміру монет)
    diameter = max(w, h)
    diameters.append(diameter)

# Знаходження найбільш схожих діаметрів
similar_diameters = []
for i in range(len(diameters)):
    for j in range(i + 1, len(diameters)):
        if abs(diameters[i] - diameters[j]) < 10:  # Адаптувати поріг
відповідно до потреб
            similar_diameters.append((i, j))

# Створення чорного зображення для відображення монет
segmented_coins = np.zeros_like(img)

# Фарбування пар монет у спільний колір
for pair in similar_diameters:
    color = (np.random.randint(0, 255), np.random.randint(0, 255),
np.random.randint(0, 255))
```

```
    cv2.drawContours(segmented_coins, [filtered_contours[pair[0]], fil-
tered_contours[pair[1]]], -1, color, -1)

# Відображення результату та оригінального зображення
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title('Original Image')

plt.subplot(1, 2, 2)
plt.imshow(segmented_coins)
plt.title('Segmented Coins')

plt.show()
```

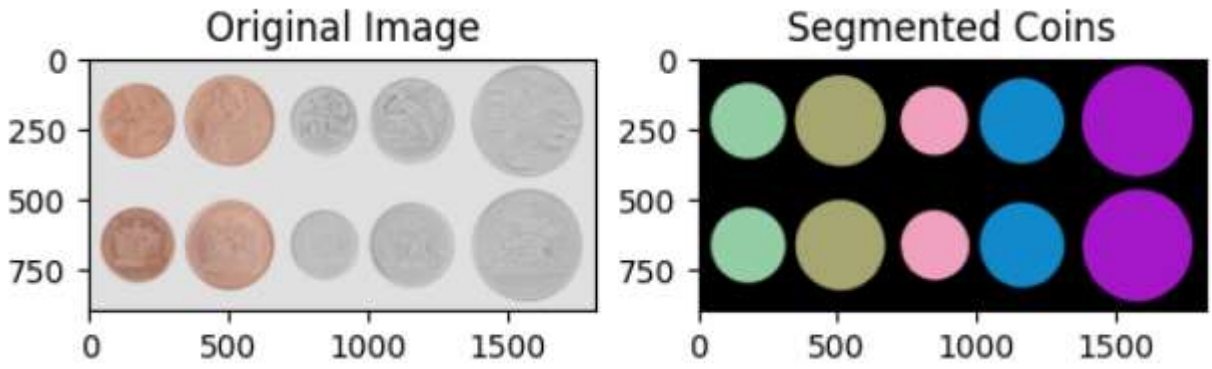

Рис.13. Результат.

Висновки: виконавши дане завдання ми зробили сегментацію зображення монет з використанням бінаризації, фільтрації за розміром та групування монет зі схожими діаметрами.

Спочатку знаходяться зовнішні контури на бінарному зображенні, потім фільтруються за мінімальною площею, визначеною змінною. Після цього обчислюються діаметри для кожного контуру за допомогою обрамлюючих прямокутників. І нарешті знаходяться пари монет зі схожими діаметрами.

На відміну від попереднього завдання тут не було використано операції морфології, водяний алгоритм та маркування для сегментації, тому що діаметри монет мають відносно невелику різницю та можуть бути неточно об’єднані. Код із завдання 6 використовує більш складні методи обробки зображень та може бути ефективним у випадках, коли форма та розмір областей на зображенні значно різняться або складніші.