Lupu Elena
Grupa 331AB

Image Sharpening

- documentație –

1. Tema proiectului:

Procedeul de **Image Sharpening** este des folosit în prelucrarea de imagini și constă în intensificarea culorilor astfel încât acestea să pară mai vii.

Principala metodă folosită este cea a măștii de convoluție, care înseamnă trecerea unei matrici 3x3 predefinite peste matricea de pixeli a imaginii, urmând ca în cadrul acestei matrici 3x3 să se înmulțească valorile pixelilor cu diverse constante și apoi să se adune pentru a forma un nou pixel.

2. Descrierea metodei folosite:

Având în vedere că imaginea originală este reprezentată pe spațiul RGB 24 de biți și că imaginea prelucrată trebuie să fie de același format, a fost nevoie să adaptez algoritmul corespunzător.

Pasul 1:

Imaginea originală este trecută din spațiul RGB în spațiul de culori YUV.

Spațiul YUV este folosit în algoritmii de prelucrare de imaginii, deoarece transformarea este foarte simplă, liniară, iar imaginea rezultată este ușor de prelucrat. Cele 3 canale semnfică: luminozitatea (Y), nivelul de albastru (U) și nivelul de roșu (V). Formulele pentru transformare pe care le-am folosit sunt:

$$Y = 0,257 * R + 0,504 * G + 0,098 * B + 16$$

$$U = -0,148 * R – 0,291 * G – 0,071 * B + 128$$

$$V = 0,439 * R – 0,368 * G – 0,071 * B + 128$$

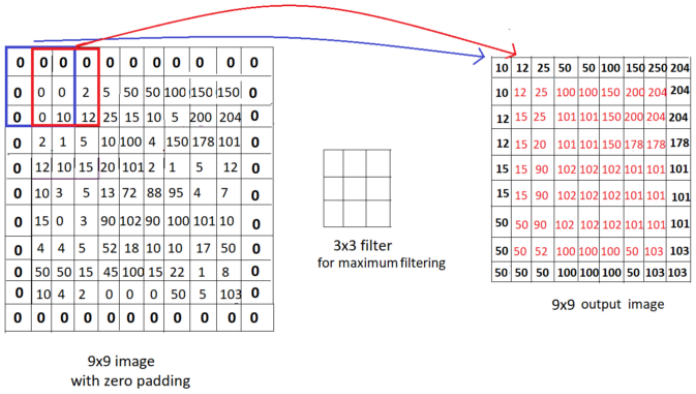
$$R = 1,164 * (Y – 16) + 1,596 * (V – 128)$$

$$G = 1,164 * (Y – 16) – 0,395 * (U – 128) – 0,813 * (V – 128)$$

$$B = 1,164 * (Y – 16) + 2,018 * (U – 128)$$

Pasul 2:

Urmează aplicarea măștii de convoluție. Pentru a păstra dimensiunile imaginii, imaginea este extinsă cu un cadru de pixeli de valoare 0:



Trebuie menționat faptul că la prelucrare se modifică doar canalul Y, deoarece se dorește doar modificarea luminozității. Modificarea canalelor U sau V ar duce la modificarea culorilor și, implicit, la alterarea imaginii într-un mod nedorit.

Masca pentru transformare are 3x3 valori constante:

0	-1	0
-1	5	-1
0	-1	0

Pasul 3:

După ce se formează noua imagine cu valorile pixelilor modificate, se convertește imaginea înapoi la spațiul RGB pentru a putea fi scrisă sub formă de fișier BMP.

3. Descrierea workflow-ului:

Din funcția *main()* se preiau datele pentru identificarea imaginii originale și pentru calea unde se va scrie imaginea prelucrată. De asemenea, sunt pornite cele 3 thread-uri care se vor ocupa de întregul proces, *Producer, Consumer* și *WriterResult.*

Thread-ul *Producer* primește calea de unde trebuie să citească fișierul BMP original. Acesta citește fișierul și-l salvează într-un obiect de tipul BufferedImage (clasă predefinită din java.awt.image.BufferedImage). Acest

obiect este segmentat și trimis în 4 părți prin intermediul *Buffer-ului* la thread-ul *Consumer*.

Thread-ul *Consumer* citește pe rând aceste 4 segmente pe care le primește de la *Producer* și le salvează într-un vector de obiecte BufferedImage.

Cele 4 segmente sunt convertite la tipul *RGBImage* și apoi sunt „lipite" printr-o funcție a clasei *RGBImage* pentru a forma imaginea originală.

