

## PRELUCRAREA IMAGINILOR

MARIA-ADINA SĂNDOIU

### 1. DESCRIERE COD

Temele 1,2,3: Acest cod este scris în Python și utilizează biblioteca OpenCV (cv2) pentru a lucra cu imagini. Mai precis, el încarcă o imagine JPEG dintr-un fișier, o afișează pe ecran, o salvează în același fișier și apoi eliberează fereastra de afișare. Importă biblioteca cv2, încarcă imaginea JPEG din fișierul "t1pi.jpeg" într-o variabilă numită "img", afișează imaginea utilizând cv2.imshow() și așteaptă până când se apasă o tastă, utilizând cv2.waitKey(0), salvează imaginea în desktop utilizând cv2.imwrite().

În continuare, sunt prezentate trei transformări ale imaginii:

Transformarea imaginii în tonuri de gri prin utilizarea funcției cv2.cvtColor() care convertește imaginea color la un format de tonuri de gri.

Transformarea imaginii la o imagine binară prin utilizarea funcției cv2.threshold() care convertește tonurile de gri ale imaginii la o imagine binară utilizând o valoare de prag de 127.

Transformarea imaginii color la un format HSV prin utilizarea funcției cv2.cvtColor() care convertește imaginea color la formatul HSV.

Se eliberează memoria pentru fereastra de afișare utilizând cv2.destroyAllWindows(). Am calculat histograma în tonuri de gri, dar și histograma color, folosind calcHist(), iar cu plt.show() le-am afisat.

In continuare prima parte este pentru segmentarea unei imagini grayscale folosind praguri multiple, iar cea de-a doua parte este pentru cuantizarea unei imagini grayscale la un număr specific de culori folosind algoritmul de corecție Floyd-Steinberg.

Pentru a aplica pragurile multiple, se parcurge lista pragurilor și pentru fiecare prag, se aplică o transformare specifică. Dacă este ultimul prag, se aplică o transformare binară simplă, în caz contrar se aplică transformarea THRESH\_TOZERO pentru a păstra pixelii cu valoare peste pragul curent.

Imaginea cuantizată este inițializată cu dimensiunea și tipul imaginii grayscale de intrare, adică matricea de pixeli ai imaginii este inițializată cu valoarea 0 de tip uint8. Algoritmul de corecție Floyd-Steinberg este aplicat prin parcurgerea matricei cuantizate de pixeli de la stânga la dreapta și de sus în jos. Temele 4,5: Codul reprezintă o implementare simplă a unui algoritm de etichetare a imaginilor binare și a unui algoritm de cuantizare a imaginilor, urmate de un calcul geometric al perimetrului și ariei unui obiect într-o imagine binară.

În primul segment de cod, imaginea binară este citită utilizând biblioteca OpenCV într-un mod de scală de gri (IMREAD\_GRAYSCALE). Apoi, o matrice goală de etichete cu aceeași dimensiune ca imaginea originală este inițializată cu valori zero

(labels = np.zeros like(imag)). Un label este folosit pentru a identifica diferitele regiuni conectate în imaginea binară. Algoritmul parcurge fiecare pixel al imaginii binare și etichetează regiunile negre (cu valoarea de intensitate 0) care nu au fost încă etichetate. Aceasta folosește o coadă pentru a păstra trackul vecinilor și a extinde regiunea etichetată până la identificarea întregii regiuni. Etichetele sunt reprezentate ca valori între 0 și 255, cu o incrementare la fiecare nouă regiune identificată.

În următorul segment de cod, se aplică un algoritm de cuantizare a imaginilor, numit algoritmul Floyd-Steinberg. Acesta constă în împărțirea imaginii în clase de echivalență în funcție de nivelul de intensitate al fiecărui pixel, urmată de înlocuirea valorii fiecărui pixel cu valoarea medie a clasei de echivalență respective. Algoritmul se aplică pe o imagine deja binarizată și are ca rezultat o imagine cu un număr redus de nivele de intensitate, ceea ce poate reduce cantitatea de informație necesară pentru a prezenta imaginea și poate duce la o compresie a imaginii. În acest cod, sunt calculate clasele de echivalență, se aplică algoritmul pe imaginea cuantizată și se obține o imagine rezultat cu valorile de intensitate actualizate.

