

Algoritmi moderni pentru compresia imaginilor

Raport de proiect
Procesarea Semnalelor

Cristian Borcan
Maria Telea
Andreea Draghioti

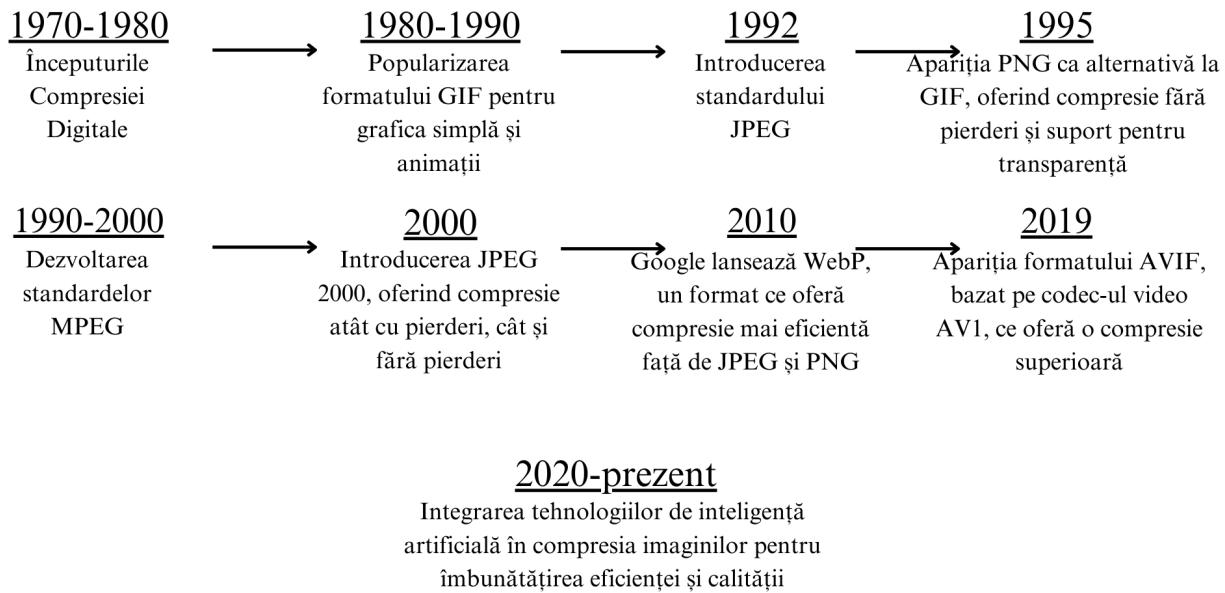
1. Cuprins:

1. Cuprins:	2
2. Introducere.....	3
2.1 Scurt Istoric.....	3
2.2 Lossless vs Lossy.....	3
3. De ce avem nevoie de image compression?	5
4. Descriere tehnică.....	6
4.1 Subsampling & downsampling.....	6
4.1.1 Downsampling.....	6
4.1.2 Subsampling.....	7
4.1.3 Reconstruirea imaginii după subsampling/ downsampling.....	7
4.2 Transform coding.....	8
4.2.1 Discrete Cosine Transform (DCT).....	8
4.3 Cuantizare.....	10
4.4 Machine Learning K-Means.....	11
5. Tehnologiile folosite.....	12
6. Rezultate.....	13
7. Concluzii.....	18
8. Bibliografie.....	19

2. Introducere

2.1 Scurt Istorico

De la nașterea sa în anii '70 și până în prezent, domeniul compresiei imaginilor digitale a parcurs un drum lung, marcând progrese semnificative în dezvoltarea tehnologică și optimizarea stocării datelor.



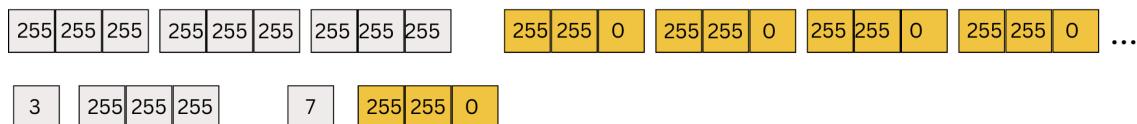
Cronologia prezentată mai sus arată avansările înregistrate în tehnologia de compresie a imaginilor, marcând etapele semnificative ce au influențat acest domeniu esențial în lumea modernă și era digitală.

2.2 Lossless vs Lossy

Ideal vrem ca fișierele noastre să fie cât mai mici posibil, pentru a stoca cât mai multe și a fi transmise mai rapid. Aici intervine compresia, prin intermediul căreia comprimăm datele într-o

dimensiune mai mică. Pentru a face acest lucru, trebuie să codificăm aceste date folosind mai puțini biți.

O metodă de a compresa datele constă în a reduce informația inutilă sau care se repetă. Să presupunem că avem o imagine formată din 3 pixeli albi și 7 pixeli galbeni.



