UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

DOMENIUL CALCULATOARE ȘI TEHNOLOGIA INFORMAȚIEI

DISCIPLINA PROCESAREA SEMNALELOR (ANUL IV)

ALGORITMI MODERNI PENTRU COMPRESIA IMAGINILOR

- WEBP -

FLORIAN LUCA-PAUL

IANCU FLORENTINA-MIHAELA

ȚUGUI IUSTIN

GRUPA 461

2024

BUCUREȘTI

Cuprins

1. Introducere

Compresia unei imagini se referă la aplicarea unui algoritm peste datele sale (pixelii), cu scopul de a îi reduce dimensiunea acesteia pe disc și de a scădea costul transmiterii sale pe diverse canale de comunicare. Metodele utilizate pentru comprimarea fișierelor de imagine se încadrează, de regulă, în două categorii: cu pierderi de informație (*lossy*) și fără pierderi de informație (*lossless*).

În cazul particular al compresiei *lossy* , dimensiunea unui fișier este redusă prin eliminarea permanentă a informațiilor mai puțin critice, în special a datelor redundante. Consecința acestui proces este diminuarea calității imaginii până la un punct de distorsiune ușor observabilă, ce se accentuează direct proporțional cu agresivitatea compresiei. Menținerea calității este, în acest sens, determinată de alegerea parametrilor optimi ai algoritmului utilizat.

2. Algoritmi de compresie des întâlniți

2.1. Transform Coding

Transform coding este un algoritm din categoria *lossy* care utilizează transformata cosinus discretă (DCT), transformare matematică aplicată frecvent în compresia imaginilor.

Similară cu aceasta, o altă metodă de transformare utilizată în compresia imaginii este transformarea *wavelet*. Aceasta funcționează prin împărțirea datelor imaginii într-un set de coeficienți la diferite nivele de rezoluție (*wavelets*), pentru ca apoi să se păstreze doar cei mai semnificativi dintre aceștia. Rezultatul final este o reprezentare a imaginii care este mai compactă și mai ușor de comprimat.

2.2. Block Truncation Coding

De asemenea un algoritm *lossy*, acesta împarte imaginea în blocuri de pixeli, iar valoarea medie a pixelilor unui bloc este folosită ca prag. Toți pixelii care depășesc valoarea pragului primesc, în continuare, valoarea de 1 sau 0.[1] Cu cât blocurile sunt mai mari, cu atât crește proporția compresiei, dar în același timp scade și calitatea imaginii.

2.3. Arithmetic Coding

Fiind un algoritm *lossless*, acesta codifică caracterele utilizate frecvent în fișierul imaginii cu mai puțini biți și caracterele utilizate rar cu mai mulți, obținându-se astfel un total mai mic de biți.[2]

Unul dintre avantajele cheie ale codificării aritmetice este capacitatea sa de a produce coduri foarte mici, chiar și pentru imagini foarte complexe. Acest lucru se datorează faptului că ia în considerare corelațiile dintre diferitele simboluri, permițându-i să codifice datele imaginii într-o formă mai compactă.

2.4. Run Length Encoding

Această metodă este *lossless* și folosește codificarea secvențelor de pixeli care se repetă consecutiv.[3] Tipul acesta de compresie este în special util pentru imaginile care conțin zone mari de culori solide sau modele repetate, cum ar fi graficele sau ilustrațiile.

2.5. Deflate

Metoda Deflate este un algoritm popular de compresie a imaginii fără pierderi care utilizează o combinație a algoritmului de compresie LZ77 și codarea Huffman. Această metodă este utilizată des în compresia de imagini și este metoda implicită de compresie folosită de formatul PNG.[4]

Algoritmul LZ77 funcționează prin identificarea și înlocuirea modelelor repetate din datele imaginii cu referință la o singură instanță a modelului. Acest proces reduce efectiv dimensiunea datelor prin înlocuirea modelelor repetate cu o referință mai mică.

Codarea Huffman este utilizată în metoda Deflate pentru a reduce și mai mult dimensiunea datelor de imagine prin alocarea de coduri scurte modelelor care apar frecvent și coduri mai lungi modelelor care apar mai puțin frecvent.

3. Formate de imagini uzuale

3.1. JPEG

Cel mai comun exemplu de compresie cu pierderi este JPEG (*Joint Photographic Experts Group*, organizația care a standardizat formatul în anii 1980[5]), un format utilizat pe scară largă pe internet și în fotografia digitală, susținut de numeroase instrumente și aplicații. O imagine JPEG

poate conține până la 16,8 milioane de culori[5], păstrând o dimensiune redusă, datorită transformărilor matematice și a cuantizării aplicate peste datele acesteia.