În final, în ultimul segment de cod, se calculează perimetru și aria unui obiect în imaginea binară utilizând funcțiile din biblioteca OpenCV. Aceste valori sunt afișate la ieșire, fără a fi salvate într-o imagine sau a fi afișate grafic. Temele 6,7,8: Se citește o imagine binară și se caută contururile obiectelor din imagine folosind funcția cv2.findContours(). Se desenează contururile găsite pe o copie a imaginii originale folosind funcția cv2.drawContours(). Se afișează imaginea cu contururile. Algoritmul de extragere a codurilor înlățuite (Freeman) pentru un obiect dintr-o imagine binară: Se definește o funcție freeman chaincode() care primește ca argument un contur și returnează codurile înlățuite conform algoritmului Freeman. Se citesc o imagine și se convertește la tonuri de gri. Se aplică Threshold pentru a obține o imagine binară. Se găsesc contururile obiectelor din imagine folosind funcția cv2.findContours(). Se identifică cel mai mare contur (obiect) folosind funcția cv2.contourArea() și max(). Se apelează funcția freeman chaincode() pe conturul identificat pentru a obține codurile înlățuite. Se afișează codurile înlățuite obținute. Tema 7: Se citește o imagine și se convertește la tonuri de gri. Se aplică Threshold pentru a obține o imagine binară. Se găsesc contururile obiectelor din imagine folosind funcția cv2.findContours(). Se copiază imaginea inițială pentru a desena conturul pe aceasta. Se desenează conturul găsit pe imaginea copiată folosind funcția cv2.drawContours(). Se afișează imaginea cu conturul. Algoritmul de umplere a regiunilor: Se citește o imagine. Se selectează un punct de start pentru umplere și o culoare pentru umplere. Se creează o matrice de umplere cu valori de 0. Se apelează funcția cv2.floodFill() pentru a umple regiunea în jurul punctului de start cu culoarea specificată. Se afișează imaginea umplută. Tema 8: Se citește o imagine. Se convertește imaginea la tonuri de gri. Se calculează histograma imaginii. Se normalizează histograma. Se calculează probabilitățile pentru valorile de intensitate ale pixelilor. Se calculează media globală a intensității pixelilor. Se initializează pragul cu valoarea mediei globale. Se aplică algoritmul de binarizare globală

Temele 9,10,11,12: Codul prezentat pare a fi un rezumat al unor tehnici de filtrare a imaginilor în domeniul spațial și frecvențial, folosind diferite tipuri de filtre.

În prima parte a codului (TEMA 9), se aplică un filtru "trece jos" pentru netezirea imaginilor și eliminarea zgomotelor. Pentru aceasta, se definește dimensiunea ferestrei de filtrare și se adaugă padding (umplere) pentru a evita pierderea informației la marginea imaginii. Apoi, se definește filtrul medie ca o matrice de valori și se aplică pe imaginea de intrare folosind funcția cv2.filter2D(). Se elimină apoi padding-ul și se afișează imaginea filtrată rezultată.

În continuare (TEMA 9), se aplică un filtru Laplace pentru evidențierea muchiilor în imaginea grayscale de intrare. Acesta se realizează prin aplicarea funcției cv2.Laplacian() pe imaginea grayscale și se afișează imaginea rezultată.

În partea a doua a codului (TEMA 10), se aplică un filtru ideal de tip "trece jos" în domeniul frecvențial, folosind transformata Fourier. Se încarcă o imagine de intrare și se aplică transformata Fourier pe aceasta folosind np.fft.fft2(). Se realizează apoi o deplasare a spectrului Fourier pentru a centra componentele de frecvență folosind np.fft.fftshift(). Se definește un filtru ideal de tip "trece jos" ca o mască binară în jurul centrului spectrului Fourier și se aplică pe acesta. Se inversează deplasarea spectrului Fourier folosind np.fft.ifftshift() și se aplică transformata Fourier inversă folosind np.fft.ifft2(). Se afișează imaginea filtrată rezultată alături de imaginea de intrare.

În continuare (TEMA 10), se aplică un filtru ideal de tip "trece sus" în domeniul frecvențial, inversând rolurile valorilor de tăiere în raport cu un filtru "trece jos". Se definește o nouă mască binară pentru filtrul "trece sus" și se aplică pe spectrul Fourier al imaginii de intrare. Se inversează deplasarea spectrului Fourier și se aplică transformata Fourier inversă. Se afișează imaginea filtrată rezultată alături de imaginea de intrare.

În ultima parte a codului (TEMA 11-12), se aplică un filtru Gaussian pe o imagine de intrare folosind funcția cv2.GaussianBlur(). Se definește un nucleu Gaussian și un sigma (deviere standard) pentru filtru și se măsoară timpul de procesare al filtrului. Imaginea filtrată rezultată este salvată într-un fișier de ieșire. Apoi, se aplică un filtru bidimensional folosind funcția cv2.filter2D() și un kernel definit într-o matrice.

## 2. COD IN PYTHON

```

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

img = cv2.imread('t1pi.jpeg')
cv2.imshow('NORMALA', img)
cv2.waitKey(0)
cv2.imwrite("imaginenua1.jpg", img)

##Tema 2
# Gri
gray_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2.imshow('GRI', gray_image)
cv2.waitKey(0)
cv2.imwrite("imagine_gri.jpg", gray_image)