Obiectul *RGBImage* este apoi convertit la tipul *PaddedYUVImage*. Acesta reprezintă imaginea convertită la spațiul de culori YUV împreună cu conturul de pixeli nuli.

Prin apelarea unei funcții din *PaddedYUVImage* se efectuează prelucrarea efectivă a imaginii, care este salvată ca obiect de tip *YUVImage*.

Acest obiect trebuie convertit apoi la tipul *RGBImage*.

La final, se trimite prin pipe imaginea refăcută în spațiul RGB, fragmentată în 5 segmente, la thread-ul pentru scriere *WriterResult*.

Thread-ul *WriterResult* preia segmentele într-un vector de obiecte *RGBImage*, le „lipește", apoi se folosește o funcție din *RGBImage* pentru a face transformarea la tipul BufferedImage, necesar pentru scrierea în noul fișier BMP.

4. Descrierea surselor:

4.1. ImageSharpening.java :

Se declară obiectul *Buffer,* folosit pentru comunicația *Producer-Consumer*. Se declară elementele necesare pentru construcția pipe-ului care va asigura comunicarea între thread-urile *Consumer* și *WriterResult* (pipeOut, pipeIn, out, in) .

Variabila statică startTime reține momentul de timp t0 față de care se va calcula timpul de citire a datelor de identificare a fișierului original și a destinației. Variabila statică startTime0 este o copie a lui startTime*,* care este folosită la finalul programului pentru a calcula timpul total de execuție.

Cu ajutorul variabilei inputReader de tip Scanner se citesc de la tastatură calea fișierului original, bmpPath*,* și calea destinației unde trebuie depusă imaginea prelucrată, resultPath.

Se instațiază cele 3 thread-uri și se pornesc. Se poate observa că există un delay adăugat pentru thread-ul *WriterResult.* A fost nevoie de acest delay,

deoarece thread-ul pornea prea repede, înainte de a avea informație de prelucrat, generând eroare (de asemenea, există și o sincronizare implementată pe thread-urile *Consumer* și *WriterResult* pentru a preveni această problemă).

4.2. Producer.java :

Această clasă ce implementează interfața Runnable și există în program o singură instanță de acest tip, care reprezeintă un thread. El se ocupă de cititrea și transmiterea la thread-ul *Consumer* a imaginii originale.

Câmpuri:

Thread t = obiect de tip thread;

String bmpPath = calea de unde trebuie citit fișierul BMP;

Private Buffer imageBuffer = obiect de tip *Buffer,* folosit la trasmiterea sincronă a datelor între *Producer* și *Consumer;*

Metode:

-> public Producer(String bmpPath, Buffer imageBuffer) :

String bmpPath: calea fișierului de citit;

Buffer imageBuffer: obiectul *Buffer* în care trebuie pus fragmentul de imagine

Constructorul care creează obiectul *Producer*.

-> public void run():

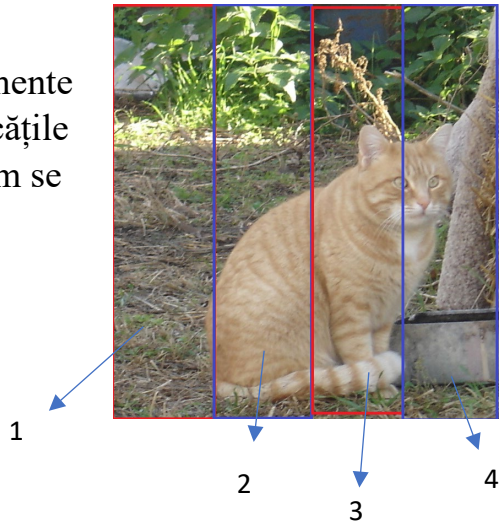
În această funcție se desfășoară activitatea thread-ului.

În variabila *img* se salvează imaginea citită de la sursa dată de către utilizator.

Variabila *startTime* ia valoarea momentului de timp t0 față de care se va calcula timpul de citire al imaginii din sursă.

Variabilele *imgWidth* și *imgHeight* păstrează valorile lățimii, respectiv, înălțimii imaginii citite.

Imaginea este transmisă împărțită în 4 fragmente secționate vertical, astfel încât să aibă toate bucățile aceeași înălțime (doar lăimea să difere), așa cum se poate vedea și în poza din dreapta.



-> public void start():

Pornește thread-ul *Producer*.

