

Project ECISV

Tehnici de verificare a autenticității imaginilor – ELA (Error Level Analysis)

Master: TAID anul 2

Student: Stancovici Marian

CAPITOLUL 1. INTRODUCERE SI CONTEXT GENERAL

1.1. Motivația alegerii temei

În ultimii ani, evoluția rapidă a tehnologiilor de imagistică digitală a condus la o utilizare extrem de largă a imaginilor digitale în diverse scopuri, de la presă și rețele sociale, până la probe juridice. Din păcate, această accesibilitate tehnologică a venit la pachet cu o creștere alarmantă a metodelor de manipulare și denaturare a conținutului imaginilor originale.

Trăim într-o eră în care "a vedea nu mai înseamnă a crede". Software-uri precum Adobe Photoshop sau GIMP permit oricărui utilizator să altereze realitatea surprinsă într-o fotografie, iar majoritatea acestor aplicații vor resalva imaginea modificată ca o nouă imagine JPEG, ascunzând adesea urmele vizibile ochiului liber. În acest context, expertiza criminalistică a imaginilor devine esențială pentru a stabili veridicitatea probelor vizuale.

1.2. Obiectivele proiectului

Acest proiect a luat naștere din dorința de a înțelege și combate fenomenul tot mai răspândit al falsificării imaginilor digitale, propunându-și dezvoltarea unei soluții software proprii, denumită "Forensic ELA Expert". Scopul principal a fost acela de a crea o aplicație desktop completă, care să nu se rezume doar la teorie, ci să permită aplicarea practică a metodei Error Level Analysis (ELA) pentru a depista modificările invizibile ochiului uman din fișierele JPEG. Ne-am propus ca aplicația să funcționeze ca un laborator criminalistic virtual, unde utilizatorul poate atât să analizeze imagini suspecte, cât și să genereze falsuri controlate pentru a studia comportamentul algoritmilor de compresie.

Un accent deosebit s-a pus pe implementarea unor instrumente moderne de editare, pentru a simula cât mai fidel scenariile reale de manipulare. Astfel, am integrat module de Inteligență Artificială (folosind biblioteca rembg) pentru decuparea automată și precisă a obiectelor, dar și unelte manuale de tip "Lasso" pentru o flexibilitate sporită. Obiectivul a fost să demonstrezi că, indiferent cât de bine este realizat vizual un fals prin rotire sau scalare, structura matematică a fișierului păstrează urmele intervenției, urme pe care aplicația noastră le poate evidenția prin hărți de eroare colorate.

Nu în ultimul rând, proiectul a vizat explorarea avansată a degradării JPEG prin funcționalități inovatoare, precum compresia locală (zonală) și cea succesivă. Am dorit să demonstrezi experimental cum compresia aplicată inegal pe o imagine — doar pe o față sau pe un număr de înmatriculare — creează discrepanțe majore la analiza ELA.

Pentru a valida științific rezultatele obținute în aplicația proprie, am inclus și un mecanism de verificare încrucișată cu platformele standard FotoForensics și photo-forensics,

asigurând astfel că instrumentul dezvoltat oferă rezultate corecte și relevante din punct de vedere criminalistic.

1.3. Structura lucrării

Prezenta lucrare este structurată în șapte capitole, organizate într-o succesiune logică, pornind de la fundamentele teoretice ale imagisticii digitale și ajungând la implementarea practică a soluțiilor de detectie a falsurilor.

Capitolul 1, Introducere, stabilește contextul actual al manipulării imaginilor digitale, evidențiind necesitatea dezvoltării unor instrumente automate de expertiză. De asemenea, sunt definite motivația alegerii temei și obiectivele specifice ale proiectului, atât cele de cercetare, cât și cele de dezvoltare software.

Capitolul 2, Fundamente teoretice ale falsificării imaginilor, realizează o clasificare a principalelor metode de manipulare digitală întâlnite în practică, precum tehniciile de „Splicing” (îmbinare), „Copy-Move” și retușare. Sunt prezentate diferențele dintre autentificarea activă (bazată pe semnături digitale) și cea pasivă, categorie din care face parte și metoda abordată în această lucrare.

Capitolul 3, Compresia JPEG – Elementul cheie în analiză, aprofundează mecanismul tehnic din spatele celui mai utilizat format de imagine. Se explică în detaliu procesul de divizare a imaginii în blocuri de 8x8 pixeli, transformata cosinus discretă (DCT) și, cel mai important, procesul de cuantizare, care introduce erorile specifice pe care se bazează algoritmul nostru de detectie.

Capitolul 4, Metodologia ELA (Error Level Analysis), este dedicat descrierii algoritmului utilizat. Sunt prezentate principiile matematice de calcul al ratei de eroare la recomprimare și modul în care aceste erori pot fi vizualizate sub forma unei hărți termice. Capitolul explică ipoteza fundamentală conform căreia zonele modificate dintr-o imagine prezintă niveluri de compresie inconsistente față de restul imaginii.

Capitolul 5, Studii de caz și analiza experimentală, reprezintă componenta de cercetare a proiectului. Aici sunt prezentate rezultatele obținute în urma testării algoritmului pe diverse scenarii de falsificare: inserția simplă de obiecte, manipularea cu compresie locală (zonală) și efectele compresiei succesive asupra detectabilității urmelor. Fiecare studiu de caz este însoțit de imagini comparative și o interpretare a rezultatelor vizuale.

Capitolul 6, Contribuția personală: Aplicație software pentru expertiză ELA, detaliază arhitectura și funcționalitățile aplicației "Forensic ELA Expert", dezvoltată în cadrul acestui proiect. Sunt descrise modulele implementate, inclusiv integrarea Inteligenței Artificiale pentru

decuparea automată a elementelor, uneltele de selecție manuală, simulatorul de compresie și interfața grafică modernă destinată utilizatorului final.

Lucrarea se încheie cu **Capitolul 7, Concluzii**, care sintetizează rezultatele obținute, discută eficiența metodei ELA ca instrument preliminar de triere în expertiza criminalistică și propune direcții viitoare de dezvoltare a aplicației.

CAPITOLUL 2. FUNDAMENTE TEORETICE ALE FALSIFICĂRII IMAGINILOR

Odată cu răspândirea camerelor digitale și a dispozitivelor mobile performante, imaginile au devenit principalul mijloc de comunicare și documentare a realității. Totuși, natura digitală a acestora le face extrem de vulnerabile la alterări. Falsificarea imaginilor nu mai este apanajul experților, software-uri avansate de editare fiind accesibile oricui.