3.2. GIF

GIF (*Graphics Interchange Format*) comprimă imaginile prin reducerea numărului de culori disponibile. Acest tip de compresie se bazează pe parcurgerea imaginii rând cu rând și detectarea pixelilor similari de pe același rând (nu și de pe rânduri diferite)[6]. Cea mai bună utilizare pentru formatul GIF este în cazul imaginilor animate. Din păcate, aplicarea compresiei GIF în multe situații se poate dovedi a fi ineficientă, deoarece aceasta este realizată unidimensional (rând cu rând), fiind incompatibilă raportat la imaginile bidimensionale.

3.3. PNG

Unul dintre cele mai des întâlnite formate *lossless* este PNG (*Portable Network Graphics*), format utilizat pe scară largă ce reduce dimensiunea fișierului prin identificarea modelelor repetitive și comprimând aceste modele împreună. Deși fișierele PNG sunt în general mai mari decât fișierele JPEG, site-urile web le folosesc deseori atunci când sunt necesare mai multe detalii ale imaginii, spre exemplu în cazul logo-urilor, pictogramelor, capturilor de ecran sau imaginilor cu text.[7]

4. WebP

WebP este un format de imagini dezvoltat de Google în 2013 ce oferă atât compresie *lossy*, cât și *lossless*, la dimensiuni semnificativ mai mici decât alte variante similare.

Acest format își are originile în codec-ul video VP8 (sau WebM), dezvoltat în 2008 de On2 Technologies (acum subsidiar Google). La bază, o imagine WebP este un singur cadru comprimat dintr-un videoclip WebM, asupra căruia au fost aplicate o serie de procese complexe de prelucrare.[8]

Asemenea JPEG, în procesul de compresie spațiul de culori al imaginii este convertit într-unul ce separă componenta de luminanță de cele de culoare. Pentru WebP, acest spațiu este Y’UV.

4.1. WebP *lossy*

Versiunea *lossy* a formatului WebP a fost concepută drept o îmbunătățire față de JPEG, cele două fiind similare în mai multe privințe ale implementării.

Într-o primă fază, imaginea (după aplicare conversiei la Y'UV) este împărțită în „macroblocuri", seturi compuse dintr-un bloc de 16x16 pixeli de luminanță (componenta Y) și două blocuri de 8x8 pixeli de crominanță (componentele U și V).

Fiecare dintre cele trei tipuri de blocuri sunt împărțite în continuare în segmente de 4x4 pixeli, urmând ca asupra celor 16 componente de luminanță să se aplice un proces de predicție a pixelilor. Acest pas, specific implementării *lossy* a formatului, implică, pentru început, extragerea, pentru fiecare segment în parte, a următoarelor componente:

- Linia de 4 pixeli de deasupra segmentului;
- Coloana de 4 pixeli din stânga segmentului;
- Pixelul din stânga sus relativ la poziția segmentului în bloc.

Aceste informații sunt transmise unor „predictori" ce generează o serie de noi segmente de 4x4 pixeli pe baza lor. Segmentul estimat ca fiind cel mai apropiat de cel original (i.e. eroarea dintre cele două este minimă) este ales pentru a fi pus în blocul final.

Modurile de predicție utilizate, specificate de standardul VP8[9], sunt următoarele:

- Predicție orizontală (*H_PRED*) – Fiecare coloană a segmentului este înlocuită cu cea colectată din stânga sa;
- Predicție verticală (*V_PRED*) – Fiecare linie a segmentului este înlocuită cu cea colectată de deasupra sa;
- Predicție DC (*DC_PRED*) – Umple tot segmentul cu o singură valoare aferentă mediei dintre pixelii din linia de deasupra segmentului și cei din coloana din stânga sa;
- Predicție *TrueMotion* (*TM_PRED*) – Folosește toate informațiile disponibile, atât linia și coloana, cât și pixelul selectat din stânga sus, pentru a aplica formula:

$$X_{ij} = L_i + A_j - C$$

pentru toți pixelii segmentului. X_{ij} reprezintă pixelul de pe linia *i* și coloana *j* din segment.