Un total de 10 pixeli care la rândul lor folosesc câte 3 biți pentru a stoca informații despre culoare. Asta ar presupune stocarea următoarelor informații: pixel alb (255,255,255), pixel alb (255,255,255), pixel alb (255,255,255), pixel galben (255,255,0), pixel galben (255,255,0) etc. având un consum total de 30 de biți. Cum putem comprima aceste date? Vom reduce informația care se repetă folosind RUN-LENGTH ENCODING. Așadar, vom insera niște biți care stochează lungimea informației care se repetă și vom avea: [3] [pixel alb (255,255,255)] [7] [pixel galben (255,255,0)], având un total de 8 biți. Astfel, am economisit 22 de biți. Am eliminat aproximativ 73% din memoria ocupată inițial fără a pierde date. Acest fel de compresie se numește LOSSLESS compression, deoarece nu pierdem nimic, informația decompresată este identică cu cea originală înainte de compresie. Compresia lossless este esențială în situații în care orice pierdere de date poate cauza probleme, cum ar fi coruperea datelor sau rezultate inexacte. Menținerea integrității datelor originale este esențială pentru anumite fișiere, precum: fișierele de imagine medicală (imaginile RMN), fișierele audio din producțiile muzicale profesionale, date de backup, arhivele de tip zip, anumite tipuri de imagini (PNG, spre deosebire de JPEG, este un format lossless pentru imagini, fiind preferat pentru cazuri care necesită detalii fine, cum ar fi logo-uri sau grafica web) etc.

Cu toate acestea, există și alte tipuri de fișiere unde mici modificări ale informației pot avea loc, prin eliminarea datelor care nu sunt importante sau necesare, în special cele pe care percepția umană nu este capabilă să le detecteze.

De exemplu, în cazul imaginilor, anumite nuanțe de culoare pot fi îndepărtate dacă depășesc capacitatea ochiului uman de a le distinge. În audio, compresia lossy elimină frecvențele de sunet care sunt acoperite de sunete mai puternice sau nu sunt detectabile de către urechea noastră. În contextul video, există un proces care redă un anumit număr de cadre pe

secundă (FPS frames per second) care poate varia între 24 și 60 de cadre pe secundă în cazul videoclipurilor de pe Youtube. Aici se poate folosi metoda redundanței temporale. Spre exemplu, în ceea ce privește prezența unui fundal, nu trebuie să retransmitem anumiți pixeli în fiecare cadru. Majoritatea standardelor de compresie video utilizează o tehnică numită predicție compensată de mișcare, codificând blocuri de pixeli prin referire la o altă zonă din aceeași imagine (numită intra-predicție) sau din altă imagine (numită inter-predicție). High Efficiency Video Coding (HEVC) este un standard de compresie video dezvoltat recent ce poate defini blocuri de până la 64×64 pixeli.

Acest principiu al reducerii preciziei în concordanță cu percepția umană stă la baza faimoasei compresii JPEG. Suntem buni la a detecta contrastele, dar nu și la a detecta schimbările subtile de culoare. Cum am descoperit și la tema 1, JPEG profită de acest lucru prin împărțirea imaginilor în blocuri de 8×8 pixeli, eliminând mare parte din datele cu frecvență înaltă.

3. De ce avem nevoie de image compression?

Stocarea imaginilor pe dispozitive cu un spațiu de memorie limitat sau transmiterea rapidă a fișierelor pe internet sunt câteva dintre motivele care au dus la necesitatea de a găsi modalități eficiente de manipulare a imaginilor.

Image compression este un proces prin care se reduce mărimea imaginilor. În contextul compresiei de tip lossy, o imagine comprimată va ocupa mai puțini biți în memorie, fapt care aduce mai multe beneficii.

Unul dintre ele este reprezentat de “storage efficiency”. Diferențele de spațiu de stocare utilizat pentru a păstra o imagine neprocesată și una comprimată sunt mari. Conform unui articol de la Sony, o poză RAW făcută cu o cameră de la brand-ul lor ocupă 100MB, iar una comprimată folosind o metodă lossy ocupă 52 MB. Vedem, deci, o scădere semnificativă în ceea ce privește memoria ocupată de imaginile comprimate. Acest lucru se aplică și când vine vorba de stocarea video-urilor. HEVC, un cunoscut standard de comprimare al video-urilor, poate atinge un raport de compresie a datelor de până la 1000:1.

Un alt avantaj constă în utilizarea redusă a lățimii de bandă care este definită drept rata maximă de transfer al datelor în contextul internetului sau al unei rețele și se măsoară în biți per secundă. Utilizarea redusă a lungimii de bandă aduce cu sine și o experiență mult mai bună a

utilizatorilor diferitelor aplicații, fie web sau mobile. Imaginele comprimate se încarcă mult mai rapid în UI datorită numărului redus de biți pe care sunt reprezentate.

4. Descriere tehnică

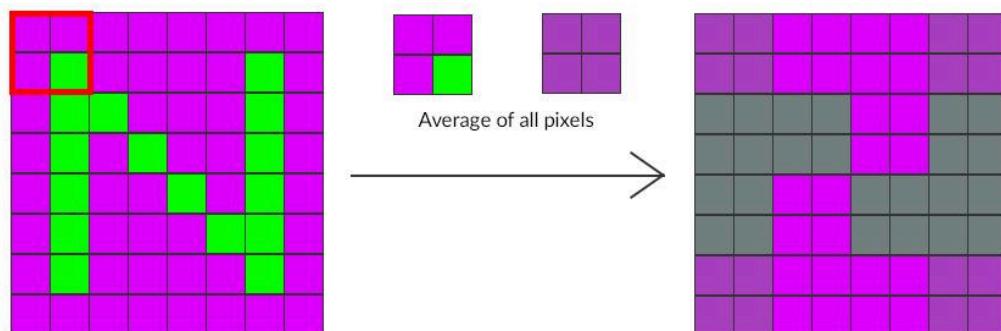
4.1 Subsampling & downsampling

Ideea cheie a acestor procese este de a lua mai puține samples în considerare din canalele Chroma blue și Chroma red, deoarece ochii noștri sunt mai puțin sensibili la culori, dar mai responsive la lumină și contrast (canalul Y). Tocmai de aceea ne permitem să pierdem informații care țin de culori.