În literatura de specialitate, manipularea imaginilor digitale este clasificată în funcție de procesul tehnic prin care a fost alterat conținutul. Această clasificare este esențială pentru expertiza criminalistică, deoarece fiecare tip de falsificare lasă urme specifice (artefacte) care pot fi detectate prin algoritmi dedicați, precum ELA.

2.1. Tipuri de manipulare a imaginilor digitale

Modificarea imaginilor poate fi clasificată, în principal, în trei categorii distincte:

2.1.1. Falsificarea prin îmbinare (Image Splicing, figura 1)

Aceasta implică combinarea a două sau mai multe imagini diferite pentru a crea un conținut nou. Tehnic, o regiune dintr-o imagine "sursă" este decupată și inserată într-o imagine "destinație" (exemplu: plasarea unei persoane într-o scenă în care nu a fost prezentă).

Detectabilitate: Deoarece cele două imagini provin din camere diferite sau au istorice de compresie diferite, algoritmul ELA evidențiază discrepanța dintre rata de eroare a obiectului inserat și cea a fundalului..

Un exemplu potrivit pentru aceasta categorie se poate observa în figura 1, unde am introdus pisica din poza 2 în poza 1, introducerea fiind evidentă din cauza diferențelor de compozitie, umbra, luminozitate din cele 2 poze alese.



Figura 1. Exemplu de falsificare prin imbinare

2.1.2. Falsificarea prin Copiere-Mutare (Copy-Move Forgery)

Spre deosebire de îmbinare, acest proces presupune duplicarea unei regiuni și mutarea ei în cadrul aceleiași imagini. Scopul este dublu: fie de a **ascunde** un element nedorit (acoperindu-l cu o textură de fundal), fie de a **multiplica** elemente (ex: exagerarea numărului de persoane dintr-o mulțime), așa cum vedem în figura 2.

Detectabilitate: Deși păstrează paleta de culori originală, transformările aplicate pentru integrare distrug alinierarea grilei JPEG de 8×8 pixeli, creând anomalii vizibile la analiză.

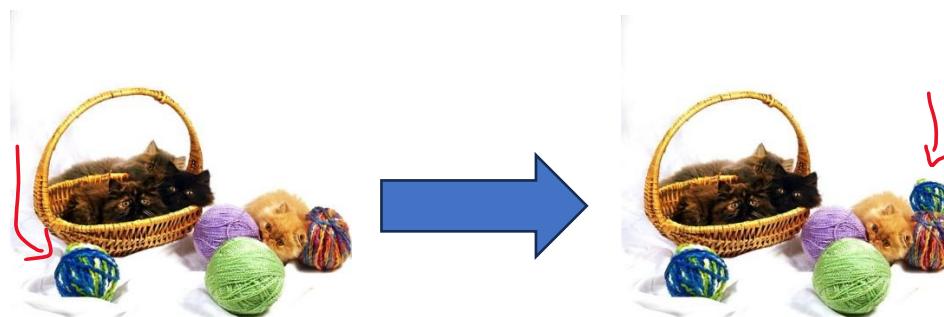


Figura 2. Exemplu Copy-Move: Multiplicarea unui element în același cadru

2.1.3. Retușarea imaginii (Retouching): Îmbunătățirea aspectului fără a modifica sensul.

Această metodă nu introduce elemente noi, ci alterează caracteristicile pixelilor existenți (netezire, modificare contrast, stergere detaliu). Deși frecventă în estetică, devine o problemă criminalistică atunci când ascunde trăsături esențiale (ex: numere de înmatriculare, cicatrici). Urmele sunt detectabile prin zone de compresie abnormal de uniforme comparativ cu zgomotul natural al imaginii. Figura 3 prezintă o retusare concreta a imaginii sursă cu ajutorul modificării luminozității și al contrastului pentru a da impresia că poza este făcută mai pe zi.



Figura 3. Exemplu de falsificare prin retusare (retouching)

2.2. Expertiza criminalistică a imaginilor

Expertiza criminalistică a imaginilor digitale este o ramură a științelor forensice care se ocupă cu investigarea istoriei unui fișier digital. Scopul principal nu este doar de a spune dacă o imagine este "falsă", ci de a răspunde la două întrebări fundamentale:

1. **Identificarea sursei:** Cu ce dispozitiv a fost capturată imaginea? (analiza senzorului, a lentilelor).
2. **Verificarea integrității:** A suferit imaginea modificări după momentul capturii?

În funcție de informațiile disponibile despre imaginea originală, tehniciile de verificare se clasifică în două mari categorii enumerate mai jos și evidențiate în figura 4:

2.2.1. Autentificare Activă: Watermarking, semnături digitale.

Această metodă necesită măsuri de precauție luate chiar în momentul generării imaginii. Include tehnici precum:

- **Watermarking digital:** Inserarea unei "semnături" invizibile în pixeli, care se distrug dacă imaginea este modificată.
- **Semnături digitale (Hash):** Generarea unui cod unic bazat pe conținutul imaginii.

- **Limitare:** Deși foarte sigure, aceste metode sunt rar întâlnite în imaginile obișnuite de pe internet, deoarece necesită camere specializate sau software dedicat instalat în momentul fotografierii.

2.2.2. Autentificare Pasivă: Detecția la nivel de pixeli, format, cameră, geometrie.

Aceasta este categoria din care face parte și **metoda ELA (Error Level Analysis)**, utilizată în acest proiect. Tehnicile pasive nu necesită nicio informație prealabilă despre imaginea originală sau despre dispozitivul care a creat-o.

Ele se bazează pe ipoteza că, deși o imagine falsificată poate părea perfectă ochiului uman, procesul de editare introduce inevitabil anomalii statistice la nivel de biți. Expertiza pasivă caută aceste "cicatrici digitale":

- Inconsistențe de iluminare sau umbre.
- Discontinuități în zgomotul senzorului (ISO noise).
- **Artefacte de compresie:** Analiza modului în care algoritmul JPEG a procesat diferite zone ale imaginii.