```

```

# Binara
_, binary_image = cv2.threshold(gray_image, 127, 255, cv2.
    THRESH_BINARY)
cv2.imshow('BINARA', binary_image)
cv2.waitKey(0)

cv2.imwrite("imagine_binara.jpg", binary_image)

# RGB in HSV
hsv_image = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
cv2.imshow('HSV', hsv_image)
cv2.waitKey(0)
cv2.imwrite("imagine_hsv.jpg", hsv_image)
cv2.destroyAllWindows()

# Calculare histograma
hist = cv2.calcHist([gray_image], [0], None, [256], [0, 256])

# Afisare histograma
plt.plot(hist)
plt.title('Histograma tonurilor de gri')
plt.xlabel('Intensitate')
plt.ylabel('Numar de pixeli')
plt.show()

## Calculare histograma color
b, g, r = cv2.split(img)

hist_r = cv2.calcHist([r], [0], None, [256], [0, 256])
hist_g = cv2.calcHist([g], [0], None, [256], [0, 256])
hist_b = cv2.calcHist([b], [0], None, [256], [0, 256])

plt.plot(hist_r, color='r')
plt.plot(hist_g, color='g')
plt.plot(hist_b, color='b')
plt.xlim([0, 256])
plt.title('Histogram for Color Image')
plt.show()

# Determinare praguri multiple
thresholds = [50, 100, 150] # pragurile multiple
max_val = 255 # valoarea maxima pentru pixelii selectati

# Initializare imagine rezultat
result_image = np.zeros(gray_image.shape, dtype=np.uint8)

# Aplicare praguri multiple
for i, thresh in enumerate(thresholds):
    # Daca este ultimul prag
    if i == len(thresholds) - 1:
        _, thresh_image = cv2.threshold(gray_image, thresh, max_val,
            cv2.THRESH_BINARY)

```

```

else:
    - , thresh_image = cv2.threshold(gray_image, thresh, max_val,
        cv2.THRESH_TOZERO)
    - , thresh_image = cv2.threshold(thresh_image, thresholds[i+1],
        max_val, cv2.THRESH_BINARY)
# Adaugare la imaginea rezultat
result_image = cv2.bitwise_or(result_image, thresh_image)

# Afisare praguri si imagine rezultat
print("Praguri obtinute:", thresholds)
cv2.imshow('Imaginea segmentata', result_image)
cv2.imwrite("imaginesegmentata.jpg", result_image)
cv2.waitKey(0)
cv2.destroyAllWindows()

# Cuantizare imagine la 4 culori
num_colors = 4
div = 256 // num_colors
quantized_image = np.uint8(gray_image // div * div + div // 2)

# Initializare imagine rezultat
h, w = quantized_image.shape[:2]
rezultat_image = np.zeros((h, w), dtype=np.uint8)

# Algoritm de corectie Floyd–Steinberg
for y in range(h - 1):
    for x in range(1, w - 1):
        old_pixel = quantized_image[y, x]
        new_pixel = np.round(old_pixel / div) * div
        rezultat_image[y, x] = new_pixel
        error = old_pixel - new_pixel

        quantized_image[y, x+1] += error * 7 // 16
        quantized_image[y+1, x-1] += error * 3 // 16
        quantized_image[y+1, x] += error * 5 // 16
        quantized_image[y+1, x+1] += error * 1 // 16

# Afisare imagine cuantizata si imagine rezultat
cv2.imshow('Imagine_cuantizata_Floyd–Steinberg', quantized_image)
cv2.imshow('Imagine_corectata_Floyd–Steinberg', rezultat_image)
cv2.imwrite("imagineFloyd.jpg", rezultat_image)

cv2.waitKey(0)
cv2.destroyAllWindows()
##TEMA 4

imag = cv2.imread('imagine_binara.jpg', cv2.IMREAD_GRAYSCALE)

labels = np.zeros_like(imag)

# Initializare eticheta
label = 1

```

```

# Definire vecini
neighbors = [(0, 1), (0, -1), (1, 0), (-1, 0)]

# Traversare imagine
for i in range(imag.shape[0]):
    for j in range(imag.shape[1]):
        # Verificare daca pixelul este negru si nu a fost inca
        # etichetat
        if imag[i, j] == 0 and labels[i, j] == 0:
            # Adaugare eticheta noua
            labels[i, j] = np.clip(label, 0, 255)

        # Initializare coada
        q = [(i, j)]
        # Parcursere vecini
        while len(q) > 0:
            x, y = q.pop(0)
            # Parcursere vecini
            for dx, dy in neighbors:
                nx, ny = x + dx, y + dy
                # Verificare daca vecinul este negru si nu a fost
                # inca etichetat
                if nx >= 0 and nx < imag.shape[0] and ny >= 0 and
                   ny < imag.shape[1] and imag[nx, ny] == 0 and
                   labels[nx, ny] == 0:
                    # Adsugare eticheta
                    labels[nx, ny] = np.clip(label, 0, 255)
                    # Adaugare vecin in coada
                    q.append((nx, ny))
            # Incrementare eticheta
            label += 1

        # Afisare imagine etichetata
cv2.imshow('Labeled_Image', labels*50)
cv2.imwrite("labelImage.jpg", labels*50)
cv2.waitKey(0)
cv2.destroyAllWindows()

# Algoritm de corectie Floyd–Steinberg cu doua trecheri cu clase de
# echivalente
classes = np.zeros(256, dtype=np.uint8)
for i in range(num_colors):
    classes[i * div:(i + 1) * div] = i
for y in range(h):
    for x in range(w):
        old_pixel = quantized_image[y, x]
        new_pixel = classes[old_pixel]
        result_image[y, x] = new_pixel * div
        error = old_pixel - new_pixel * div
        if x < w - 1:
            quantized_image[y, x + 1] += error * 7 // 16
        if y < h - 1:
            if x > 0:

