

Academia de Studii Economice din București

Facultatea de Cibernetică, Statistică și Informatică Economică

Specializarea Informatică Economică

Tehnici de procesare a imaginilor

Documentația completă a proiectului

Coordonatori:

Prof.univ.dr Cătălina COCIANU

Prof.univ.dr Cristian USCATU

Student

Ioana-Diana PETRE

București 2021

Cuprins

Contents

[Temele abordate 3](#_Toc60593448)

[Partea I 3](#_Toc60593449)

[Partea II 4](#_Toc60593450)

[Partea III 5](#_Toc60593451)

[Partea IV 5](#_Toc60593452)

[Partea V 6](#_Toc60593453)

# Temele abordate

În acest capitol sunt descrise temele alese, modul de implementare pentru acestea, datele de intrare necesare, rezultatele obținute și exemple de apel pentru a putea rula sursele.

Fiecare sursă prezintă funcții pentru citirea unei imagini și afișarea acesteia cu un titlu ales. Imaginile cu care se lucrează sunt monocrome.

## Partea I

Implementarea reprezentării Karhunen-Loeve pentru seturi de imagini de dimensiune mai mare sau egală de 250 x 250. Imaginile sunt împărțite în blocuri de dimensiune egală.

Descrierea algoritmului pentru imagini cu dimensiuni mai mici de 250x250 pixeli:

* Se va citi un numar ales de imagini inițiale. Este important ca imaginile să fie monocrome și să prezinte aceeași dimensiune și tematică,
* Imaginile inițiale sunt liniarizate,
* Se va calcula medie apoi datele liniarizate vor fi centrate,
* Se va calcula matricea de covarianță de selecție, apoi pe baza acesteia vom putea obține valorile și vectorii proprii asociați,
* Determinăm componentele principale apoi reținem primele ”nrc” componente, unde ”nrc” este o constantă primită ca parametru,
* Reprezentăm datele folosind doar componentele reținute anterior,
* Reconstruim imaginile folosind reprezentarea primelor ”nrc” componente,
* Adăugăm medie și revenim la forma matriceală,
* Afișăm și salvăm imaginile obținute.

Pentru imaginile de dimensiuni mai mari de 250x250 pixeli trebuie să aplicăm algoritmului Karhunen-Loeve pe câte un bloc din fiecare imagine.

În exemplul implementat în acest proiect au fost alese imagini de dimensiunea 250x250 pixeli. Fiecare imagine a fost împărțită în 25 de blocuri de dimensiunea 50x50 pixeli.

Algoritmul se aplică pentru fiecare set curent de blocuri (i,j) și se vor obține următoarele variabile:

* Un masiv cu 3 dimensiuni în care sunt stocate infromațiile despre fiecare bloc dintr-o imagine, *A = zeros(50, 50, nrp);*
* Un masiv cu 2 dimensiuni în care stocăm liniarizarea fiecărui bloc din imaginea curentă, *bloc\_liniarizat = zeros(nr\_total\_elemente\_bloc, nrp);*
* Un masiv cu 2 dimensiuni pentru păstrarea rezultatelor în urma calculul mediei, *vector\_medie = zeros(nr\_total\_elemente\_bloc, 25);*
* O matrice de vectori proprii, *matrice\_vp = zeros(nr\_total\_elemente\_bloc, nrc, 25);*
* O matrice de reprezentare, *matrice\_rep = zeros(nrc, nrp, 25);*
* Pentru reconstrucția imaginilor sunt folosite masivele:
  + im\_noi\_l = zeros(nr\_total\_elemente\_bloc, nrp);
  + im\_noi = uint8( zeros(50, 50, nrp, 25) );

Este important ca la fiecare pas să actualizăm variabilele de mai sus. Deoarece lucrăm cu subimagini de dimensiunea 50x50 vom obține, pentru fiecare imagine, 25 de blocuri pe care le vom prelucra. Locația blocului curent este dat de rang-ul matricei: *(i-1) \* n1 + j. Unde* i, j este blocul curent iar n1 este 5.

Pentru o primă împărține a unei matrice aleatoare s-a construit **scriptul test\_impartire\_blocuri**, ulterior preluând conținutul acestui script în funcția **function [blocuri] = impartire\_blocuri(img).** Rezultatul acestei funcții întoarce un masiv *blocuri = zeros(m1, n1, m/m1, n/n1),* unde m1 și n1 sunt 50 (dimensiunea blocului).

Imaginile restaurate sunt afișate și salvate. Pentru a verifica modificările algoritmului aplicat am construit un **script test\_verificare\_imagini**pentru a afișa diferențele de pixeli dintre imaginile inițiale și cele restaurate.