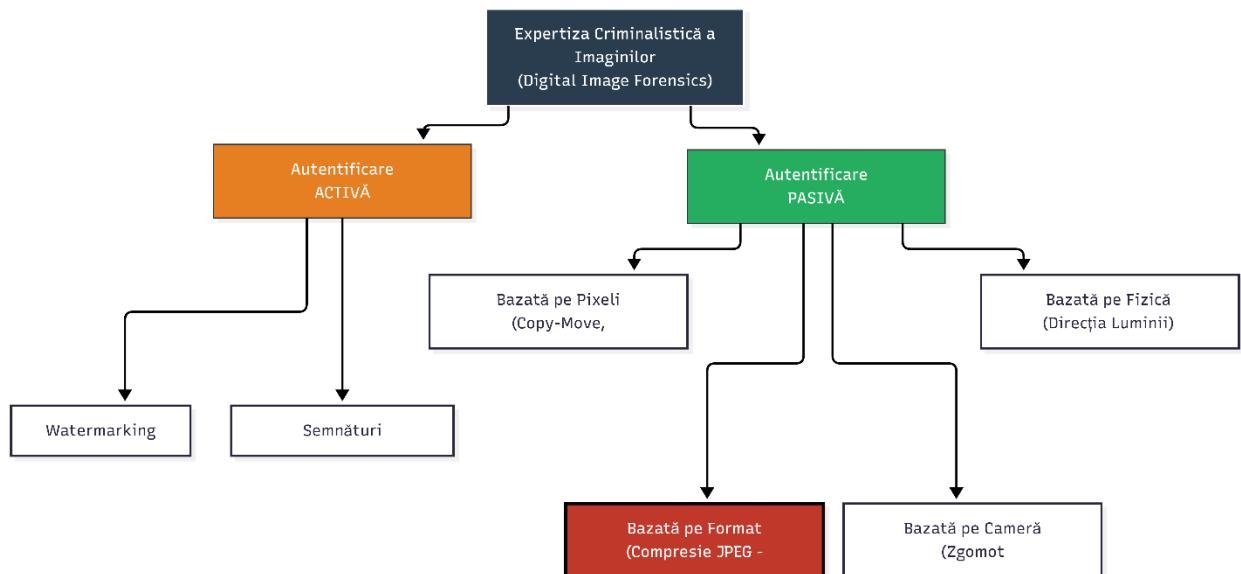


Figura 4. Autentificarea activă și pasivă

CAPITOLUL 3. COMPRESIA JPEG – ELEMENTUL CHEIE ÎN ANALIZĂ

Formatul JPEG (Joint Photographic Experts Group) este standardul dominant pentru stocarea imaginilor digitale, datorită capacitatea sa de a reduce dimensiunea fișierelor prin eliminarea informațiilor redundante sau imperceptibile ochiului uman. În contextul expertizei criminalistice, înțelegerea mecanismului de compresie JPEG este fundamentală, deoarece algoritmul ELA (Error Level Analysis) nu analizează conținutul semantic al imaginii (ce reprezintă ea), ci artefactele introduse de acest proces de compresie.

3.1. Principiul compresiei JPEG (figura 5 & 6)

Procesul de compresie JPEG este unul de tip "lossy" (cu pierderi), ceea ce înseamnă că imaginea reconstruită nu este identică, bit cu bit, cu originalul. Algoritmul începe prin convertirea spațiului de culoare din RGB (Roșu-Verde-Albastru) în YCbCr. Această transformare separă informația de luminozitate (Luminanță - Y) de informația de culoare (Cromianță - Cb și Cr). Deoarece ochiul uman este mult mai sensibil la variațiile de luminozitate decât la cele de culoare, algoritmul aplică o sub-eșantionare (subsampling) asupra canalelor de culoare, reducând rezoluția acestora fără a afecta semnificativ calitatea percepță.

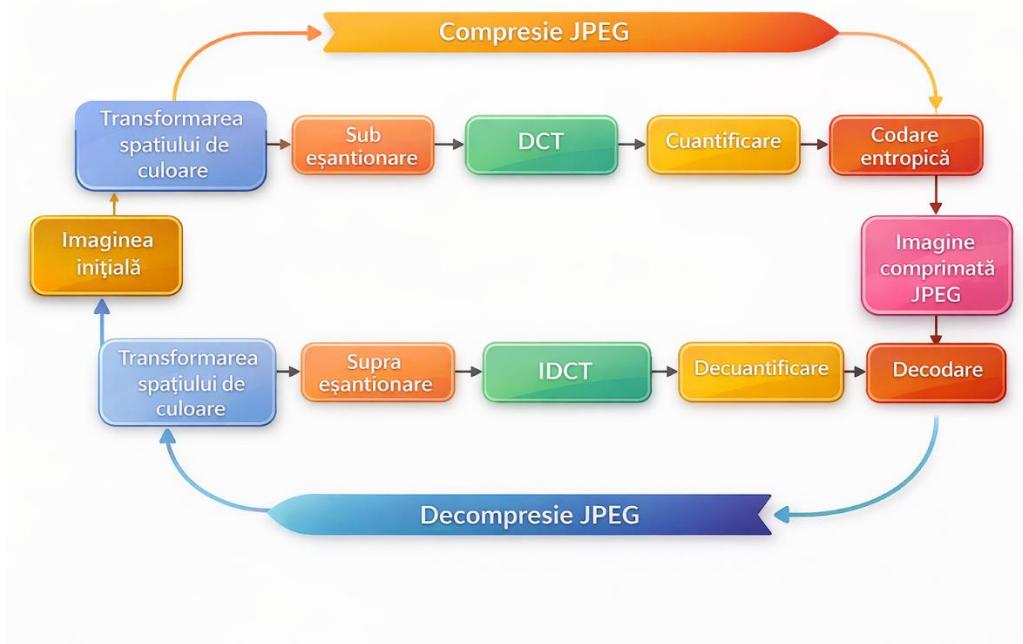


Figura 5. Compresia JPEG

Etapa critică pentru analiza ELA este divizarea imaginii. Întreaga imagine este împărțită într-o grilă de blocuri independente de **8x8 pixeli**. Fiecare bloc este procesat individual. Această structură de grilă este motivul pentru care, la o mărire excesivă, imaginile JPEG par "pixelate"

sau alcătuite din pătrate mici, și totodată motivul pentru care analiza ELA scoate în evidență marginile acestor blocuri în zonele manipulate.