```

```

        quantized_image[y + 1, x - 1] += error * 3 // 16
        quantized_image[y + 1, x] += error * 5 // 16
        if x < w - 1:
            quantized_image[y + 1, x + 1] += error * 1 // 16
    # Reasignare clase dupa prima trecere
    for i in range(num_colors):
        classes[i * div:(i + 1) * div] = i + (result_image[y, 0] // div + 1) % 2 * num_colors

    # Afisare imagine cuantizata si imagine rezultat
    cv2.imshow('Imagine_Algoritm_doua_trecheri_cu_clase_de_echivalente-' +
               'corectie', quantized_image)
    cv2.imshow('Imagine_Algoritm_doua_trecheri_cu_clase_de_echivalente-' +
               'corectata', result_image)
    cv2.imwrite("Algoritmdouatrecericuclase.jpg", result_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

#TEMA 5 – Calcul geometric

imag = cv2.imread('imagine_binara.jpg', cv2.IMREAD_GRAYSCALE)
# conversia imaginii la o imagine binara
thresh, binarized = cv2.threshold(imag, 127, 255, cv2.THRESH_BINARY)

# calcularea conturului
contours, hierarchy = cv2.findContours(binarized, cv2.RETR_TREE, cv2.
                                         CHAIN_APPROX_SIMPLE)

# calcularea perimetrelui si a aria
perimeter = cv2.arcLength(contours[0], True)
area = cv2.contourArea(contours[0])

# afisarea perimetrelui si a ariei
print("Perimetru obiectului:", perimeter)
print("Aria obiectului:", area)

#Tema 6
##Algoritmul de urmarire a conturului obiectelor dintr-o imagine
binara (etichetata in prealabil)

# Citirea imaginii
img = cv2.imread('imagine_binara.jpg', cv2.IMREAD_GRAYSCALE)

# Cautarea contururilor in imagine
contours, hierarchy = cv2.findContours(img, cv2.RETR_EXTERNAL, cv2.
                                         CHAIN_APPROX_SIMPLE)

# Desenarea contururilor pe o copie a imaginii originale
img_contours = img.copy()
cv2.drawContours(img_contours, contours, -1, (0, 255, 0), 2)

```

```

# Afisarea imaginii cu contururi
cv2.imshow('Contururi', img_contours)
cv2.waitKey(0)
cv2.destroyAllWindows()

#Algoritmul de extragere a codurilor inlantuite pentru un obiect dintr
#-o imagine binara
#Freeman
img = cv2.imread('t1pi.jpeg')
def freeman_chaincode(contour):
    code = []
    direction = 0
    x, y = contour[0]

    for i in range(len(contour) - 1):
        dx = contour[i + 1][0] - x
        dy = contour[i + 1][1] - y

        if dx > 0:
            if dy > 0:
                direction = 0
            elif dy == 0:
                direction = 7
            else:
                direction = 6
        elif dx == 0:
            if dy > 0:
                direction = 1
            else:
                direction = 5
        else:
            if dy > 0:
                direction = 2
            elif dy == 0:
                direction = 3
            else:
                direction = 4

        code.append(direction)
        x, y = contour[i + 1]

    return code

img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
_, thresh = cv2.threshold(img_gray, 127, 255, cv2.THRESH_BINARY)
contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.
    CHAIN_APPROX_NONE)

contours, hierarchy = cv2.findContours(binary_image, cv2.RETR_EXTERNAL
    , cv2.CHAIN_APPROX_NONE)

largest_contour = max(contours, key=cv2.contourArea)
chain_code = freeman_chaincode(largest_contour.reshape(-1, 2))
print('Alg_de_extragere_a_codurilor:', chain_code)

```

```

##Tema 7
#algoritmul de extragere a conturului

# Citire imagine
img = cv2.imread('t1pi.jpeg')

# Convertire imagine in gri
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Aplicare Threshold pentru a obtine imaginea binara
-, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY + cv2.
    THRESH_OTSU)

# Extrage conturul
contours, hierarchy = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.
    CHAIN_APPROX_NONE)

# Copie imaginea initiala pentru a desena conturul
img_contour = img.copy()

#Desenare contur
cv2.drawContours(img_contour, contours, -1, (0, 0, 255), 2)

#Afisare imagine cu conturul
cv2.imshow('Contur', img_contour)
cv2.waitKey(0)
cv2.destroyAllWindows()

#algoritmul de umplere a regiunilor

#citire imagine
img = cv2.imread('t1pi.jpeg')

#selectare punct de start pentru umplere (in acest caz coltul din
#stanga sus al imaginii)
start_point = (0, 0)

#culoarea cu care se va umple regiunea (in acest caz rosu)
new_color = (0, 0, 255)

#crearea unei matrici de umplere cu valori de 0
mask = np.zeros((img.shape[0] + 2, img.shape[1] + 2), dtype=np.uint8)

#umplerea regiunii cu culoarea specificata
cv2.floodFill(img, mask, start_point, new_color)

#afisare imagine
cv2.imshow('Imaginea_umatrata', img)
cv2.waitKey(0)
cv2.destroyAllWindows()