Funcția k\_h\_l(nrc, nrp) implementează algorimul prezentat, unde **nrc** este numărul de componente pe care dorim să îl păstrăm (este important ca acestă variabilă să fie semnificativ mai mica decât dimensiunea imaginilor alese) și **nrp** numărul de imagini pe care dorim să aplicăm algoritmul.

Exemple de apel:

* k\_h\_l(20, 5),
* k\_h\_l(15, 7),
* test\_impartire\_blocuri, script-ul de împărțire pe bloculețe (optional),
* test\_verificare\_imagini, pentru script-ul de afișare a diferențelor (optional).

## Partea II

Implementarea procedurii unsharp masking.

Construim o mască, dimensiunea este dată ca parametru. Inițial masca se va inițializa cu valoarea 1 apoi vom aduna elementele acestei matrice. Vom împărți elementele inițiale ale măști cu suma calculată anterior.

Descrierea algoritmului:

* Se citește imaginea inițială și se convertește către una grayscale dacă este cazul,
* Construim masca de dimensiunea dată,
* Construim o imagine nivelată apicând un filtru de tipul medie asupra imaginii inițiale,
* Procedura de unsharp masking rezultă din calculul imaginii inițiale – imaginea nivelată + imaginea inițială,
* Afișăm rezultatul obținut în urma acestui calcul.

S-a observat că pentru o imagine inițială perturbată rezultatele sunt foarte bune. Pentru o imagine inițială clară nu putem observa atât de bine beneficiile algoritmului.

Funcția ***unsharp\_masking( img, dimensiune\_masca )*** implemenetează algoritmul descris anterior, unde ***img*** este imaginea inițială iar ***dimensiune\_masca*** reprezintă dimensiunea măștii pentru care aplicăm filtrul de nivelare medie.

Exemple de rulare:

* unsharp\_masking('img.png', 11);
* unsharp\_masking('img.png', 5);
* unsharp\_masking('img\_perturb.bmp', 5);

## Partea III

Implementarea filtrului trece-jos gaussian.

Descrierea algoritmului:

* Se citește imaginea inițială și se convertește către una grayscale dacă este cazul,
* Imaginea inițială trebuie expandată având de două ori mai multe linii și coloane. Este important să reținem centrul acesteia înainte de expandare,
* Se va centra imaginea expandată,
* Imaginea centrată o vom transforma în domeniul frecvențelor,
* Construim filtrul H utilizând următoarea formulă: , unde D0 este o rază dată a filtrului trece-jos gaussian iar reprezintă distanța de la (n,m) la centrul de simetrie al imaginii calculat la început,
* Se va calcula matricea G care rezultă prin înmulțirea element cu element a imaginii centrate în domeniul frecvențelor și filtrul trece-jos gaussian,
* Reconstruiesc imaginea filtrată revenid în domeniul spatial,
* Elimin centrarea asupra imaginii reconstruite,
* Extrag rezultatul dorit revenind la dimensiunile imaginii inițiale,
* Afișez rezultatul.

Funcția ***filtru\_low\_pass\_gauss(img, D0)*** implementează acest algoritm, unde ***img*** reprezintă imaginea inițială și ***D0*** raza filtrului gaussian.

Folosind o imagine de dimensiuni mai mici, s-a observat că cu cât raza este mai mare cu atât imaginea rezultată va fi mai fin nivelată.

Exemple de apel:

* filtru\_low\_pass\_gauss('img.png', 50);
* filtru\_low\_pass\_gauss('img.png', 30);

## Partea IV

Implementarea filtrului median adaptiv. Acest tip de filtrare elimină zgomotul de tip impuls (sare, piper) cu o probabilitate de apariție mare și păstrează detalii de tip contur.

Algoritmul extinde imaginea perturbată folosind o mască de dimensiune impară. La început masca va avea dimensiunea 3x3. Extinderea imaginii se face adăugând (dimensiunea măștii curente + 1) / 2 elemente de 0 la frontierele.

Vom extrage o mască curentă pentru fiecare pixel de dimensiune d, apoi aceasta se va liniariza și sorta pentru a putea obține valoarea minima, maximă și mediană din șirul de gri curent.

Indicii alpha1, alpha2, beta1 și beta2 sunt calculați pe baza valoriilor obținute anterior astfel:

* alpha1 = val\_mediana - val\_min;
* alpha2 = val\_mediana - val\_max;
* beta1 = fc(i, j) - val\_min;
* beta2 = fc(i, j) - val\_max;

fc(i, j) reprezintă valoarea pixelului din imaginea perturbată de la coordonatele (i,j).