4.3. Buffer.java :

Obiectul de tip *Buffer* este folosit ca resursă comună, accesată de thread-urile *Producer* și *Consumer*.

Câmpuri:

Private BufferedImage subImage = fragmentul de imagine ce trebuie stocat;

Private Boolean available = semnalează dacă obiectul poate primii un fragment de imagine sau dacă acesta este ocupat;

Metode:

->public synchronized BufferedImage get() :

Această funcție este apelată de Consumer pentru a prelua fragmentul din Buffer.

->public synchronized void put(BufferedImage subImage) :

BufferedImage subImage: imaginea de stocat;

Această funcție este apelată de Producer pentru a stoca un fragment în Buffer.

4.4. Consumer.java :

Acesta este thread-ul care efectuează transformarea cerută.

Câmpuri:

Private Buffer imageBuffer = obiectul *Buffer* din care trebuie citit fragmentul de imagine;

Thread t = obiect de tip thread;

Private ObjectOutputStream out = stream-ul pentru scriere prin care se trimite informația prin calea de comunicație de tip pipe;

Metode:

->public Consumer(Buffer imageBuffer, ObjectOutputStream out) :

Buffer imageBuffer: obiectul *Buffer* din care trebuie preluat fragmentul;

ObjectOutputStream out: stream-ul pe care se va trimite informația prelucrată;

->public void start() : Pornește thread-ul Consumer.

->public void run() :

Se citesc pe rând fragmentele trimise de Producer și se salvează în vectorul de obiecte de tip BufferedImage *subImage.*

Se formează *originalImage* prin apelarea funcției statice din RGBImage sumImages(). Parametrii sunt convertiți la tipul RGBImage prin unul din constructorii clasei RGBImage care returnează un nou obiect de acest tip creat pe baza unui obiect de tip BufferedImage.

Se construiește imaginea convertită la spațiul YUV și cu contur de pixeli nuli, *convertedYUVImage,* prin apelarea constructorului clasei PaddedYUVImage care construiește un nou astfel de obiect dintr-un alt obiect de tipul RGBImage.

Se formează imaginea prelucrată *sharpenedYUVImage* prin apelarea funcției clasei PaddedYUVImage, sharpenImage(), care returnează un obiect de tipul YUVImage.

Se folosește un alt constructor al clasei RGBImage pentru a crea un nou obiect de acest tip pe baza obiectului de tip YUVImage, *sharpenedRGBImage*. Acesta va fi imaginea prelucrată, reprezentată pe spațiul de culori RGB.

Variabila *startTime* reține din nou momentul de timp curent, pentru a calcula mai departe timpul de prelucrare, dar apoi se repetă operația pentru a se calcula la final timpul de scriere al noii imagini.

La finalul acestei funcții, imaginea este împărțită în 5 fragmente și trimisă prin pipe la WriterResult pentru a fi scrisă.

4.5. RGBImage.java :

Această clasă extinde clasa *Image*. Conține informațiile specifice unei imagini reprezentate pe spațiul de culori RGB.

Câmpuri:

Private RGBPixel[] pixels = o matrice de obiecte de tip *RGBPixel*;

Private short width, height = lățimea, respectiv înălțimea unei imagini;

Metode:

->public RGBImage(BufferedImage img) :

BufferedImage img: o imagine de tip BufferedImage pe baza căreia se crează noul obiect de tip RGBImage;

Constructorul preia din *img* înălțimea, lățimea și pixelii folosind funcțiile lui BufferdImage getWidth(), getHeight() și getRGB().

Pentru preluarea fiecărui canala la fiecare pixel, se convertește valoarea întreagă returnată de getRGB() la un string de cifre binare. Din acest string se selectează cele 3 canale (R, G, B), se convertesc în numere întregi și se salvează în câmpurile corespunzătoare ale obiectului de tip RGBPixel.