```

```

#tema 8
##Algoritmul de binarizare automata globala

#citirea imaginii
img = cv2.imread('t1pi.jpeg')
#conversia la tonuri de gri
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# calcularea histogramelor
hist = cv2.calcHist([gray], [0], None, [256], [0, 256])

#normalizarea histogramelor
hist_norm = hist.ravel() / np.prod(gray.shape)

#calcularea probabilitatilor
prob = np.zeros(256)
for i in range(256):
    prob[i] = np.sum(hist_norm[:i+1])

#calcularea mediei globale a intensitatii pixelilor
mean_global = np.mean(gray)

#initializarea pragului cu valoarea medie globala
threshold = mean_global

#repetarea pana cand pragul nu se mai schimba
while True:
    #impartirea valorilor de intensitate in doua grupuri
    group1 = gray[gray <= threshold]
    group2 = gray[gray > threshold]

    #calcularea mediilor valorilor de intensitate ale pixelilor pentru
    # fiecare grup
    mean1 = np.mean(group1)
    mean2 = np.mean(group2)

    #calcularea noului prag
    new_threshold = (mean1 + mean2) / 2

    #verificarea daca pragul s-a schimbat
    if np.abs(new_threshold - threshold) < 0.5:
        break

    #actualizarea pragului
    threshold = new_threshold

#aplicarea pragului la imaginea in tonuri de gri
binary = gray.copy()
binary[binary <= threshold] = 0
binary[binary > threshold] = 255
#afisare imag
cv2.imshow('Binarizare_automata_globala', binary)
cv2.waitKey(0)

```

```

cv2.destroyAllWindows()

#Funcțiile de transformare cu forma analitică , precum: negativarea
    imaginii ,
#modificarea contrastului , corectia gamma, modificarea luminozitatii

#Negativarea imaginii
negative_image = 255 - img
cv2.imshow( 'IMAGINE_NEGATIVA' , negative_image)
cv2.waitKey(0)
cv2.imwrite("imagine_negativa.jpg" , negative_image)

#Modificarea contrastului
alpha = 1.5 # factorul de contrast
beta = 0 # offset-ul
contrast_image = cv2.convertScaleAbs(img, alpha=alpha, beta=beta)
cv2.imshow( 'IMAGINE CU CONTRAST' , contrast_image)
cv2.waitKey(0)
cv2.imwrite("imagine_contrast.jpg" , contrast_image)

#Corectia gamma
gamma = 0.5 # valoarea gamma
gamma_correction_image = np.uint8(np.power((img / 255.0) , gamma) *
    255)
cv2.imshow( 'IMAGINE CU CORECTIE GAMMA' , gamma_correction_image)
cv2.waitKey(0)
cv2.imwrite("imagine_gamma.jpg" , gamma_correction_image)

#Modificarea luminozitatii
brightness = 70 # valoarea de ajustare a luminozitatii
brightness_image = cv2.convertScaleAbs(img, beta=brightness)
cv2.imshow( 'IMAGINE CU LUMINOZITATE ADJUSTATA' , brightness_image)
cv2.waitKey(0)
cv2.imwrite("imagine_luminozitate.jpg" , brightness_image)

##Algoritmul de egalizare a histogramei
#Citim imaginea
img = cv2.imread('t1pi.jpeg',0)

#Egalizam histograma
equ = cv2.equalizeHist(img)

#Afisam rezultatele
plt.subplot(1,2,1)
plt.imshow(img, cmap='gray')
plt.title('Imaginea originala')

plt.subplot(1,2,2)
plt.imshow(equ, cmap='gray')
plt.title('Imaginea cu histograma egala')

plt.show()

```

## #TEMA 9

```

#Filtrarea imaginilor in domeniul spatial
#un filtru "trece jos" (de netezire a imaginilor, de eliminare a
#zgomotelor)

#definearea dimensiunii ferestrei de filtrare
window_size = 5

#adaugarea padding-ului pentru a evita pierderea informatiei la
#marginea imaginii
padding_size = window_size // 2
img = cv2.copyMakeBorder(img, padding_size, padding_size, padding_size,
                       padding_size, cv2.BORDER_REPLICATE)

#definearea filtrului medie
filter = np.ones((window_size, window_size), dtype=np.float32) / (
    window_size * window_size)

#aplicarea filtrului
filtered_image = cv2.filter2D(img, -1, filter)

#eliminarea padding-ului
filtered_image = filtered_image[padding_size:-padding_size,
                                 padding_size:-padding_size]

#afisarea imagine
cv2.imshow('Imaginea_filtrata', filtered_image)
cv2.waitKey(0)
cv2.destroyAllWindows()

##un filtru "trece sus" (de evidențiere a muchiilor).
#aplicarea filtrului Laplace
laplacian = cv2.Laplacian(gray, cv2.CV_64F)

#afisarea imaginii rezultate
cv2.imshow('Filtru_Laplace', laplacian)
cv2.waitKey(0)
cv2.destroyAllWindows()

#TEMA 10
#Filtrarea imaginilor in domeniul frecvential
#un filtru ideal de tip "trece jos"

img = cv2.imread('t1pi.jpeg', 0)
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)
rows, cols = img.shape
crow, ccol = int(rows / 2), int(cols / 2)
mask = np.zeros((rows, cols), np.uint8)
mask[crow - 30:crow + 30, ccol - 30:ccol + 30] = 1
fshift = fshift * mask
f_ishift = np.fft.ifftshift(fshift)