4.1.1 Downsampling

Fie o imagine de 8x8 pixeli. Parcurgem imaginea în blocuri mai mici, de exemplu 2x2 și calculăm media tuturor pixelilor din acel bloc. Setăm apoi valoarea fiecărui pixel dintr-un bloc la media obținută.

Chroma downsampling 4:2:0

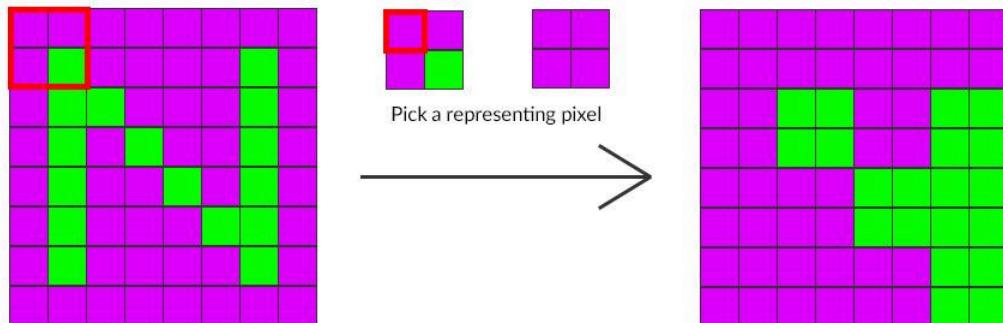


Exemplu downsampling

4.1.2 Subsampling

Pe aproximativ același principiu ca și la downsampling, imaginea este parcursă în blocks mai mici și pentru fiecare block se alege o valoare a unui pixel din block (de obicei cel din colțul stânga sus) care va fi setată pentru fiecare pixel din block-ul respectiv.

Chroma subsampling 4:2:0



Exemplu subsampling

4.1.3 Reconstruirea imaginii după subsampling/ downsampling

La final, după ce canalele Chroma blue și Chroma red au fost “preprocesate”, acestea se contopesc cu canalul Y care reprezintă componenta cu cea mai multă informație vizuală, care are cel mai mare impact asupra ochiului.

4.2 Transform coding

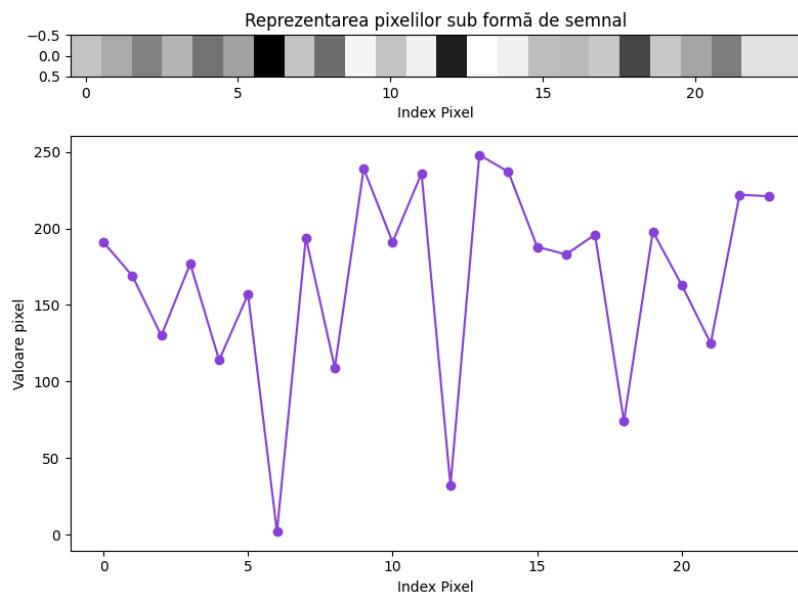
Una dintre cele mai folosite tehnici în compresia imaginilor este Transform Coding. Aceasta presupune transformarea reprezentării unei imagini din domeniul spațial (spatial domain) în domeniul frecvențial (frequency domain). Mai specific, vorbim despre trecerea de la reprezentarea sub formă de matrice a intensității unui pixel (în cazul imaginilor grayscale) sau a vectorului 3D de matrice 2D (în cazul imaginilor RGB), la o reprezentare sub formă de semnale și coeficienți, spre exemplu prin folosirea Transformatei Fourier. Această tehnică este utilizată deoarece procesarea imaginilor poate fi făcută mai ușor atunci când operațiile sunt efectuate

asupra semnalelor, precum manipularea acestora în funcție de intensitatea schimbărilor între pixeli, proces despre care vom vorbi în continuare.