->public RGBImage(YUVImage img) :

YUVImage img: obiect de tip YUVImage pe baza căruia de formează un nou obiect de tip RGBImage; Acesta reprezintă o imagine RGB convertită de la spațiul YUV;

Se preiau lățimea și înălțimea, iar pentru fiecare pixel, înainte ca acesta să se salveze, se convertește la formatul RGB prin funcția convertYUVtoRGB();

-> private RGBImage(short bigWidth, short bigHeight, RGBPixel[][] bigPixels)

short bigWidth: lățimea noii imagini;

short bigHeight: înălțimea noii imagini;

RGBPixel[][] bigPixels: matricea de pixeli a noii imagini;

Este un constructor privat, folosit doar în interiorul acestei clase pentru a crea un nou obiect de tip RGBImage pe baza unui set de elemente calculate în alte funcții;

->public RGBPixel getPixel(int x, int y) :

Int x, int y: coordonatele pixelului căutat;

Returnează pixelul de la poziția (x, y);

->public short getWidth() : getter pentru câmpul *width;*

->public short getHeight() : getter pentru câmpul *height;*

-> public static RGBImage sumImages(RGBImage...images) :

RGBImage...images: imaginile ce trebuie „lipite”;

Această funcție primește un număr variabil de parametrii. Procedeul de „lipire” începe prin calcularea lățimii totale (înălțimea se cunoaște deja, considerând că toate imaginile au aceeași înălțime).

În continuare, se construiește matricea pixelilor. Pentru acest lucru, se parcurge fiecare imagine în parte și se salvează la coordonatele corespunzătoare pixelii.

-> public BufferedImage convertToBufferedImage() :

Această funcție face conversia între tipul RGBImage și tipul BufferedImage. Se preiau lățimea și înălțimea, apoi pentru fiecare pixel, se concatenează valorile celor 3 canale astfel încât să se obțină un singur număr de format corespunzător care să poată fi setat pe pixelul noii imagini prin funcția setRGB() a clasei BufferedImage;

-> public RGBImage cutImage(int x, int y, int miniWidth, int miniHeight) :

int x, int y: coordonatele pixelului din stânga sus al secțiunii dorite;

int miniWidth: lățimea secțiunii dorite;

int miniHeight: înălțimea secțiunii dorite;

Această funcție returnează o secțiune de imagine pe baza coordonatelor și a dimensiunilor oferite. Pur și simplu salvează într-o nouă matrice de pixeli pixelii de pe pozițiie încadrate în coordonate și cheamă constructorul privat pentru a forma un nou obiect RGBImage.

4.6. RGBPixel.java :

Cuprinde valorile celor 3 canale de culoare pentru reprezentarea pe spațiul RGB.

Implementează interfața Serializable, deoarece este nevoie ca acest tip de obiect să fie serializabil pentru transmiterea prin pipe.

Câmpuri:

Private short r = valoarea canalului R;

Private short g = valoarea canalului G;

Private short b = valoarea canalului B;

Metode:

->public short getR() : getter pentru câmpul *r*;

->public short getG() : getter pentru câmpul *g*;

->public short getB() : getter pentru câmpul *b*;

->public void setR(short r) : setter pentru câmpul *r*;

->public void setG(short g) : setter pentru câmpul *g*;

->public void setB(short b) : setter pentru câmpul *b*;

->public YUVPixel convertRGBtoYUV() :

Variabilele *red, green, blue* păstrează valorile câmpurilor *r, g* și *b,* ca tipuri double, pentru a putea executa calculele.

Se calculează *y, u* și *v* conform formulelor menționate la început. Se ține cont că în cazul în care un rezultat depășește 255, acesta trebuie setat să aibă valoarea de 255 fix, pentru a nu altera culorile imaginii. În cazul în care un rezultat are valoare negativă, acesta trebuie setat 0, din aceleași motive.

Se declară un nou obiect de tip YUVPixel, căruia i se vor seta valorile celor 3 canale calculate. Acesta reprezintă obiectul returnat de funcție.

4.7. YUVPixel.java :

Cuprinde valorile celor 3 canale de culoare pentru reprezentarea pe spațiul YUV.

Câmpuri:

Private float y = valoarea canalului Y;

Private float u = valoarea canalului U;

Private float v = valoarea canalului V;

Metode:

->public float getY() : getter pentru câmpul *y*;

->public float getU() : getter pentru câmpul *u*;

->public float getV() : getter pentru câmpul *v*;

->public void setY(float y) : setter pentru câmpul *y*;

->public void setU(float u) : setter pentru câmpul *u*;

->public void setV(float v) : setter pentru câmpul *v*;

->public RGBPixel convertYUVtoRGB() :

Se calculează *r, g* și *b* (variabile de tip double) conform formulelor menționate la început. Se ține cont că în cazul în care un rezultat depășește 255, acesta trebuie setat să aibă valoarea de 255 fix, pentru a nu altera culorile imaginii. În cazul în care un rezultat are valoare negativă, acesta trebuie setat 0, din aceleași motive.