După prelucrarea fiecărui segment și recompunerea blocurilor din care fac parte, acestea din urmă sunt trecute prin procesul de compresie identic formatului JPEG: peste datele lor se aplică transformata cosinus discretă (DCT), rezultatul transformării este cuantizat cu o matrice dată de standardul JPEG, urmând ca pixelii să fie „restaurați" prin aplicare inversei operației menționate.

Pentru scrierea pe disc a imaginii comprimate, nu este nevoie de toate informațiile din fiecare segment peste care s-a aplicat predicția. În schimb, se rețin doar informațiile reziduale - diferența dintre segmentul original și cea mai bună predicție.

Peste aceste segmente reziduale se aplică DCT, urmând ca apoi să fie cuantizate, reordonate și transmise unui comprimator algoritmic ce le codifică în conformitate cu entropia Shannon, într-un mod mai stabil decât codarea Huffman.[10]

Pentru o calitate îmbunătățită a rezultatului compresiei, WebP aplică cuantizare adaptivă pentru fiecare bloc în funcție de conținutul său. Astfel, părțile mai dificil de comprimat pot avea parametrii algoritmului ajustați în mod independent față de celelalte. VP8, pe care este bazat WebP, permite până la maxim patru fragmente de genul, din cauza unei limitări tehnice.[10]

4.1.1. WebP lossy cu alpha

WebP *lossy* oferă o variantă a codificării imagini ce păstrează canalul *alpha* al transparenței intact, într-o manieră *lossless*. Spre deosebire de comprimarea PNG, alternativa actuală pentru păstrarea transparenței, WebP *alpha* adaugă doar 22% octeți peste comprimarea *lossy*, oferind o economie de spațiu pe disc de aproximativ 60-70%.[10]

4.2. WebP *lossless*

Spre deosebire de varianta *lossy*, compresia WebP *lossless* implică aplicare mai multor tehnici pentru obținerea rezultatului final.

Printre cele mai importante tehnici se numără:

- Transformarea spațială (predictivă) – Valoarea unui pixel este codificată în funcție de valorile pixelilor vecini deja decodificați. Există 13 modele posibile de predicție bazate

pe cele patru direcții principale și combinațiile lor, cu precădere stânga, sus, stânga sus și dreapta sus;[10]

- Decorelarea culorilor – Valorile RGB din fiecare pixel sunt decorelate. Se păstrează canalul doar canalul verde (G), urmând ca celelalte două, roșu (R) și albastru (B) să fie codificate pe baza acestuia. Similar cu predicția în WebP *lossy*, imaginea este împărțită în segmente, peste ai căror pixeli se aplică una din cele trei tipuri de transformări: verde-roșu, verde-albastru și roșu-albastru;[10]
- Indexarea culorilor – Dacă variația culorilor din imagine este limitată, se poate utiliza un vector de frecvență care codifică acele culori, valorile pixelilor fiind convertite, pe rând, în valori ale vectorului. În acest proces poate fi inclusă, pe lângă valorile RGB, și valoarea A (*alpha*) a transparenței;[10]
- *Caching*-ul culorilor – Algoritmul poate reține o paletă conținând cele mai recente 32 de culori întâlnite, adaptată dinamic pe măsură ce scanarea imaginii progresează vertical.[10]

Pentru a putea fi scrisă pe disc, la fel ca în cazul *lossy*, atât datele imaginii comprimate, cât și parametrii procesului sunt codificate. Algoritmul utilizat pentru varianta *lossless* este LZ77-Huffman.[10]

5. Implementarea WebP *lossy*

Pentru implementarea algoritmului *lossy* al WebP[a], am ales limabajul de programare Python, versiunea 3.10.7, și librăriile:

- *numpy* și *scipy*, pentru prelucrarea imaginilor și alte operații aritmetice;
- *skimage* pentru obținerea unor imagini de testare;
- *matplotlib* pentru afișarea unor statistici legate de performanța algoritmului față de JPEG.

Înainte de aplicarea efectivă a compresiei, imaginea este translatată din spațiul de culori RGB în spațiul YUV. Apoi, funcția *webp()* urmărește în detaliu pașii de implementare prezentați în documentația oficială a formatului, cu excepția părții de scriere pe disc, ce a fost omisă. Astfel, la pasul predicției segmentelor de 4x4 pixeli extrase din blocurile de dimensiune 16x16, rezultatul este extras ca atare, evitând compunerea segmentelor reziduale.

Mai mult decât atât, dat fiind că imaginea inițială este descompusă în „macroblocuri”, dimensiunile sale sunt „căptușite”, dacă este cazul, până când acestea sunt ambele divizibile cu 16. Funcția din *numpy* ce ajută la acest proces este *numpy.pad()*, iar după prelucrare imaginea este restaurată la dimensiunile inițiale.