4.2.1 Discrete Cosine Transform (DCT)

Cea mai des întâlnită formă de compresie din categoria Transform Coding este DCT, sau Discrete Cosine Transform. Aceasta este un tip de transformare înrudită cu transformata Fourier. Pe scurt, o transformată fourier reprezintă procesul prin care descompunem un semnal în suma unor funcții trigonometrice. Se numește transformată deoarece aceasta transformă date dintr-un anumit tip (amplitudine/timp) într-o listă de coeficienți ce corespund unor funcții trigonometrice, oferindu-ne posibilitatea de a descompune astfel un semnal în părți simple și totodată capacitatea de a reconstrui semnalul în întregime din acestea, fără a pierde date pe parcurs, astfel DCT în sine poate fi încadrată în categoria compresiilor lossless.

În contextul imaginilor, ne putem imagina fiecare rând de pixeli din imagine sub forma unui semnal, punctele din semnal fiind valorile intensității acelui pixel, precum în următoarea imagine:



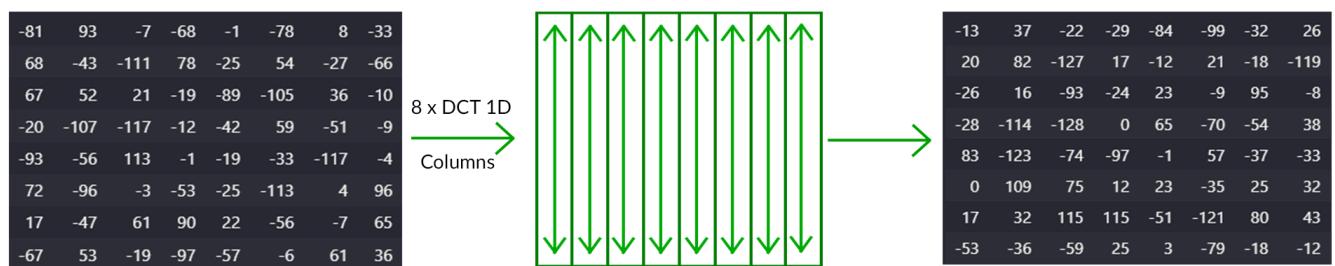
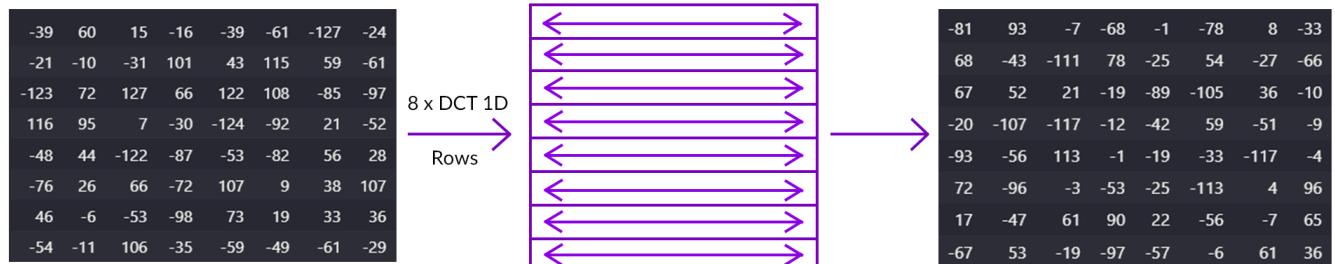
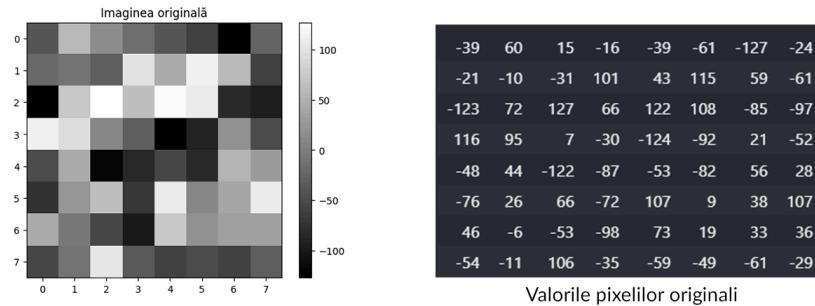
Având acestea în minte, putem continua cu generarea funcțiilor cosinusoidale folosind DCT. Pentru un sir de pixeli de lungime n , va fi generată o matrice de $n * n$, unde fiecare rând

corespunde unui semnal cosinusoidal diferit. Reprezentarea semnalului cosinusoidal în matrice este dată de formula:

$$X_{i,j} = \cos\left(\frac{\pi}{N} * i * (j + \frac{1}{2})\right)$$

, unde N este lungimea totală a matricei. Astfel, putem deduce că o valoare mai ridicată a lui i corespunde unei unde cosinusoidale cu o frecvență mai mare.

Acum că am generat o matrice DCT, putem căuta coeficienții pentru rândul nostru de pixeli prin multiplicarea valorii intensității pixelului cu fiecare element x din matricea generată, din care va rezulta o matrice a coeficienților DCT. Pentru o imagine 2D vom face același lucru, dar vom aplica DCT o dată pe fiecare rând, iar apoi pe fiecare coloană.