```

```

img_back = np.fft.ifft2(f_ishift)
img_back = np.abs(img_back)
plt.subplot(121), plt.imshow(img, cmap='gray')
plt.title('Imaginea_de_intrare'), plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(img_back, cmap='gray')
plt.title('Imaginea_dupa_filtrare_ideală_in_jos'), plt.xticks([]), plt
    .ytics([])
plt.show()

#un filtru ideal de tip "trece sus".

#Transformata Fourier a imaginii
dft = cv2.dft(np.float32(img), flags=cv2.DFT_COMPLEX_OUTPUT)
dft_shift = np.fft.fftshift(dft)

#Dimensiunea imaginii
rows, cols = img.shape
crow, ccol = rows // 2, cols // 2

#Creatia filtrului
#Pentru un filtru ideal de tip "trece sus", inversam rolurile
    valorilor de taiere
# in raport cu un filtru ideal de tip "trece jos"
#Filtul are valori de taiere:50 in cazul nostru (pentru aceasta
    dimensiune de imagine)
##Valorile taierei depind de dimensiunea imaginii
mask = np.ones((rows, cols, 2), np.uint8)
mask[crow - 50:crow + 50, ccol - 50:ccol + 50] = 0

#Aplicarea filtrului pe spectrul imaginii
fshift = dft_shift * mask

#Transformarea Fourier inversa
f_ishift = np.fft.ifftshift(fshift)
img_back = cv2.idft(f_ishift)
img_back = cv2.magnitude(img_back[:, :, 0], img_back[:, :, 1])

#Afisarea imaginilor
plt.subplot(121), plt.imshow(img, cmap='gray')
plt.title('Imaginea_initială'), plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(img_back, cmap='gray')
plt.title('Imaginea_filtrată_in_sus'), plt.xticks([]), plt.yticks([])
plt.show()

#ITEMA 11
#    nrcarea    imaginii de intrare
input_image = cv2.imread('t1pi.jpeg')

# Definirea nucleului gaussian i aplicarea filtrului gaussian prin
    functia cv2.GaussianBlur()
gaussian_kernel = (5, 5)
gaussian_sigma = 1.5

```

```

start_time = time.time()
output_image_gaussian = cv2.GaussianBlur(input_image, gaussian_kernel,
                                           gaussian_sigma)
end_time = time.time()

# Calcularea și afiarea timpului de procesare pentru filtrul
# gaussian
processing_time_gaussian = end_time - start_time
print("Timpul de procesare pentru filtrul gaussian: ", processing_time_gaussian, " secunde")

# Salvarea imaginii de ieșire filtrate prin convoluție cu nucleul
# gaussian
cv2.imwrite('output_image_gaussian.jpg', output_image_gaussian)

# Definirea nucleului bidimensional și aplicarea filtrului
# bidimensional prin funcția cv2.filter2D()
kernel = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
start_time = time.time()
output_image_kernel = cv2.filter2D(input_image, -1, kernel, borderType
                                    =cv2.BORDER.CONSTANT)
end_time = time.time()

# Calcularea și afiarea timpului de procesare pentru filtrul
# bidimensional
processing_time_kernel = end_time - start_time
print("Timpul de procesare pentru filtrul bidimensional: ", processing_time_kernel, " secunde")

# Salvarea imaginii de ieșire filtrate prin convoluție cu nucleul
# bidimensional
cv2.imwrite('output_image_kernel.jpg', output_image_kernel)
cv2.imshow('output_image_kernel', output_image_kernel)
cv2.waitKey(0)
cv2.imshow('output_image_gaussian', output_image_gaussian)
cv2.waitKey(0)

# Tema 12

# Încarcarea imaginii
img = cv2.imread('t1pi.jpeg', cv2.IMREAD.GRAYSCALE)

# Aplicarea filtrului Gaussian pentru reducerea zgomotului
img = cv2.GaussianBlur(img, (3, 3), 0)

# Aplicarea operatorului Laplacian pentru detectarea muchiilor
laplacian = cv2.Laplacian(img, cv2.CV_64F)

# Determinarea valorii de prag adaptiv pentru fiecare pixel
thresh = cv2.adaptiveThreshold(laplacian.astype('uint8'), 255, cv2.
                               ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)

# Salvarea imaginii binarizate

```

```
cv2.imwrite( 'output_image_binary.jpg' , thresh)
cv2.imshow( 'output_image_binary' , thresh)
cv2.waitKey(0)
```

### 3. CERINTA SI REZULTAT

#### 3.1. Tema 1(figurile: 1,2)

citirea unei imagini din fișier  
 afișarea unei imagini  
 salvarea unei imagini pe disc

.