Componentele U și V sunt subeșantionate la o rată de 4:2:0 (de asemenea indicată[9] de documentația oficială), astfel că acestea sunt reduse la o dimensiune de 8x8 înaintea prelucrării: se păstrează doar cel de-al doilea pixel de-a lungul fiecărei dimensiuni. Procesul este reversibil, prin apelarea funcției *zoom()* din librăria *scipy*, după finalul compresiei JPEG.

Deși rezultatul compresiei prezintă un grad observabil de scădere a calității imaginii, integritatea totală a acesteia este păstrată aproape intactă, iar mărimea sa pe disc este considerabil mai mică decât în cazul compresiei JPEG.

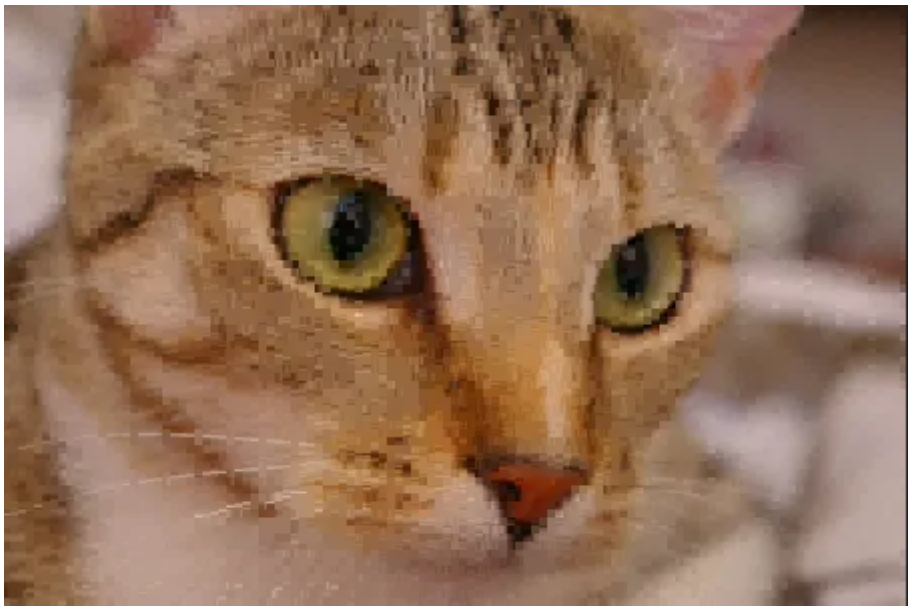

Fig. 1 - Rezultatul compresiei WebP asupra imaginii *cat* din setul *skimage.data*

Pentru imaginea *cat* din setul *skimage.data*, compresia WebP (figura 1) reduce dimensiunea fișierului final la 12 Kb, jumătate din dimensiunea compresiei JPEG.

6. Comparație între JPEG și WebP

Cea mai evidentă diferență este faptul că formatul JPEG permite doar compresia *lossy*, pe când WebP acceptă atât compresia *lossless*, cât și *lossy*. Mai mult, WebP prezintă suport și pentru animații, date fiind originile sale în WebM.

În ceea ce privește performanța compresiei, WebP depășește JPEG. Conform documentației oficiale[11], imaiginile WebP *lossy* sunt de până la 25-34% mai mici în dimensiune decât echivalentele JPEG, păstrând un index de similaritate cu imaginea originală (SSIM - *Structural Similarity Index*) echivalent. Acest lucru este observabil și în implementarea proprie, unde diferențele măsurate asupra a patru imagini comprimate atât cu WebP *lossy*, cât și cu JPEG (figura 2), sunt neglijabile.