4.3 Cuantizare

Cuantizarea este, în termeni simpli, procesul prin care eliminăm elementele din imagine pe care ochiul uman nu le percep; practic se elimină din imagine componente de frecvență înaltă (treceri bruste între culorile pixelilor). Acest lucru este realizat prin împărțirea matricei rezultate din DCT cu o tabelă de cuantizare pre-stabilită (de obicei avem una pentru Y - luminance, și una pentru Cb, Cr- chrominance) și rotunjirea la cel mai apropiat număr întreg.

În esență, acest proces rezultă în câțiva pixeli importanți din bloc-ul inițial de pixeli (se află, de regula, în colțul stanga sus), lângă un număr mare de pixeli neimportanți cu valoarea zero.

O metodă de cuantizare populară în cazul culorilor se numește Color Quantization, și rezultă în reducerea culorilor unei imagini la un set limitat de culori semnificative pentru imagine, set stabilit prin algoritmi diferenți. Scade astfel numărul utilizat de culori pentru a reprezenta noua imagine fără a neglijă asemănarea acesteia cu cea originală.

Median cut este una dintre cele mai folosite metode de Color Quantization și presupune împărțirea recursivă a spațiului de culori folosit, în punctul median al acestora. De exemplu, pentru a reduce o imagine la un număr dorit de n culori, se poate aplica Median Cut pentru a găsi cele n culori relevante pornind de la imaginea originală.

4.4 Machine Learning | K-Means

Îmbunătățirile recente din domeniul inteligenței artificiale au fost transpusă și în domeniul procesării imaginilor, în special în compresia acestora. Folosind algoritmi de machine learning, putem ajunge la metode mult mai eficiente și la rezultate extrem de bune din punct de vedere al raportului dimensiune-calitate, acestea funcționând de multe ori împreună cu metode clasice de compresie. Un exemplu bun și ușor de înțeles este K-Means clustering, ce se bazează pe Color Quantization, dar alege în mod "inteligent" cele mai bune culori pe care imaginea le folosește, spre deosebire de clasicul Median Cut, ce se bazează pe un algoritm inflexibil.

K-Means Clustering este o tehnică de învățare nesupervizată, folosită clasice pentru gruparea datelor în funcție de similaritățile lor. Putem profita de această proprietate pentru a

grupa, în cazul nostru, culorile pixelilor. În esență, procesul de compresie a imaginilor folosind acest algoritm are următorii pași:

- a. Inițializarea centroizilor, care se face în primă fază în mod aleator (prima iterare nu va fi niciodată reprezentativă pentru culorile optime).
- b. Asignarea valorii centroidului la fiecare pixel în funcție de distanța dintre ele. Acest proces este realizat folosind distanța euclidiana.
- c. Updatarea centroizilor - O dată ce pixelii au primit valoarea nouă, mutăm centroizii în mijlocul cluster-ului de pixeli, prin media acestora (mean).
- d. Repetăm pașii până când centroizii nu își mai schimbă poziția. În acest moment am ajuns la convergență.

5. Tehnologiile folosite

- *Numpy*: diverse operații ce ne ajută în image processing
- *Matplotlib*: bibliotecă din Python care oferă funcții pentru plotări și afișări.
 - *pyplot*: plotarea diverselor date cu care am lucrat
- *Scipy*: pentru algoritmul DCT și IDCT
- *OpenCV*: pentru importarea imaginilor și diverse transformări BGR/RGB
- *Pandas*: pentru a genera tabele ordonate cu coeficienții DCT
- *Pillow*: pentru a ne ajuta în salvarea imaginilor generate într-un format gif