#### 3.2. Tema 2(figurile: 3,4,5)

Conversia unei imagini color în imagine în tonuri de gri  
 Conversia unei imagini în tonuri de gri în imagine binară  
 conversia unei imagini din spațiul de culoare RGB în spațiul de culoare HSV

.

#### 3.3. Tema 3(figurile: 6,7,8)

histograma unei imagini în tonuri de gri  
 implementarea algoritmului de determinare a pragurilor multiple (se vor afișa valorile pragurilor (maximelor) obținute, precum și imaginea obținută după aplicarea algoritmului)  
 implementarea algoritmului de corecție Floyd-Steinberg (se va afișa imaginea obținută după aplicarea algoritmului asupra imaginii cuantizate, obținute la pasul anterior)  
 -;histograma unei imagini color.

#### 3.4. Tema 4(figurile: 11,12)

Algoritm 1 - Traversare în lățime  
 Algoritm 2 - Două treceri cu clase de echivalențe

.

#### 3.5. Tema 5(figurile: 13)

Se vor calcula și afișa câteva dintre trăsăturile geometrice ale unor obiecte binare dintr-o imagine digitală.

#### 3.6. Tema 6(figurile: 14, 13)

Algoritmul de urmărire a conturului obiectelor dintr-o imagine binară (etichetată în prealabil) Algoritmul de extragere a codurilor înlăntuite pentru un obiect dintr-o imagine binară.

#### 3.7. Tema 7(figurile: 15, 16)

algoritmul de extragere a conturului algoritmul de umplere a regiunilor.

#### 3.8. Tema 8(figurile: 17, 18, 19, 20, 21 , 22)

Algoritmul de binarizare automată globală Funcțiile de transformare cu formă analitică, precum: negativarea imaginii, modificarea contrastului, corecția gamma, modificarea luminozității Algoritmul de egalizare a histogramei.

**3.9. Tema 9(figurile: 23, 24)**

un filtru ”trece jos” (de netezire a imaginilor, de eliminare a zgomotelor), un filtru ”trece sus” (de evidențiere a muchiilor).

**3.10. Tema 10(figurile: 25, 26)**

un filtru ideal de tip ”trece jos”, un filtru ideal de tip ”trece sus”.

**3.11. Tema 11(figurile: 27, 28, 13)**

Implementarea a doi algoritmi de filtrare/restaurare a unei imagini prin convoluția acesteia cu un nucleu gaussian, respectiv cu un nucleu bidimensional. Se va calcula și se va afișa timpul de procesare.



FIGURA 1. Imagine originală



FIGURA 2. Rezultat

**3.12. Tema 12(figurile: 29)**

Implementarea algoritmilor de binarizare adaptivă a punctelor de muchie și de prelungire a muchiilor prin histereză.

BIBLIOGRAFIE

<https://www.geeksforgeeks.org/python-opencv-cv2-imwrite-method/>

<https://realpython.com/python-histograms/>

<https://www.mathworks.com/help/images/ref/multithresh.html>

[https://en.wikipedia.org/wiki/FloydSteinberg\\_dithering](https://en.wikipedia.org/wiki/FloydSteinberg_dithering)



FIGURA 3. Binara



FIGURA 4. Gri

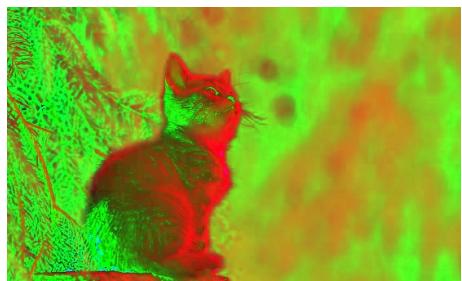


FIGURA 5. HSV

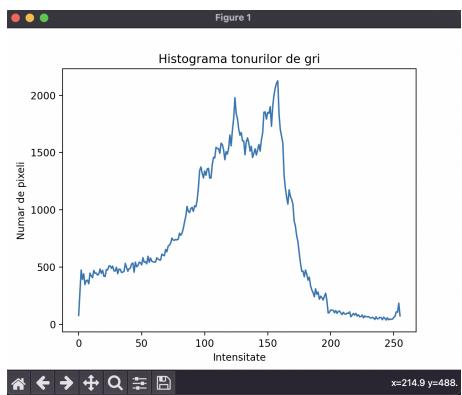


FIGURA 6. Histograma Gri

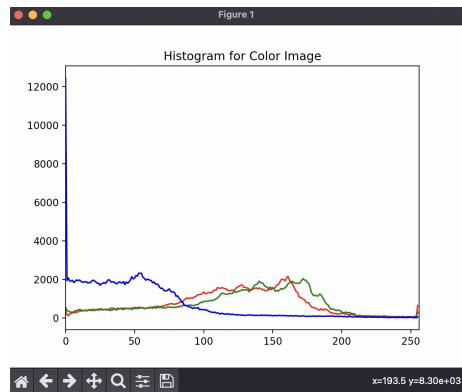


FIGURA 7. Histograma Color



FIGURA 8. Imag Segmentata

```
/usr/local/bin/python3.11 /Users/adinasky/PycharmProjects/Tema1PI/main.py
Praguri obtinute: [50, 100, 150]
Perimetru obiectului: 0.0
Aria obiectului: 0.0
Alg de extragere a codurilor: [2, 3, 2, 3, 1, 2, 3, 2, 2, 3, 2, 2, 3, 2, 3, 3, 2, 3, 3, 0, 1, 2, 6, 7, 6, 6, 7, 0, 1, 2, 0, 1, 2, 4
Process finished with exit code 0
```