Descrierea algoritmului:

* Se citește imaginea inițială și se convertește către una grayscale dacă este cazul,
* Se va perturba imaginea inițială cu un zgomot de tip impuls având o probabilitate de apariție mare,
* Parcurgem fiecare pixel al imaginii perturbată,
* Pentru fiecare pixel definim inițial dimensiuea măștii ca fiind 3 și extindem imaginea,
* Se obține o mască curentă și o folosim pentru a calcula inițial indicia alpha1, alpha2,
* Dacă alpha1 > 0 și alpha2 < 0, atunci vom calcula indicia beta1 și beta2,
* Dacă beta1 > 0 și beta2 < 0 atunci vom păstra valoarea pixelului de la coordonatele curente în imaginea filtrată, altfel vom păstra valoarea mediană corespunzătoare șirului de gri curent (al măștii),
* Dacă alpha1 și alpha2 nu respectă condiția impusă mai sus vom extinde dimensiunea măștii,
* Acești pași vor fi reluați pentru fiecare pixel în parte atâta timp cât dimensiunea măștii nu depășește un prag stability.
* După ce acești pași sunt parcurși pentru fiecare pixel vom afișa imaginea rezultată în urma filtrării.

S-a observat că cu cât mărim pragul dimensiunii măștii cu atât imaginea rezultată va fi din ce în ce mai blurată.

Funcția ***filtrare\_median\_adaptiva( img, d\_max )*** implementează algoritmul descris mai sus, unde ***img*** reprezintă imaginea inițială iar ***d\_max*** dimensiunea maximă pentru care putem extinde masca.

Exemple de apel:

* filtrare\_median\_adaptiva(’img.png’ , 7)
* filtrare\_median\_adaptiva(’img.png’ , 11)

## Partea V

Implementarea filtrului invers în domenioul frecvențelor. Filtrul invers se aplică în cazul motion blur în varianta continua pe axa OX (sau OY).

Este necesar să perturbăm o imagine prin inducerea efectului de mișcare în cazul continuu. **Funcția perturba\_mb\_continuu( img\_originala, T, axa )** implementează acest procedeu, unde ***img\_originală*** este imaginea curată, cea pe care doresc să o perturb, ***T*** reprezintă timpul iar ***axa*** poate fi ”x” sau ”y”.

Descrierea acestui procedeu:

* Citim imaginea inițială,
* Calculez transformata Furie pentru imaginea originală,
* Construiesc filtrul motion blur pentru domeniul continuu,
* Aplic perturbarea în domeniul frecvențelor, practic înmulțesc punct cu punct transformata imaginii inițiale cu filtrul,
* Revin în domeniul real și calculez imaginea rezultat, perturbată,
* Afișez rezultatul obținut și îl salvez.

Pentru a constui filtrul H se folosesc următoarele formule:

* Pentru axa OX:
* Pentru axa OY:

S-a observat că pentru a avea o imagine perturbată cât mai uniform valorile pentru variabilele ”a” și ”b” ar trebui să fie cât mai mici. Pentru variabilele a și b s-a ales implementarea funcției cu valoarea de 0.03.

**Funcția function [TFD\_h] = construieste\_filtru\_mb\_c(linii, coloane, T, axa)** construiește acest filtru. Unde ***linii*** reprezintă numărul de linii al imagini, ***coloane*** numărul de coloane, ***T*** este timpul și ***axa*** poate fi „x” sau „y”.

Implementarea filtrului invers:

* Citim imaginea perturbată,
* Se calculează transformata acesteia în domeniul frecvențelor,
* Construim filtrul motion blur,
* Aplicăm metoda ”filtru invers” pentru a putea obține o imagine mai puțin perturbată. Dacă modulul valorii filtrului dintr-un pixel este mai mare decât un prag stability atunci vom împății transformata Furie a imaginii perturbate la filtru. În caz contrar vom considera că pixelul este negru în acel punct,
* Revenim în domeniul spațial, eliminăm eventualele reziduri complexe și extragem imaginea rezultată,
* Afișăm rezultatul.

**Funcția function [] = filtru\_invers( img\_perturbata, T, axa, prag\_0)** aplicăfiltrul invers pentru o imagine perturbată în cazul motion blur varianta continuă, unde ***img\_perturbată*** este imaginea perturbată, ***T*** este timpul, ***axa*** poate fi „x” sau „y” și ***prag\_0*** reprezintă pragul pentru verificarea modulul valorii filtrului dintr-un pixel.

Exemple de apel:

* Pentru a obține o imagine perturbată:
* perturba\_mb\_continuu('img.png', 1, 'y');
* perturba\_mb\_continuu('img.png', 1, 'x');
* Pentru a aplica filtrul invers:
  + filtru\_invers('img\_p\_DC\_1\_axa\_x.png', 1, 'x', 0.09);
  + filtru\_invers('img\_p\_DC\_1\_axa\_y.png', 1, 'y', 0.1);