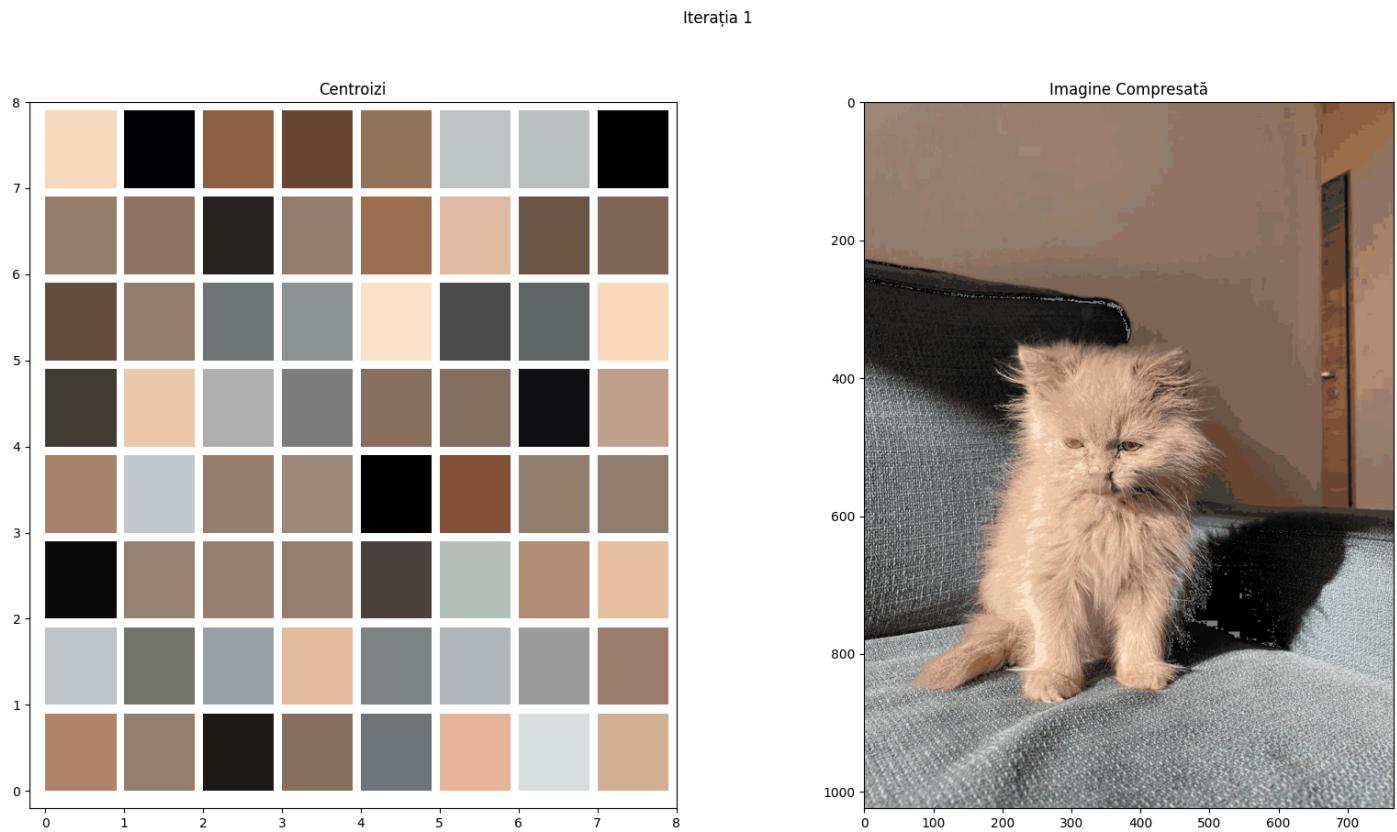
Am încărcat codul folosit în repo-urile personale ale fiecărui membru al echipei și în repo-ul principal al proiectului aflat la adresa:

<https://github.com/cristibc/SignalProcessingProject>

În repo se găsește codul sursă în fișierul Raport_final.ipynb sub forma unui Jupyter Notebook. Acolo avem sursa pentru imaginile generate în tot proiectul dar și implementarea K-Means pentru demo-urile noastre. În folderul checkpoints am salvat diverse imagini de care am avut nevoie în formularea raportului, dar și gif-uri cu parcursul algoritmului pe interacțiuni în diverse exemple. De asemenea, am încărcat și prezentarea în format Powerpoint cu numele Prezentare_initiala.pptx

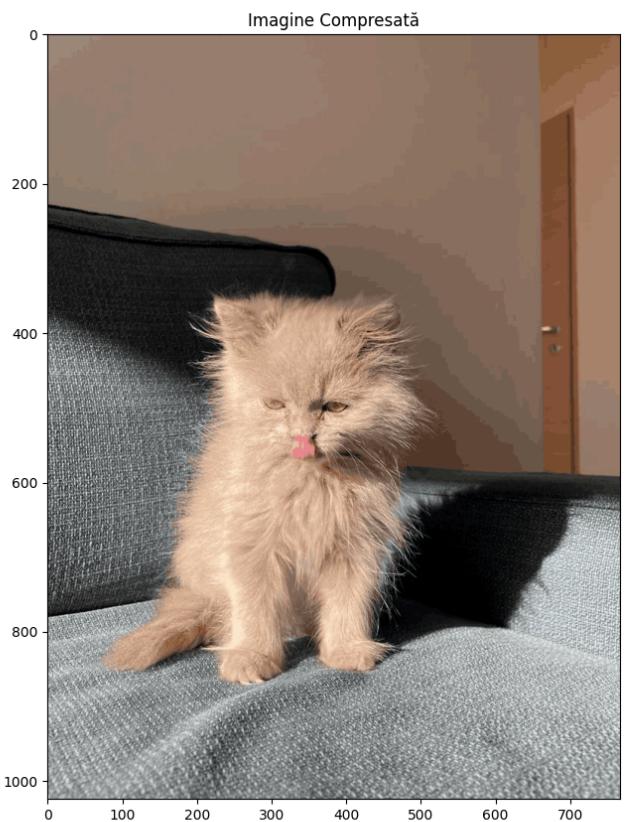
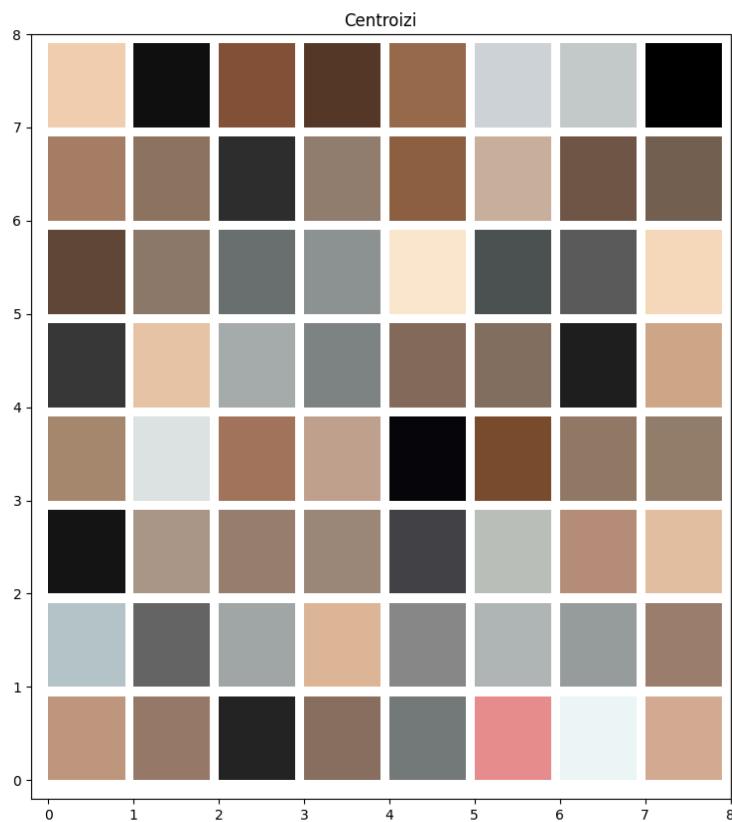
6. Rezultate

