Facultatea de Automatică și Calculatoare

Specializare: Ingineria Sistemelor

Image Sharpening

Sectiune teoretică

Nozohor Aida

Coordonator:

profesor dr. ing. Hossu Andrei

Cuprins

1. Introducere

Acest proiect își propune dezvoltarea unei modalități de prelucrare a imaginilor folosind matrici de convoluție. În scopul realizarii acestuia, am explorat în profunzime conceptele de programare orientată pe obiecte, am dobândit competențe în utilizarea mediului de dezvoltare open-source Eclipse și am studiat teoria procesării imaginilor.

2. Partea teoretică

Tehnica de `Image Sharpening` este utilizata pe scară largă în industria fotografică.[1] Aceasta este un instrument puternic pentru a evidentia textura și a atrage atenția privitorului. Atunci când este utilizata în mod excesiv, pot apărea artefacte de sharppening inestetice. Pe de altă parte, atunci când este efectuată corect, sharpening-ul poate îmbunătăți adesea calitatea aparentă a imaginii chiar mai mult decât trecerea la un obiectiv de cameră de înaltă calitate.[2]

Cum functioneaza?

Cele mai multe instrumente software de sharpening funcționează prin aplicarea a ceea ce se numește "unsharp mask", care, în ciuda numelui său, acționează de fapt pentru a face mai clară o imagine.[2]

Unsharp Masking (USM) este o tehnică de îmbunătățire a imaginii, implementată pentru prima dată în camera obscură, dar care acum este folosită în mod obișnuit în programele de procesare a imaginilor digitale. Numele său provine de la faptul că tehnica utilizează o imagine negativă neclară sau "unsharp" pentru a crea o mască a imaginii originale. Masca "unsharp" este apoi combinată cu imaginea pozitivă originală, creând o imagine care este mai puțin neclară decât cea originală. Imaginea rezultată, deși mai clară, poate fi o reprezentare mai puțin exactă a subiectului imaginii.[3]

În exemplul de mai jos, imaginea este convolută cu următorul filtru de sharpening:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Această matrice este obținută cu ajutorul ecuației:

$$\text{sharpened} = \text{original} + (\text{original} - \text{blurred}) \times \text{amount}$$

folosind un kernel uniform cu 5 pixeli pentru imaginea "neclară" și 5 pentru multiplicatorul "cantitate":

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} /5 \right) 5 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Efectul de sharpening poate fi controlat prin variația multiplicatorului. Valoarea de 5 a fost aleasă aici pentru a obține un nucleu cu valori întregi, dar aceasta nu este o caracteristică obligatorie pentru această operațiune.[3]

3. Descrierea aplicației cerute

Această aplicație primește un ansamblu de imagini drept set de date de intrare, procedând apoi la prelucrarea acestora prin aplicarea a două tipuri distincte de matrici de convoluție, cunoscute sub denumirea de "masti". Datele sunt citite, iar imaginile sunt ulterior procesate și salvate în fișiere noi pentru a evidenția în mod clar modificările aplicate, facilitând astfel observarea transformărilor.

4. Descrierea implementării

1. Inițializarea Parametrilor: Metoda care implementează procesul de accentuare primește o imagine de tip BufferedImage și o serie de măști de convoluție sub forma unui număr variabil de parametri (utilizând varargs). De asemenea, se inițializează variabile pentru stocarea rezultatului și pentru măsurarea timpului de procesare.
2. Parcurgerea Pixelilor Imaginii: Se utilizează două bucle for pentru a itera prin fiecare pixel al imaginii.
3. Sumarea Valorilor Canalelor de Culoare: Se calculează sumele ponderate ale canalului de culoare pentru fiecare pixel din vecinătatea pixelului curent, utilizând măștile de convoluție.
4. Clamparea și Setarea Pixelilor în Imaginea Rezultat: Valorile rezultate sunt clampate în intervalul [0, 255], iar apoi pixelul rezultat este setat în imaginea de ieșire.
5. Măsurarea Timpului de Procesare și Returnarea Rezultatului: La final, timpul de procesare este măsurat, și rezultatul este returnat.

5. Descrierea structurală, arhitecturală și funcțională a aplicației

Din punct de vedere structural, aplicația este organizată în două pachete: `packTest` și `packWork`.

În pachetul `packTest` se găsește clasa `MyMain`, responsabilă de apelarea metodelor necesare prelucrării, aceste metode fiind definite în clasele incluse în al doilea pachet, `packWork`.

Pachetul `packWork` adăpostește toate clasele considerate esențiale pentru implementarea aplicației, cum ar fi: `IOImage`, `IOImageMirrored`, `ReadWrite`, `GetWidthHeight` și `GetWidthHeightAbstract` pentru gestionarea citirii/scrierii și `Sharpening` pentru implementarea propriu-zisă a procesului de accentuare (sharpening).