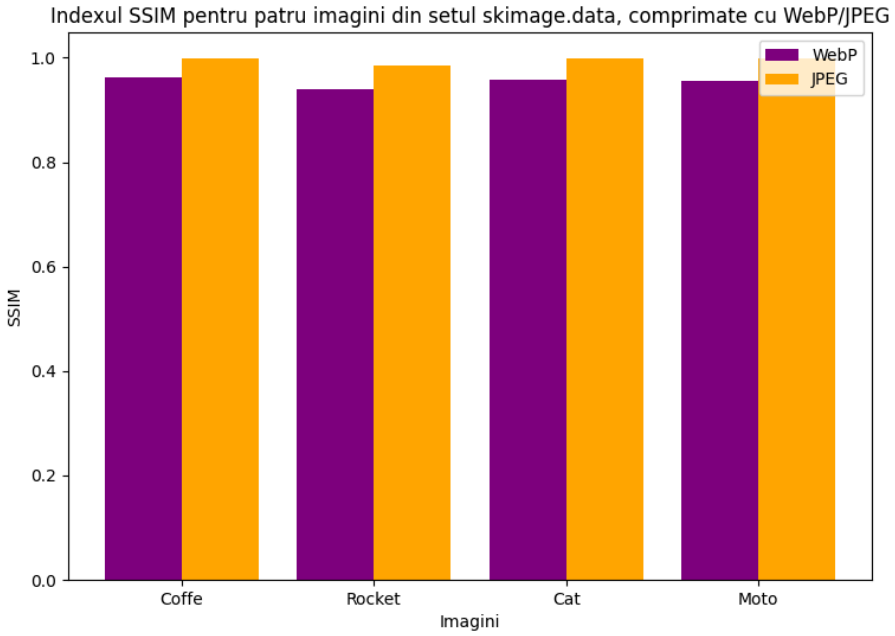

Fig. 2 - Diferențele dintre indecșii SSIM pentru patru imagini din setul *skimage.data*

În aceeași implementare, însă, din cauza faptului că nu au fost aplicate metode de atenuare a zgomotului, erorile pătrate medii (MSE - *Mean Squared Error*) sunt considerabil mai ridicate la compresia WebP *lossy* decât la cea JPEG, diferența maximă fiind aproape dublă (figura 3).

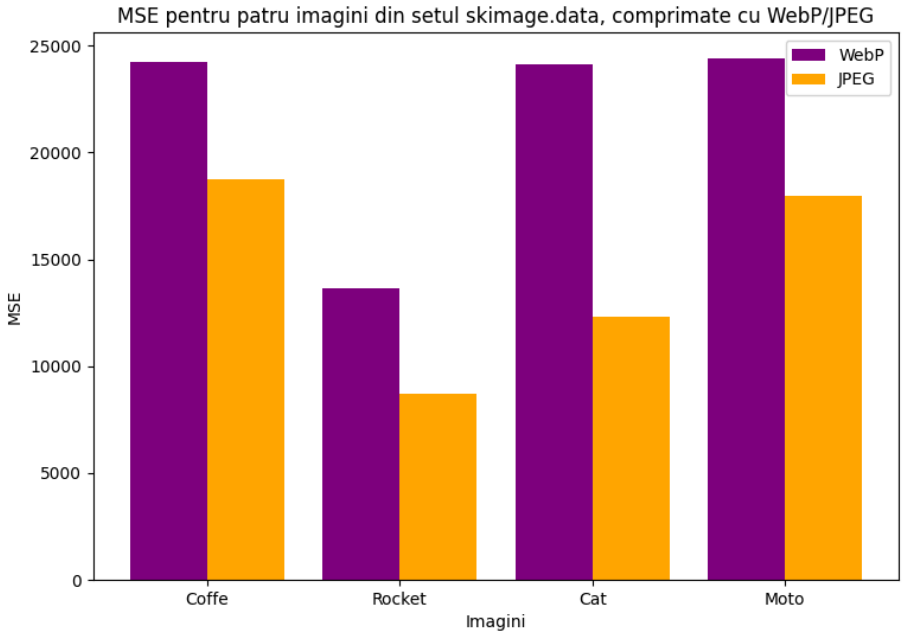

Fig. 3 - Diferențele de erori MSE pentru patru imagini din setul *skimage.data*

Același lucru se poate observa (figura 4) și la PSNR (*Peak Signal-to-Noise Ratio*), rata dintre puterea maximă a semnalului din imagine și zgomotul prezent. O valoare mai ridicată a acestei rate indică o calitate mai bună a imaginii.