Vom exemplifica în continuare în mod vizual algoritmul de compresie menționat mai sus, anume K-Means:



În exemplul de mai sus putem observa în partea stângă centroizii ce au fost aleși pentru a reprezenta culorile din imagine. În cazul nostru, am ales 64 de clustere, adică am redus culorile imaginii la doar 64 de culori reprezentative. Aceasta este prima iterație, cea unde centroizii au fost aleși în mod aleator, motiv pentru care putem observa erori, în special în umbra și în blana pisicii, în gradientul fundalului, dar și în lipsa culorii de pe limba acesteia. Vom prezenta în următoarea imagine iterația finală, în care centroizii au ajuns la convergență, motiv pentru care culorile alese sunt considerate optime:

Iterația 73



În iterația finală putem observa cum gradientul de pe fundal este mult mai uniform, nu există "artefacte" sau erori în umbra pisicii sau în cea a canapelei, iar limba a primit o culoare specifică. Având în vedere că au fost alese doar 64 de culori, rezultatele sunt promițătoare, dar este clară legătura dintre numărul de clustere și calitatea imaginii finale. În cazul acesta, gradientul de pe fundal încă este fuzzy și observăm cu ochiul liber trecerea între culorile acestuia. Un număr de clustere standard este 256, ce ne oferă un raport balansat între calitate și compresie.

Timpul de compresie crește direct proporțional cu numărul de clustere ales. Pentru configurația noastră locală, în medie, am avut un timp de comprimare completă de 2-5 minute pentru 256 de clustere, cu 4-5 secunde per iterare, iar pentru 64 de clustere am obținut o comprimare completă în 1-3 minute, cu 1-2 secunde per iterare. Procentul de reducere a dimensiunii fișierului a fost de 40-50% între run-uri.

Imaginea necompresată



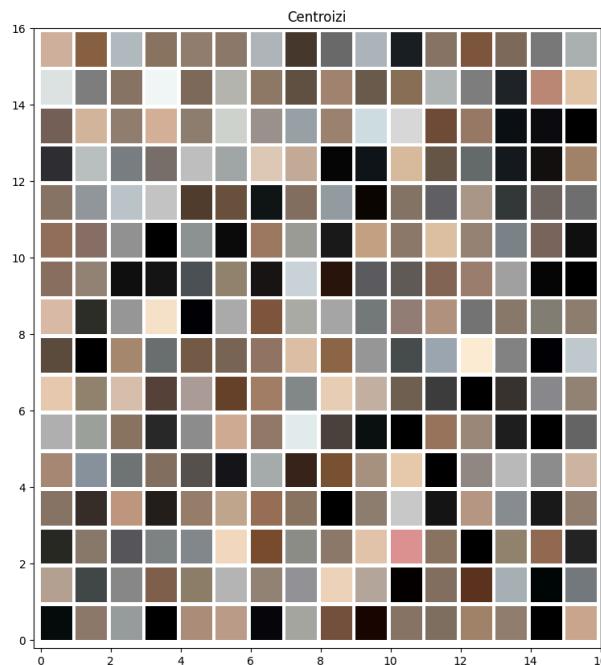
317kb

K=256



177kb (44% reduction)

În exemplul de mai sus putem observa cum un număr de clustere optim poate influența calitatea imaginii în mod pozitiv. Gradientul de pe fundal este mult mai subtil și chiar dacă am pierdut câteva nuanțe din blana pisicii, dimensiunea imaginii a fost redusă cu 44%, un procent semnificativ. Centroizii aleși au fost următorii:



Pentru a vizualiza mai bine diferențele dintre imaginile compresate cu valori diferite pentru numărul clusterelor, am generat și o imagine comparativă, cu 16, 32, 64 și 128 de clustere. Observăm cum culoarea limbii pisicii lipsește din toate, motivul fiind randomness-ul alegerii centroizilor (într-un caz ilustrat precedent, culoarea roz a apărut încă de la valoarea 64). Din acest motiv, atunci când alegem acest număr, trebuie să avem în vedere o marjă de eroare pentru detalii mici din poză.

k=16



k=32



k=64



k=128



Variațiile dintre run-uri, cauzate de alegerea în mod aleator a centroizilor în prima iterație pot fi semnificative. În unele cazuri putem ajunge la convergență mult mai repede, dar acest factor nu este corelat cu calitatea finală. În codul nostru, am ales o marjă de eroare pentru distanța euclidiană de 0.01 (10^{-2}) în confirmarea convergenței, pentru a menține un număr de iterări optim. În imaginile de mai jos, diferențele cele mai ușor de observat sunt în umbrele de pe perete.