Pentru fiecare bloc de 8x8 pixeli, valorile spațiale (intensitatea pixelilor) sunt convertite în domeniul frecvență utilizând **Transformata Cosinus Discretă (DCT)**. DCT transformă cei 64 de pixeli ai blocului într-o sumă de unde cosinusoidale de diferite frecvențe. Rezultatul este o matrice de 64 de coeficienți DCT: colțul din stânga-sus reprezintă "componenta continuă" (DC - media culorii blocului), iar restul sunt " componente alternative" (AC - detaliile fine și variațiile rapide).

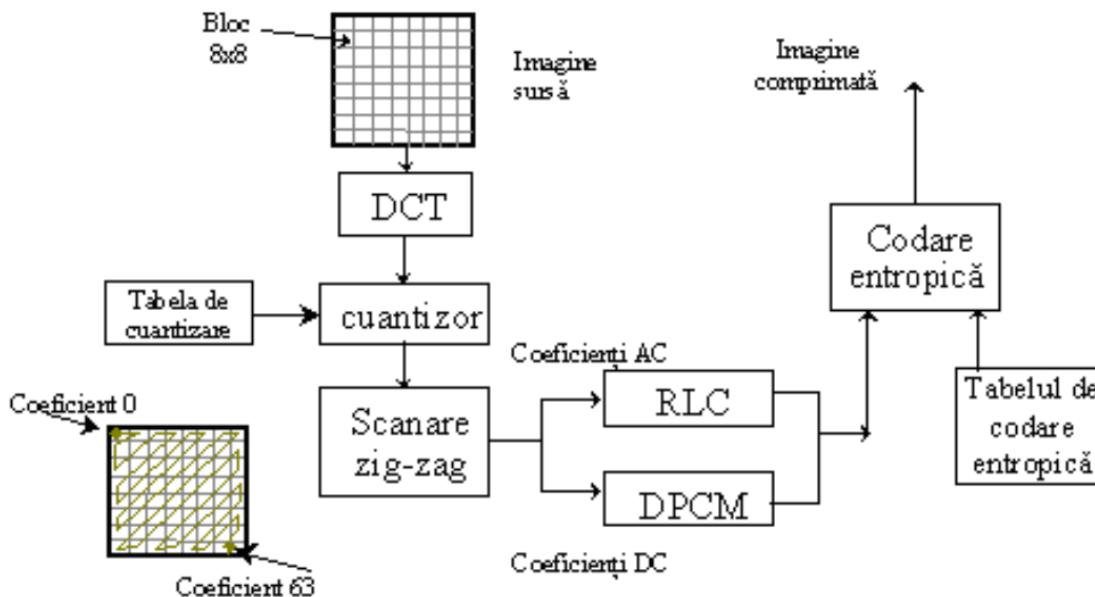


Figura 6. Schema detaliată pentru compresia JPEG incluzând divizia de blocuri 8x8

3.2. Procesul de Cuantizare

Cuantizarea este etapa în care are loc pierderea efectivă a informației și, implicit, momentul în care se introduc erorile pe care le detectăm ulterior. Coeficienții DCT obținuți anterior sunt valori reale (cu zecimale), adesea complexe. Pentru a comprima imaginea, acești coeficienți sunt împărțiți la valorile dintr-o **Matrice de Cuantizare** predefinită, iar rezultatul este rotunjite la cel mai apropiat număr întreg.

Matricea de cuantizare este cheia nivelului de calitate. Dacă utilizatorul alege o calitate ridicată (ex: 95%), matricea conține valori mici (împărțitorii sunt mici), deci rezultatele rotunjite sunt precise, păstrând detaliile. Dacă utilizatorul alege o calitate scăzută (ex: 50%), matricea conține valori mari, ceea ce face ca mulți coeficienți de frecvență înaltă (detaliile fine) să devină zero după rotunjire.

Din perspectivă criminalistică, acest proces este esențial: fiecare software de editare (Photoshop, GIMP, camere foto) folosește tabele de cuantizare ușor diferite. Atunci când o imagine este modificată, noile elemente introduse nu au trecut prin același proces de rotunjire matematică precum restul imaginii, creând o discrepanță numerică invizibilă ochiului, dar evidentă matematic.

Figura 7. Tabelele de cuantizare din algoritmul de aproximare a calitatii JPEG

3.3. Ciclul Compresie-Decompresie

Un aspect vital în analiza compresiei este comportamentul la re-salvare. Atunci când o imagine JPEG este deschisă, modificată și salvată din nou, algoritmul de compresie rulează o a doua oară peste blocurile de 8x8 pixeli.

Teoria din spatele ELA susține că nivelul de eroare (diferența dintre pixelii originali și cei comprimați) scade cu fiecare re-salvare succesivă, până la un punct de stabilizare. O imagine originală a camerei, salvată o singură dată, va avea un nivel de eroare relativ ridicat și uniform pe toată suprafața, deoarece coeficienții DCT sunt încă "proaspeti".

În schimb, dacă într-o imagine veche (deja comprimată și stabilizată) introducem un element nou (necomprimat), acel element va reacționa violent la o nouă compresie, generând o eroare mare, în timp ce fundalul va genera o eroare minimă. Această diferență de comportament la compresie între zonele stabilizate (fundal) și zonele nestabilizate (falsul) constituie baza științifică a proiectului nostru.

CAPITOLUL 4. METODOLOGIA ELA (ERROR LEVEL ANALYSIS)

Analiza Nivelului de Eroare (ELA - Error Level Analysis) este o metodă de expertiză criminalistică pasivă, dezvoltată inițial de cercetătorul Neal Krawetz. Spre deosebire de analiza vizuală clasica, care se bazează pe percepția umană asupra luminii și umbrelor, ELA este o metodă algoritmică ce detectează manipularea digitală prin evidențierea discrepanțelor în nivelul de compresie JPEG al unei imagini.

4.1. Definirea algoritmului ELA

Premisa fundamentală a ELA este că o imagine digitală originală, provenită direct de la senzorul camerei, are un nivel de compresie uniform pe toată suprafața sa. Atunci când imaginea este modificată (printr-un software de editare precum Adobe Photoshop sau GIMP) și salvată din nou, apare o desincronizare a nivelelor de eroare.