FIGURA 9. Valori praguri



FIGURA 10. Floyd-Steinberg

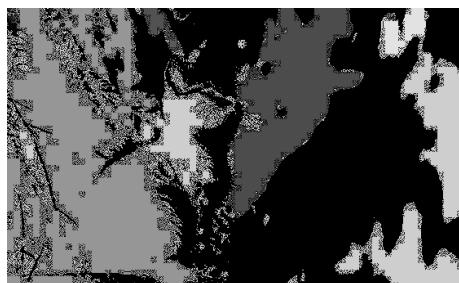


FIGURA 11. Traversare în lățime



FIGURA 12. Două treceri cu clase de echivalență

```
/usr/local/bin/python3.11 /Users/adinasky/PycharmProjects/Tema1PI/main.py
Praguri obtinute: [50, 100, 150]
Perimetru obiectului: 0.0
Aria obiectului: 0.0
Alg de extragere a codurilor: [0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 2, 1, 1, 1, 1, 2, 2, 4, 5, 4
Timpul de procesare pentru filtrul gaussian: 0.0016701221466064453 secunde
Timpul de procesare pentru filtrul bidimensional: 0.0014510154724121094 secunde

Process finished with exit code 0
```

FIGURA 13. Trasaturi geometrice

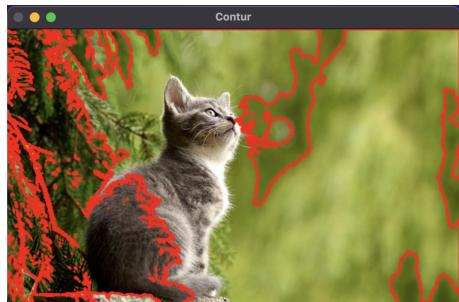


FIGURA 14. Urmarire a conturului



FIGURA 15. Extragere a conturului



FIGURA 16. Umliere a regiunilor



FIGURA 17. Binarizare



FIGURA 18. Negativa



FIGURA 19. Modificare contrast



FIGURA 20. Gamma



FIGURA 21. Modificarea Luminozitate

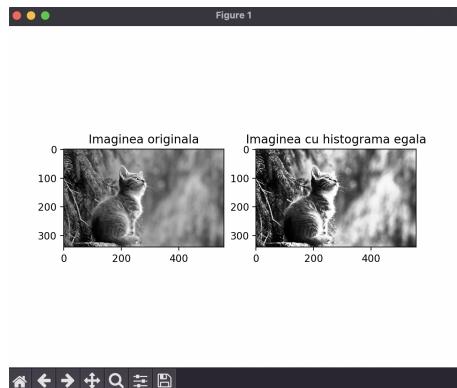


FIGURA 22. Egalare histograma



FIGURA 23. Trece jos(netezire)

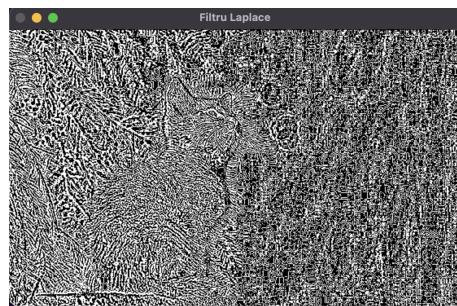


FIGURA 24. Trece sus(evidențiere)

Figure 1

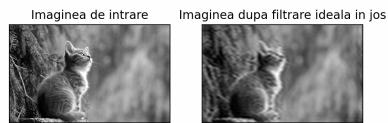


Figure 1

FIGURA 25. Ideal jos

Figure 1

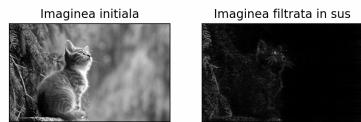


Figure 1

FIGURA 26. Ideal sus



FIGURA 27. Gaussian



FIGURA 28. Nucleu bidimensional

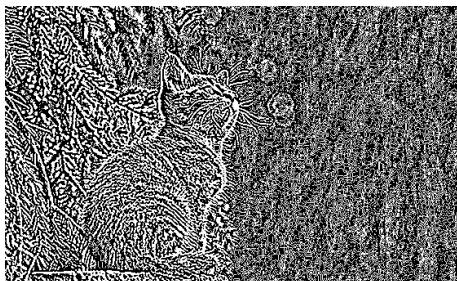


FIGURA 29. Binarizare