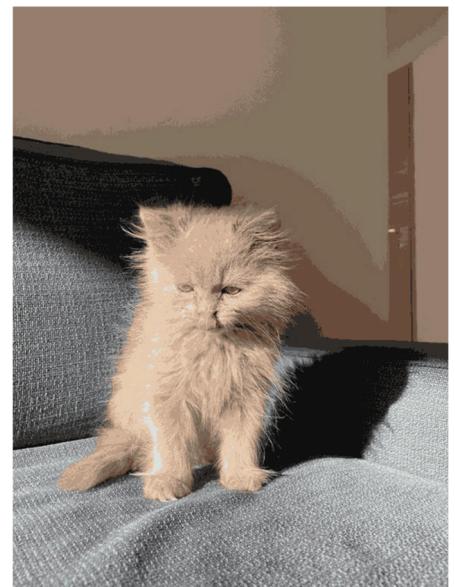
k=16, run 1, iter 47



k=16, run 2, iter 20



k=16, run 3, iter 17



Diferențele dintre run-uri devin mai subtile pe măsură ce creștem numărul clusterelor folosite:

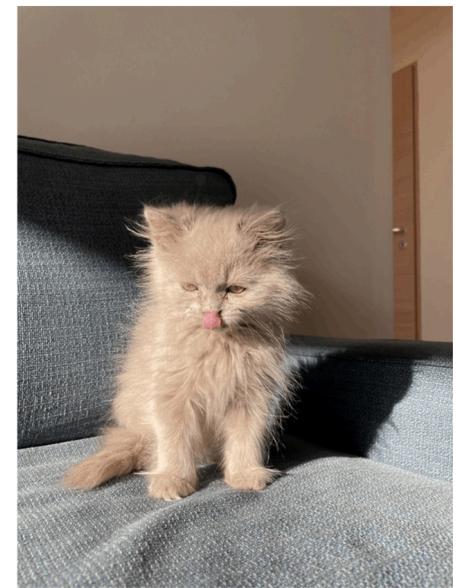
k=256, run 1, iter 100



k=256, run 2, iter 14



k=256, run 3, iter 55



7. Concluzii

Prin Image compression ajungem la o situație win-win: Analog metodelor de compresie audio precum mp3 (unde frecvențele pe care nu le putem percepe sunt eliminate), impactul negativ asupra utilizatorului este minim, pe când beneficiile aduse din punct de vedere tehnologic, al stocării datelor și transferului/încărcării mai rapid al imaginilor este unul considerabil.

Putem trage câteva concluzii despre implementarea algoritmilor de compresie moderni, în special cei folosind Machine Learning precum K-Means, având în vedere rezultatele obținute de noi.

În primul rând, trebuie să precizăm faptul că este nevoie în continuare de research în găsirea unor algoritmi optimi, dar și în implementarea acestora în diferite combinații, pentru a obține rezultatele dorite.

Algoritmul K-Means, central raportului nostru, este un exemplu de algoritm bazat pe Machine Learning ușor de înțeles și implementat. Simplicitatea lui contribuie și la eficiență, măsurată ca timp de compresie a imaginilor. Unul dintre avantajele acestuia este că funcționează foarte bine atunci când avem multe culori distințe, cu limite clare între obiectele de diferite culori, deoarece le poate separa eficient. Un dezavantaj major este că nu realizează o compresie optimă din punct de vedere vizual în momentul în care în imagine există gradiene subtile de culori asemănătoare, deoarece în aceste cazuri un număr semnificativ de centroizi vor fi alocați unor schimbări mici de culoare, în defavoarea centroizilor ce ar putea fi asignați culorilor unor obiecte mici (detalii colorate minuscule).

În consecință, trebuie să apreciem impactul pe care acești algoritmi îl au asupra vieții noastre de zi cu zi, deoarece, chiar dacă nu realizăm acest lucru, internetul, dispozitivele mobile și orice alt device din epoca modernă se folosesc de aceștia pentru a ne face viața mai ușoară și mai rapidă din punct de vedere al transmiterii datelor.

8. Bibliografie

[The Discrete Cosine Transform in Action](#)

[What Is Bandwidth in Networking? - IT Glossary | SolarWinds](#)

[What's Image Compression and How it Works](#)

[What is a RAW image file and how much memory does it require? | Sony USA](#)

[High-Efficiency Video Coding \(HEVC\) Explained - Castr](#)

[The Unreasonable Effectiveness of JPEG: A Signal Processing Approach](#)

[Clear, Visual Explanation of K-Means for Image Compression](#)

[HEVC / H.265 Video Compression](#)

[Image Compression using K-Means Clustering | by Satyam Kumar | Towards Data Science](#)

[Image Compression with K-means Clustering | by Jagajith | CodeX | Medium](#)

[Compression: Crash Course Computer Science #21](#)