Se declară un nou obiect de tip RGBPixel, căruia i se vor seta valorile celor 3 canale calculate (convertite la tipul short),. Acesta reprezintă obiectul returnat de funcție.

4.8. YUVImage.java :

Această clasă extinde clasa *Image*. Conține informațiile specifice unei imagini reprezentate pe spațiul de culori YUV.

Câmpuri:

Private YUVPixel[] pixels = o matrice de obiecte de tip *YUVPixel*;

Private short width, height = lățimea, respectiv înălțimea unei imagini;

Metode:

->protected YUVImage(short width, short height) :

Short width, height : dimensiunile noii imagini;

Constructor ce crează un obiect YUVImage cu valorile tuturor pixelilor nuli;

Această metodă este folosită doar în cadrul clasei copil, PaddedYUVImage.

->protected YUVImage() :

Construiește un obiect YUVImage de dimensiuni nule și pixeli nuli.

->public YUVPixel getPixel(int x, int y) :

Int x, int y: coordonatele pixelului care se dorește selectat;

Funcția returnează pixelul de la coordonatele date. Este practic getter-ul pentru câmpul *pixels.*

->public short getWidth() : getter pentru câmpul *width.*

->public short getHeight() : getter pentru câmpul *height.*

->public short setPixel(int x, int y) :

Int x, y: coordonatele pixelului ce trebuie modificat;

Practic, este setter-ul câmpului *pixels.*

4.9. PaddedYUVImage.java :

Clasa extinde *YUVImage* și cuprinde atributele și metodele unei imagini reprezentate pe spațiul de culoare YUV, încadrată în chenarul de pixeli nuli.

Câmpuri:

Private static byte[][] sharpeningMask = matricea pentru procesul de „Image Sharpening”

Private static FileWriterClass writer = reprezintă obiectul care asigură accesul și gestionarea fișierului în care se scrie la final

Bloc de inițializare:

Aici doar se scrie la consolă faptul că a început procesul de conversie. Acesta este scris o singură dată, atunci când se crează obiectul de tip PaddedYUVImage și semnalează începutul procesului de prelucrare a imaginii.

Bloc static de inițializare:

Acesta se execută la inițializarea clasei, adică atunci când începe programul să ruleze. Aici se inițializează masca pentru transformare, cu valorile prezentate la început. De asemenea, se pregătește fișierul pentru scriere: se șterge cel deja existent, dacă deja mai exită unul, și se crează unul nou.

Metode:

->public PaddedYUVImage(RGBImage img) :

RGBImage img: imaginea originală, ce trebuie convertită;

Acest constructor primește ca argument un obiect de tipul RGBImage și crează unul nou obiect de tip PaddedYUVImage.

Se preiau înălțimea și lățimea, apoi se atribuie fiecărui pixel valoarea pixelului corespunzător din imaginea inițială, convertit la formatul YUV cu ajutorul funcției convertRGBtoYUV() din clasa RGBPixel.

-> private static YUVPixel applyConvolutionMask(YUVPixel[][] pixelsToConvolve) :

YUVPixel[][] pixelsToConvolve: reprezintă pixelii pe care se aplică masca de convoluție;

În această funcție se aplică efectiv masca de convoluție pe un grup de pixeli. Este foarte important de menționat faptul că modificări se fac doar la nivelul canalului Y. Valorile din canalele U și V rămân neschimbate, motiv pentru care pentru rezultat se vor considera pentru canalele U și V elementele de pe poziția [1][1] (mijlocul matricei de pixeli de prelucrat).

Pentru fiecare valoare din matrice se înmulțește aceasta cu coeficientul corespunzător din matricea de convoluție, apoi se însumează valorile, rezultatul păstrându-se în variabila *y*.

La finalul algoritmului, în obiectul nou creat *resultPixel* (care este de tip YUVPixel se atribuie fiecărui canal valoarea calculată. Obiectul este returnat ca rezultat al funcției.

->public YUVImage sharpenImage() :

Această funcție execută prelucrarea efectivă a imaginii.

Pentru fiecare pixel din imagine (se parcurg pixelii imaginii originale, nu și cei din conturul de pixeli nuli) se reține în matricea *pixelsToConvolve* pixelul respectiv și cei care i se învecinează. Această matrice este transmisă la funcția applyConvolutionMask(), care aplică procedura de convoluție pe grupul de pixeli.

Rezultatul funcției este de fiecare dată depus la locul corespunzător în imaginea *sharpenedImage,* care este și returnată la final.