6. Descrierea modulelor

1. Clasa `MyMain`

Prin intermediul clasei `MyMain` din pachetul `packTest`, se orchestrează operațiile de citire, prelucrare și scriere a imaginilor folosind diverse măști de convoluție.

În primul rând, se creează instanțe ale clasei IOImage pentru trei imagini diferite: "dogDuck.bmp", "dog.bmp" și "chefDog.bmp". Ulterior, utilizatorul este invitat să introducă un număr (1 sau 2), care reprezintă masca de convoluție aplicată primei imagini, "dogDuck.bmp". În funcție de acest număr, se selectează una din cele două măști prestabilite (`mask1` sau `mask2`). Apoi, folosind metodele implementate, imaginile sunt prelucrate și salvate în fișiere separate, reflectând modificările aplicate prin filtrele de convoluție.

De asemenea, este prezentată o variantă de citire și prelucrare alternativă, prin intermediul clasei IOImageMirrored, care oglindește imaginea și aplică o mască de convoluție. Ultima parte a secvenței demonstrează implementarea procesării asincrone a imaginilor folosind thread-uri, evidențiind astfel abilitățile de gestionare eficientă a resurselor și paralelizare a operațiilor.

2. Clasa `IOImage`

Clasa `IOImage` servește drept nucleu al procesului de manipulare a imaginilor, bazându-se pe principiile programării orientate pe obiect și extinzând funcționalitatea prin intermediul claselor `Sharpening` și `ReadWrite`.

În primul rând, se definește o mască de convoluție (mask1) pentru realizarea procesului de accentuare a detaliilor în imagine. De asemenea, un bloc static de inițializare este inclus pentru a afișa un mesaj la încărcarea clasei, subliniind reușita citirii imaginilor.

Metodele `readImage` și `writeImage` sunt elaborate pentru a asigura funcționalitatea eficientă și cronometrarea fiecărei operațiuni. Pentru o abordare asincronă a procesării imaginilor, se utilizează două fire de execuție (`producerThread` și `consumerThread`). Prin intermediul obiectului de sincronizare (`lock`), aceste fire sunt coordinate într-un mod care permite producerea și consumul de imagini în mod sincronizat. Se evidențiază atât așteptarea notificării (`wait`), cât și transmiterea acesteia (`notify`) între fire pentru a realiza o comunicație eficientă.

În final, se demarează procesarea imaginilor, în care imaginea procesată este salvată sub numele "ImagineModificataThread.bmp".

3. Clasa `IOImageMirrored`

`IOImageMirrored` se ocupă de manipularea imaginilor și inversarea acestora într-un mod oglindit. Similar cu clasa IOImage, această variantă adaugă și afișează un mesaj de succes la încărcarea clasei, subliniind reușita citirii și inversării imaginilor.

Metoda `readImage` din această clasă reprezintă o alternativă pentru citirea imaginilor. În loc să doar citească imaginea, aceasta o inversează, returnând astfel imaginea oglindită. Implementarea utilizează `BufferedImage` pentru a realiza această operațiune în mod eficient, parcurgând fiecare pixel de la dreapta la stânga și de sus în jos în imaginea originală. Valoarea fiecărui pixel este preluată și setată în mod corespunzător în imaginea oglindită.

4. Clasa `ReadWrite`

Acest fragment de cod definește o interfață numită `ReadWrite`, care conține două metode: `readImage` și `writeImage`. Această interfață va fi implementată de către două clase diferite, contribuind astfel la utilizarea conceptului de polimorfism.

Interfața `ReadWrite` servește ca un contract pentru clasele care o implementează. Prin intermediul metodei `readImage`, aceasta specifică că o clasă ce implementează interfața este capabilă să citească o imagine, iar prin intermediul metodei `writeImage`, se precizează că aceasta poate scrie o imagine.

Prin aplicarea conceptului de polimorfism, două clase distincte pot implementa această interfață în mod diferit, adaptându-se specificităților lor individuale. Astfel, se promovează o abordare modulară și flexibilă a manipulării imaginilor, permitând schimbul facil între diferite implementări ale funcționalităților de citire și scriere.

5. Clasa `GetWidthHeight`

Clasa `GetWidthHeight` furnizează funcționalități pentru obținerea lățimii și înălțimii imaginii. Acest lucru este realizat prin metodele `calculateWidth` și `calculateHeight`, care parcurg pixelii imaginii în lungime și, respectiv, în înălțime, pentru a determina dimensiunile corecte. Variabilele de instanță `width` și `height` reține ultimele dimensiuni calculate și pot fi obținute prin metodele de acces `getWidth` și `getHeight`. Clasa este derivată dintr-o clasă abstractă (`GetWidthHeightAbstract`) și utilizează verificări pentru a evita eventualele probleme cu imaginile nule .