Algoritmul ELA implementat în cadrul aplicației "Forensic ELA Expert" urmează o procedură standardizată în trei pași:

1. **Re-comprimarea intenționată:** Aplicația preia imaginea suspectă (Originalul) și creează o copie temporară în memorie, pe care o re-salvează la o calitate cunoscută și fixă (de regulă 95%).
2. **Calculul diferenței (Extraction):** Se compară matematic imaginea Originală cu imaginea Re-comprimată, pixel cu pixel. Formula de bază este: $Error(x,y) = |Pixel_Original(x,y) - Pixel_Recomprimat(x,y)|$. Rezultatul este o "hartă a erorilor", care reprezintă cantitatea de informație pierdută prin noua compresie.
3. **Amplificarea vizuală:** Deoarece diferențele brute sunt adesea invizibile ochiului liber (valori foarte mici ale pixelilor, aproape de negru), algoritmul aplică o amplificare a luminozității (Brightness Enhancement) și a culorii pentru a face rezultatul interpretabil.

4.2. Interpretarea vizuală a hărții ELA

Rezultatul vizual al analizei ELA (imaginea neagră cu puncte colorate) se interpretează pe baza conceptului de "minim local al erorii".

- **Zonele nemodificate (Autentice):** Dacă o imagine a fost salvată de mai multe ori, blocurile de 8x8 pixeli ajung la o stare de echilibru. La o nouă recomprimare (pasul 1 al ELA), acestea se modifică foarte puțin, deoarece algoritmul de compresie a eliminat deja detaliile redundante la salvările anterioare. Prin urmare, zonele autentice vor apărea în ELA ca fiind întunecate (eroare mică).
- **Zonele modificate (Falsificate):** Atunci când inserăm un obiect nou (dintr-o altă sursă) sau desenăm peste imagine, acei pixeli noi nu au trecut prin același istoric de compresie ca restul imaginii. Ei sunt "proaspeti" din punct de vedere digital. La aplicarea ELA, acești pixeli vor suferi o degradare semnificativă, generând o diferență mare față de original.

Vizual, aceste zone vor apărea **strălucitoare** și puternic colorate, contrastând evident cu fundalul întunecat.

Pe scurt, un fals este trădat de faptul că zona inserată a fost comprimată de mai puține ori decât fundalul imaginii gazdă.

4.3. Limite și constrângeri

Deși este un instrument puternic, ELA nu este infailibil. Corectitudinea analizei depinde de câțiva factori critici:

- **Re-salvarea excesivă:** Dacă o imagine falsificată este salvată de foarte multe ori la o calitate scăzută, erorile din zona modificată se vor aplatiza, ajungând la același nivel cu fundalul, făcând detecția dificilă.
- **Rezoluția imaginii:** Imaginele foarte mici sau cele redimensionate drastic (thumbnail-uri) nu conțin suficientă informație în blocurile de 8x8 pentru a genera o hartă ELA relevantă.
- **Formatul fișierului:** Metoda funcționează optim pe fișiere JPEG. Aplicarea ELA pe formate "lossless" (fără pierderi) precum PNG sau BMP nu este relevantă, deoarece acestea nu folosesc cuantizarea pe blocuri.

CAPITOLUL 5. STUDII DE CAZ SI ANALIZA EXPERIMENTALĂ

(Aici acoperi cerința: „Descrierea imaginilor + analiza + elemente stralucire”)

Pentru a valida eficiența algoritmului ELA și funcționalitățile aplicației "Forensic ELA Expert", am derulat o serie de experimente practice. Acestea au vizat atât detecția falsurilor pe un set de date standardizat, recunoscut în mediul academic, cât și analiza unor scenarii complexe de manipulare generate intern cu ajutorul aplicației dezvoltat

5.1. Baza de date utilizată

Studiul a fost realizat pe două categorii de imagini:

1. Setul de date public CASIA v2:

Pentru calibrarea algoritmului, am utilizat imagini din setul de date *CASIA Image Tampering Detection Evaluation Database*. Acesta este un standard în domeniu, oferind perechi de imagini (Original vs. Falsificat) cu diverse tipuri de manipulări. S-au selectat

cateva imagini reprezentative (ex: Sp_D_CNN_A_ani0049...), care prezintă manipulări de tip *splicing* realizate profesionist, greu detectabile cu ochiul liber.

2. Imagini generate proprii:

Utilizând modulele aplicației mele (Lasso, AI Removal, Compresie), am generat un set de cateva imagini de test la rezoluție înaltă (1920x1080), simulând scenarii reale de "Fake News" (ex: obiecte inserate în contexte nepotrivite, modificarea elementelor etc)

5.2. Scenarii de testare și rezultate

Am supus imaginile la trei teste principale, corespunzătoare celor mai frecvente tipuri de falsificare digitală.

5.2.1. Experimentul 1: Detectia inserției de obiecte (Splicing)

Acest experiment vizează detectarea manipulărilor de tip *Image Splicing* (îmbinare), unde un obiect dintr-o imagine sursă este decupat și inserat într-o imagine destinație. Pentru a asigura o validare robustă a algoritmului ELA, am utilizat o abordare hibridă:

1. **Imagini proprii:** Generate folosind smartphone-ul personal și editate în cadrul aplicației mele "Forensic ELA Expert".
2. **Imagini standardizate:** Preluate din baza de date academică CASIA v2, recunoscută pentru testarea algoritmilor de detecție.
3. **Validare încrucișată:** Rezultatele au fost confirmate utilizând trei platforme distincte: aplicația proprie (Python), *FotoForensics.com* și *29a.ch (Forensically)*.

A. Analiza scenariului cu imagini proprii (Studiu de caz: "Pisica")

Pentru acest scenariu, am creat două compozitii diferite utilizând elemente cu caracteristici de iluminare și compresie total opuse.

- **Descrierea elementelor componente:**
 - **Imaginiile de fundal (Sursă 1 & 2):** Fotografiile 1.jpg și 2.jpg surprind peisaje rurale cu iluminare naturală complexă (lumină difuză de zi și respectiv apus). Aceste imagini conțin multe "detalii de înaltă frecvență" (fire de iarba, garduri, crengi), ceea ce înseamnă că algoritmul JPEG va genera un zgomot natural ridicat în aceste zone.
 - **Obiectul inserat (Sursă Obiect):** Imaginea obj_1.jpg (pisica neagră) a fost realizată în interior, la lumină artificială, pe un fundal textil. Obiectul are o textură uniformă (blana neagră) și un contrast ridicat.
- **Procesul de manipulare:** Folosind modulul de "Detecție Automată AI" al aplicației, am extras pisica și am inserat-o în cele două peisaje, rezultând fișierele result_1.jpg (curte) și result_2.jpg (gard la apus) asa cum se poate observa în figurile 8 și 9