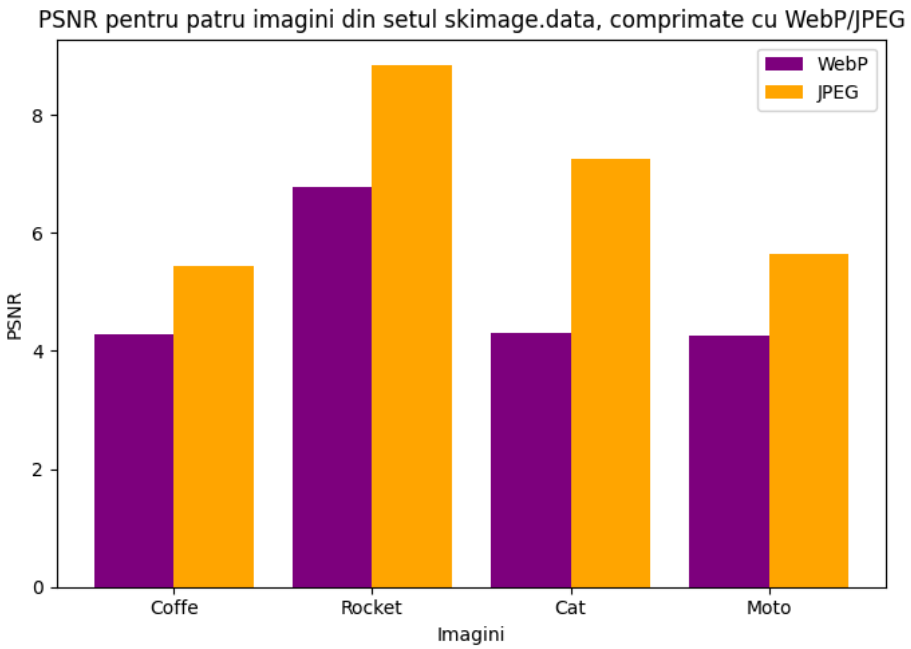

Fig. 4 - Diferențele de rate PSNR pentru patru imagini din setul *skimage.data*

Per total, procesul de predicție a segmentelor de 4x4 pixeli se poate considera ca fiind cel mai important determinant al diferenței în compresie față de formatul JPEG, în special în contextul codificării informațiilor pe disc, mai specific al informațiilor reziduale rezultate în urma acestui acestuia.

7. Concluzii

În contextul acumulării de informație digitală globală de-a lungul timpului, compresia este un domeniu ce nu trebuie neglijat. În ciuda stabilității unor formate standardizate, unele vechi de peste 30 de ani, tehnologia continuă să avanseze și să aducă îmbunătățiri într-un ritm constant. Singura problemă de constatat ar fi gradul de adoptare al acestor îmbunătățiri de către utilizatorii și dezvoltatorii din zilele noastre.

WebP, spre exemplu, construit peste bazele puse de JPEG și VP8, considerabil mai performant decât acestea, nu este la fel de larg răspândit precum JPEG, atât în domeniul browserelor web, cât și în cel al dispozitivelor de orice tip.

8. Bibliografie și resurse

1. Wikipedia. „Block Truncation Coding”. Accesat pe 25 Ianuarie, 2024. https://en.wikipedia.org/wiki/Block_Truncation_Coding
2. Wikipedia. „Arithmetic coding”. Accesat pe 25 Ianuarie, 2024. https://en.wikipedia.org/wiki/Arithmetic_coding
3. StackAbuse. „Run-Length Encoding”. Accesat pe 25 Ianuarie, 2024. https://stackabuse.com/run-length-encoding/
4. Wikipedia. „Deflate”. Accesat pe 25 Ianuarie, 2024. https://en.wikipedia.org/wiki/Deflate
5. Adobe. „Everything you need to know about JPEG”. Accesat pe 25 Ianuarie, 2024. https://www.adobe.com/creativecloud/file-types/image/raster/jpeg-file.html
6. Geeks For Geeks. „Compression of GIF images”. Accesat pe 26 Ianuarie, 2024. https://www.geeksforgeeks.org/compression-of-gif-images/
7. Adobe. „PNG files”. Accesat pe 26 Ianuarie, 2024. https://www.adobe.com/creativecloud/file-types/image/raster/png-file.html
8. McAnlis, Colt. „How WebP works (lossly mode).” *Medium*. 26 Mai, 2016. https://medium.com/@duhroach/how-webp-works-lossly-mode-33bd2b1d0670
9. Jim Bankoski, Paul Wilkins, Yaowu Xu. „Technical Overview of VP8, an Open Source Video Codec for The Web”. *IEEE*. Iulie, 2011.
10. Google. „Compression Techniques.” Accesat pe 28 Ianuarie, 2024. https://developers.google.com/speed/webp/docs/compression
11. Google. „An image format for the Web”. Access pe 28 Ianuarie, 2024. https://developers.google.com/speed/webp/

a. https://github.com/superBleep/Procesarea_Semnalelor/tree/main/Proiect_PS