6. Clasa `GetWidthHeightAbstract`

Clasa abstractă `GetWidthHeightAbstract` definește un set de metode abstracte pentru manipularea dimensiunilor imaginii. Aceste metode includ `GetWidthHeight`, `getWidth`, `calculateWidth`, `GetHeightHeight`, `getHeight` și `calculateHeight`. Această clasă furnizează un schelet comun de funcționalitate, care este extins de către clasa concretă `GetWidthHeight` pentru a implementa logica specifică de obținere a dimensiunilor imaginii. Prin intermediul acestei abstracții, se promovează un design modular și o separare a responsabilităților între clasele care utilizează aceste funcționalități.

7. Clasa `Sharpening`

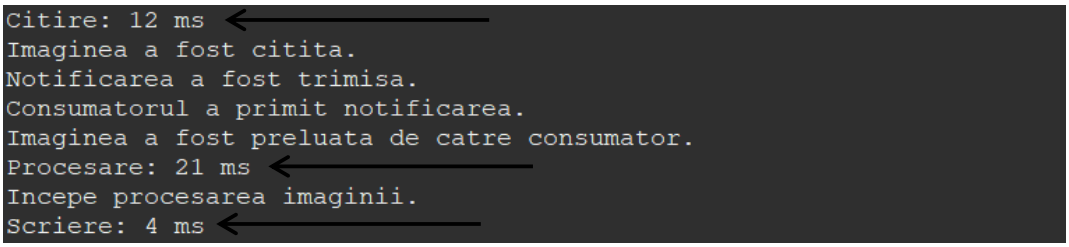
Clasa `Sharpening` extinde clasa `GetWidthHeight` și implementează logica pentru aplicarea măștilor de convoluție asupra imaginilor. Aceasta utilizează o metodă numită `applyConvolutionMask`, care primește o imagine și un număr variabil de măști de convoluție. În timpul aplicării, clasa calculează sume ponderate pentru canalele de culoare (roșu, verde, albastru și alfa) ale fiecărui pixel, folosind măștile specificate.

Procesul implică parcurgerea fiecărui pixel al imaginii și a pixelilor din jurul său. Valorile pixelilor sunt extrase și înmulțite cu valorile corespunzătoare din măștile de convoluție. Suma rezultată este utilizată pentru a calcula noile valori ale canalelor de culoare. Rezultatul final este o imagine nouă cu detalii accentuate.

Blocul non-static de inițializare este utilizat pentru a afișa un mesaj atunci când clasa este inițializată. Acesta oferă o notificare că procesul de prelucrare a imaginilor a fost implementat cu succes.

7. Evaluare performanțe

Eficiența operațională poate fi evaluată prin analiza timpului de execuție înregistrat pentru fiecare fază distinctă a procesului. Această metodă de evaluare ne permite să evidențiem rezultate favorabile în ceea ce privește cele trei etape esențiale: importul datelor, aplicarea tehnicii de accentuare a detaliilor și procesul de scriere a rezultatelor obținute.



8. Concluzii

Am abordat fiecare porțiune de cod cu atenție, evidențiind aspectele de gestionare a resurselor, manipularea eficientă a imaginilor și implementarea unui model asincron pentru a îmbunătăți performanța generală.

În clasa IOImage, am creat metode robuste pentru citirea și scrierea imaginilor, punând un accent deosebit pe tratarea adecvată a situațiilor excepționale. De asemenea, am introdus un model asincron pentru a eficientiza procesul de procesare a imaginilor, folosind două thread-uri care interacționează în mod corespunzător.

În ceea ce privește clasa GetWidthHeight, am construit funcționalități utile pentru extragerea dimensiunilor imaginilor, aducând astfel o abordare modulară în manipularea acestora. Aceasta facilitează accesul și utilizarea informațiilor despre dimensiuni într-un mod clar și simplificat.

În implementarea funcționalităților de prelucrare a imaginilor, am abordat cu atenție procesul de sharpening, folosind măști de convoluție. Această tehnică are scopul de a evidenția detaliile și de a îmbunătăți aspectul general al imaginilor. Prin aplicarea măștilor, am căutat să obțin o îmbunătățire a calității vizuale a imaginilor procesate.

În plus, am introdus și un sistem simplu de măsurare a timpului de execuție pentru fiecare etapă a procesului. Acest aspect ne permite să evaluăm eficiența algoritmilor implementați în funcție de durata totală a operațiunilor.

În esență, abordarea mea în implementarea procesului de sharpening vizează îndeplinirea cerințelor funcționale și aplicarea unei tehnici de prelucrare a imaginilor cu impact pozitiv asupra calității acestora.

9. Documentatie cod sursa

1. Cursurile furnizate si prezentate de catre domnul profesor dr. ing. Hossu Andrei
2. https://www.w3schools.com/java/java_oop.asp
3. https://www.geeksforgeeks.org/java/?ref=lbp
4. https://www.educative.io/blog/object-oriented-programming

10. Bibliografie

1. https://archive.nptel.ac.in/content/storage2/courses/117104069/chapter_8/8_32.html
2. https://www.cambridgeincolour.com/tutorials/image-sharpening.htm
3. https://en.wikipedia.org/wiki/Unsharp_masking