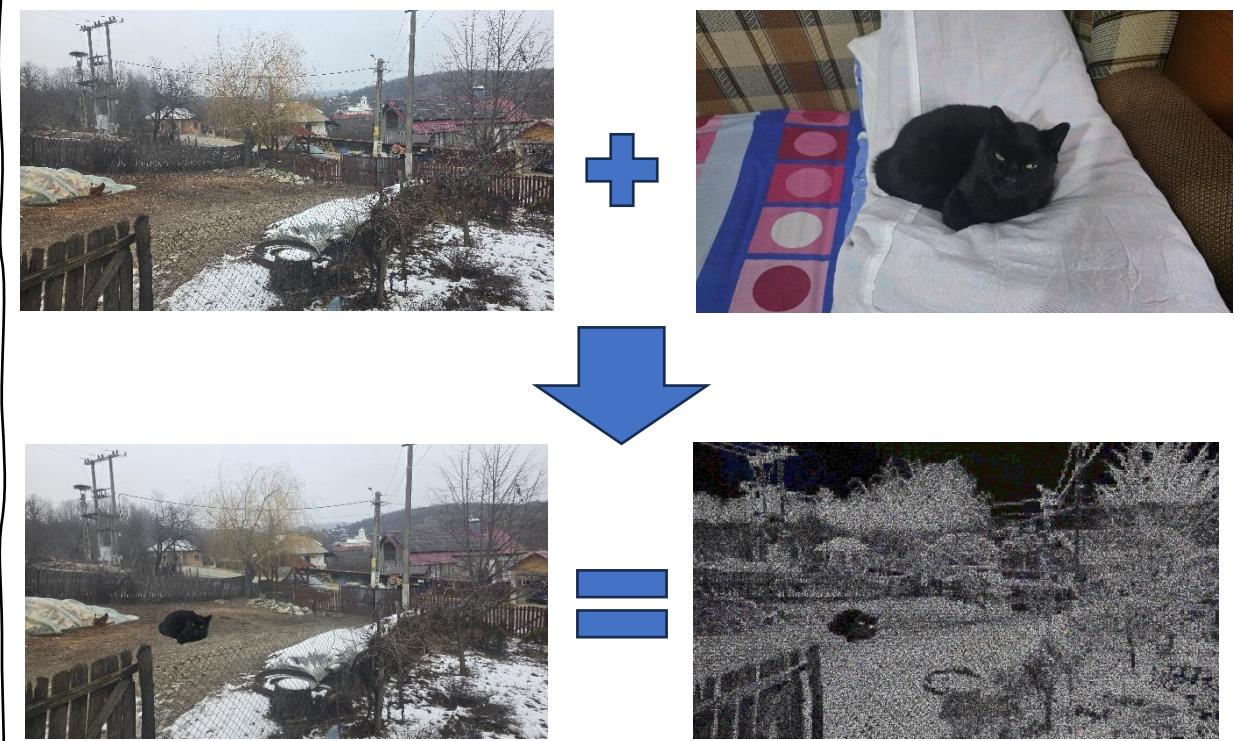


Figura 8. Experiment cu imaginile 1.jpg, obj_1.jpg, result_1.jpg si ELA_1.jpg

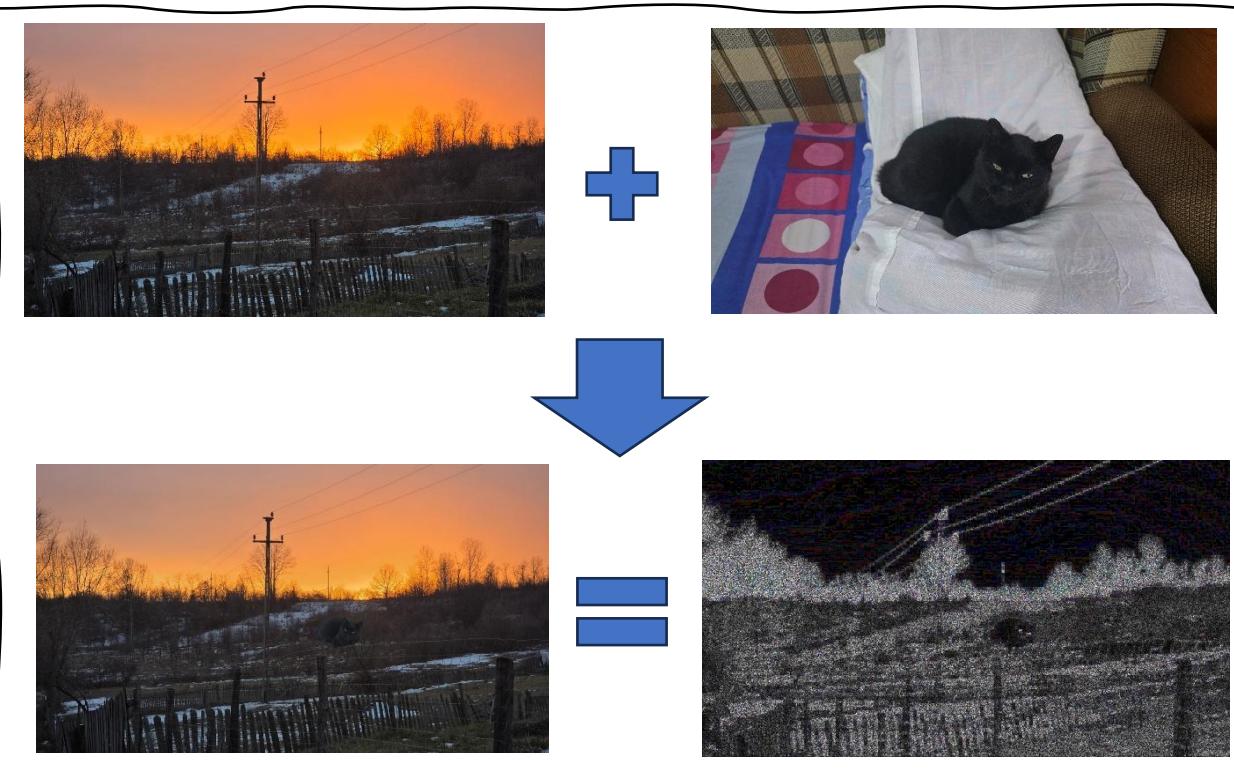


Figura 9. Experiment cu imaginile 1.jpg, obj_1.jpg, result_1.jpg si ELA_1.jpg

- **Analiza Rezultatelor ELA (ELA_1.png și ELA_2.png):** La rularea analizei ELA, s-a observat o discrepanță clară a nivelor de eroare:
 - **În ELA_1.png:** Fundalul curții prezintă un zgomot "grunjos" specific detaliilor de iarba și pietriș. Totuși, zona pisicii apare ca o pată distinctă, cu o densitate a pixelilor eronată diferită de restul solului. Marginile obiectului inserat prezintă o "coroană" de pixeli mai luminoși, indicând faptul că blocul de pixeli de la granița dintre pisică și fundal nu a fost comprimat unitar.
 - **În ELA_2.png (Apus):** Cerul, fiind o zonă cu frecvență joasă (gradient uniform), are o eroare ELA aproape nulă (negru complet). Pisica inserată pe gard iese în evidență drastic prin contrast. Deși pisica este neagră în spectrul vizual, în spectrul ELA ea conține informație de compresie diferită față de cerul din spate, demonstrând că nu aparține fotografiei originale.

B. Analiza scenariului cu imagini din baza de date (Studiu de caz: "Zebra")

Pentru calibrare, am utilizat imaginea originală Au_ani_00001.jpg (o singură zebră) și varianta manipulată Tp_S_CRN_M_B_ani00001...jpg (două zebre) ca în figura 10.

- **Descrierea imaginii:** Fotografia prezintă un contrast puternic între subiect (zebra cu dungi, frecvență înaltă) și fundal (cerul albastru, frecvență joasă).
- **Analiza ELA:** La analiza imaginii manipulate, a două zebre (cea copiată) prezintă un tip de zgomot identic cu prima, dar care nu se aliniază perfect cu grila JPEG a fundalului în nouă poziție. Validarea pe platforma 29a.ch a confirmat, prin filtrul "Noise Analysis", că ambele zebre au aceeași semnătură de zgomot, însă marginile celei de-a doua zebre prezintă artefacte de interpolare cauzate de o posibilă redimensionare sau rotire ușoară înainte de plasare.



Figura 10. Analiza ELA pe una din imaginile din baza de date CASIA v2

5.2.2. Analiza inserției cu compresie globală (figura 11):

Înainte de a analiza falsificările locale, este crucial să înțelegem cum afectează compresia JPEG o imagine în ansamblul ei. Acest experiment servește drept "baseline" (punct de referință) pentru a distinge între o imagine pur și simplu de slabă calitate și una manipulată.

- **Metodologie:** Am pornit de la imaginea originală 1.jpg (calitate ridicată, estimată >90%) și am utilizat funcția "**Globală (Toată Poza)**" din aplicație pentru a reduce calitatea întregului fișier la un nivel de **Q=40**. Rezultatul este fișierul global_compression_1.jpg.