4.10. WriterResult.java :

Această clasă implementează interfața Runnable și este folosită pentru crearea thread-ului care va scrie imaginea prelucrată la destinație.

Câmpuri:

Thread t = variabilă de tip thread;

Private String resultPath = calea unde va fi depusă imaginea prelucratăș

Private ObjectInputStream in= stream-ul pentru citire prin care vin fragmentele imaginii prelucrate;

private static FileWriterClass writer = obiectul care asigură accesul la fișierul în care se va scrie la final;

Metode:

-> public WriterResult(String resultPath, ObjectInputStream in) :

String resultPath: calea unde se va scrie imaginea prelucrată;

ObjectInputStream in: Stream-ul pentru citire;

Acesta este constructorul acestei clase.

->public void start() : pornește thread-ul *WriterResult.*

->public void run() :

Aici se desfășoară procesul de scriere a noii imagini.

Pe măsură ce se citesc cele 5 fragmente venite de la *Consumer*, ele sunt salvate în vectorul de obiecte RGBImage *sharpenedSubImage.*

Se formează imaginea finală, *sharpenedImage,* prin apelarea funcției din clasa RGBImage sumImages(), care „lipește" fragmentele primite.

Se cheamă din nou funcția din RGBImage convertToBufferedImage(), care convertește obiectul de tip RGBImage *sharpenedImage* în obiect de tip BufferedImage, care poate fi scris într-un fișier BMP.

La final, *writer* este folosit pentru a scrie într-un fișier .txt informații despre poza ce tocmai a fost prelucrată, împreună cu timpul total de execuție al programului.

4.11. FileWriterClass.java :

Această clasă implementează ipublic WriterResult(String resultPath, ObjectInputStream interfața *FileOperations* (care cuprinde operațiile de scriere, ștergere și creare de fișier).

Câmpuri:

Public static long startTime = variabilă statică care reține momentul de timp de început pentru o etapă a execuției programului și care este folosită pentru calcularea timpului de executare a unei etape;

Public static long startTime0 = are același rol ca și *startTime,* doar că este folosită pentru calculul timpului execuției întregului program;

Metode:

->public void writeToFile(String text) :

String text: textul ce se dorește scris în fișier;

Această funcție scrie un text dat într-un fișierul predefinit Rezultate.txt.

->public void deleteFile() : Șterge fișierul Rezultate.txt.

->public void createFile() : Crează fișierul Rezultate.txt.

5. Exemplu de execuție al programului:

Se preiau de la utilizatorul căile pentru citirea imaginii originale și pentru depunerea imaginii prelucrate:

```
De unde se citeste imaginea originala (Scrieti calea absoluta):
D:\Materiale educative\AWJ\Proiect\Poze_Originale\Pisica.jpg
Unde se depune imaginea prelucrata?:
D:\Materiale educative\AWJ\Proiect\Poze_Prelucrate\Pisica.bmp
```

Imaginea se prelucrează, apoi se depune la calea aleasă:

Pisica.jpg

Poza originală

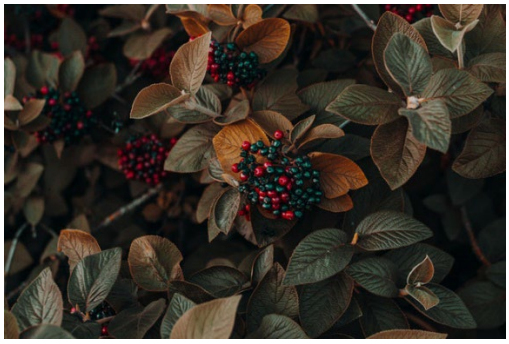
Poza prelucrată

Sample_1.bmp

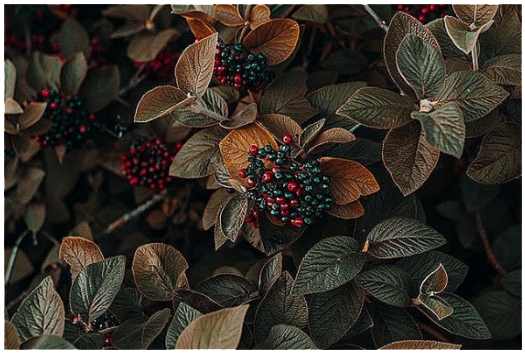
Poza originală

Poza prelucrată

Sample_2.bmp



Poza originală



Poza prelucrată