- **Analiza Vizuală și ELA:** Comparând vizual 1.jpg cu global_compression_1.jpg, degradarea este distribuită uniform. Detaliile fine ale crengilor și textura gardului și-au pierdut claritatea pe toată suprafața imaginii. Din punct de vedere al analizei ELA, o astfel de imagine nu ridică suspiciuni de falsificare (Splicing), deoarece "Rata de Eroare" este consistentă. Deși imaginea are o calitate slabă, ea este **omogenă**. Dacă rulăm ELA, întreaga imagine va apărea fie foarte întunecată (dacă analizăm la Q=40), fie uniform de luminoasă (dacă analizăm la Q=95), dar fără "pete" de culoare distincte care să indice o inserție străină.



Figura 11. Analiza ELA pentru insertia cu compresie globala

5.2.3. Analiza compresiei locale (figura 12):

Acest experiment demonstrează una dintre cele mai subtile forme de manipulare: alterarea calității unei singure regiuni din imagine, simulând scenariul în care o porțiune (ex: un număr de înmatriculare sau o față) este "anonimizată" sau modificată și apoi re-integrată.

- **Descrierea imaginii de bază (1.jpg):** Fotografia surprinde un peisaj rural complex, caracterizat prin numeroase "detalii de înaltă frecvență": gardul de sârmă din prim-plan, crengile copacilor fără frunze și textura pietrișului. Iluminarea este difuză (cer înnorat), ceea ce elimină umbrele puternice care ar putea ascunde artefactele. Această complexitate vizuală face ca ochiul uman să ignore ușor micile imperfecțiuni, fiind candidatul ideal pentru testarea sensibilității ELA.
- **Metodologie:** Utilizând funcția dedicată "**Doar Zonă (Selectie)**" din aplicația proprie, am definit o regiune rectangulară în centrul imaginii. Algoritmul a decupat această zonă, a comprimat-o agresiv la un factor de calitate de **40%**, și a lipit-o înapoi peste originalul care avea o calitate superioară (estimată la 90-95%). Rezultatul este fișierul local_compression_1.jpg.
- **Analiza Rezultatelor:**
 - **Inspectare Vizuală:** La o privire sumară asupra imaginii local_compression_1.jpg, zona comprimată se integrează bine cromatic. Totuși, la o mărire de 200%, se pot observa artefacte de tip "blocking" (pătrate 8x8) în zona gardului din centru, detalii care s-au pierdut prin cuantizare.
 - **Analiza ELA:** Rulând analiza, discrepanța devine evidentă. Restul imaginii (zăpada, casa din dreapta) prezintă o eroare uniformă. În schimb, dreptunghiul central apare cu o intensitate diferită a erorii. Deoarece zona a fost salvată la calitate 40%, ea a pierdut deja majoritatea informațiilor pe care algoritmul ELA (care re-comprimă la 95%) se aștepta să le găsească. Această "lipsă de informație" se traduce vizual printr-o textură distinctă în harta ELA, confirmând că acea zonă are un istoric de procesare diferit de restul cadrului.



Figura 12. Analiza ELA pentru insertia cu compresie locala

5.2.4. Analiza compresiei succesive/Iterative (figura 13):

Ultimul experiment testează limitele detecției și validează teoria conform căreia erorile de compresie tind să se stabilizeze după multiple salvări, fenomen cunoscut drept "JPEG Error Convergence".

- **Metodologie:** Am utilizat funcția de automatizare "**Succesivă (Loop)**" pentru a supune imaginea originală unui ciclu de **10 re-salvări consecutive** la o calitate scăzută (**Q=40**). Procesul a simula degradarea suferită de o imagine virală distribuită repetat. Rezultatul final este `succesive_comression_x10_40quality_1.jpg`.
- **Analiza Vizuală:** Degradarea este severă. În imaginea rezultată, detaliile fine au dispărut aproape complet (firele subțiri ale gardului s-au contopit cu fundalul), iar în zonele de contrast ridicat (acoperiș vs. cer) au apărut halouri cromatice (artefacte de culoare).
- **Concluzie ELA:** Deși imaginea arată foarte rău vizual, paradoxal, analiza ELA va arăta o eroare foarte mică (imagine neagră). **Explicație:** După 10 iterații la Q=40, imaginea a ajuns la un "minim local" de energie. Algoritmul de compresie a eliminat deja tot ce putea elimina. La a 11-a salvare (efectuată de ELA), pixelii nu se mai modifică semnificativ, deoarece s-au stabilizat în noua grilă de quantizare. Acest experiment demonstrează că o imagine falsificată, dacă este re-salvată de suficiente ori, își poate "ascunde" urmele ELA, deși calitatea vizuală are de suferit iremediabil.



Figura 13. Analiza ELA pentru compresii iterative

5.3. Discuție asupra limitărilor

Deși analiza nivelului de eroare (ELA) s-a dovedit a fi un instrument eficient în detecția inserțiilor și a manipulărilor locale, experimentele derulate au scos la iveală și o serie de limitări tehnice inerente algoritmului JPEG. Este important de subliniat că ELA nu este o "probă absolută" de tipul Da/Nu, ci un instrument de vizualizare care necesită interpretarea unui expert uman.

Pe parcursul testării aplicației "Forensic ELA Expert", am identificat trei categorii majore de limitări:

1. Stabilizarea Erorii și Anti-Forensics Așa cum a demonstrat **Experimentul 4 (re-salvarea succesivă)**, ELA depinde de existența unei diferențe de potențial între zona modificată și restul imaginii. Dacă un falsificator cunoaște acest principiu, el poate salva intenționat imaginea falsificată de multiple ori (sau la o calitate foarte scăzută) înainte de distribuire. Acest proces forțează pixelii inserați să se alinieze la grila de cuantizare a imaginii gazdă. În acest caz, harta ELA va apărea uniformă (neagră), ascunzând falsul. Totuși, contramăsura este vizibilă prin degradarea calității estetice a imaginii.

2. Rezoluția și scara imaginii Algoritmul funcționează pe blocuri de **8x8 pixeli**. Pentru ca analiza să fie relevantă, obiectul inserat trebuie să fie suficient de mare pentru a acoperi un număr semnificativ de astfel de blocuri.

- În cazul imaginilor de rezoluție mică (ex: thumbnail-uri de 300x300 pixeli), numărul total de blocuri este redus, iar zgomotul de compresie devine dominant, făcând imposibilă distincția dintre un artefact de compresie și o manipulare reală.
- De asemenea, dacă imaginea a fost redimensionată (Resize) după manipulare, grila originală de 8x8 pixeli este distrusă și interpolată, generând un model de interferență (Moiré) care maschează urmele inițiale.

3. Interpretarea zonelor de înaltă frecvență (Positive false) Un risc major în interpretarea ELA este confuzia dintre manipulare și texturile naturale complexe.

- În **Experimentul 1** (imaginea result_1.jpg), am observat că pietrișul și crengile copacilor apar natural foarte "luminoase" în harta ELA. Aceasta nu este o eroare, ci o caracteristică a compresiei JPEG: zonele cu contrast ridicat (muchii ascuțite) sunt greu de comprimat și generează mereu erori mari.
- **Provocarea:** Utilizatorul trebuie să distingă între "zgomotul natural" (distribuit uniform pe zonele texturate) și "zgomotul artificial" (o pată strălucitoare într-o zonă care ar trebui să fie netedă, cum ar fi cerul sau un perete).

Concluzie asupra experimentelor: Rezultatele obținute validează utilitatea aplicației dezvoltate ca instrument de **triere preliminară**. ELA excedează în a indica *unde* să ne uităm într-o imagine suspectă, dar decizia finală trebuie coroborată și cu alte metode de analiză (analiza direcției

luminii, analiza zgomotului senzorului sau a metadatelor EXIF), întrucât compresia JPEG este un proces complex și distructiv care poate, în anumite condiții, să steargă urmele intervenției umane.

CAPITOLUL 6. CONTRIBUTIA PERSONALĂ: APLICATIE SOFTWARE PENTRU EXPERTIZĂ ELA

Pentru a transpune conceptele teoretice prezentate în capitolele anterioare într-un instrument practic, am proiectat și dezvoltat aplicația desktop "**Forensic ELA Expert**". Obiectivul a fost crearea unui mediu integrat care să permită atât generarea controlată a imaginilor falsificate (pentru studiu), cât și analiza criminalistică a acestora.

6.1. Arhitectura sistemului și tehnologii utilizate

Aplicația a fost dezvoltată în limbajul de programare **Python**, ales datorită bibliotecilor puternice de procesare a imaginii și a suportului pentru prototipare rapidă. Arhitectura este una modulară, bazată pe programare orientată pe obiecte (OOP), asigurând extensibilitatea și menținerea codului.

Principalele biblioteci software integrate în proiect sunt:

- **Tkinter:** Utilizată pentru construcția Interfeței Grafice cu Utilizatorul (GUI). Deși este biblioteca standard a Python, am extins funcționalitățile native prin desenarea vectorială pe Canvas pentru a crea elemente moderne de UI (butoane rotunjite, temă Dark Mode).
- **Pillow (PIL - Python Imaging Library):** Nucleul aplicației, responsabil pentru manipularea pixelilor, conversia spațiilor de culoare (RGB/YCbCr) și gestionarea algoritmului JPEG.
- **Rembg & U2-Net:** O bibliotecă de ultimă generație bazată pe rețele neuronale profunde, utilizată pentru segmentarea semantică și eliminarea automată a fundalului din imagini.
- **IO & OS:** Module utilizate pentru gestionarea fluxurilor de date (streams) în memorie RAM, esențiale pentru simularea compresiei fără a scrie fișiere intermedii pe disc.

6.2. Descrierea modulelor funcționale

Structura logică a aplicației este divizată în trei componente majore, gestionate de clasa principală **ForensicELAApp**.

6.2.1. Modulul de analiză ELA (Core Engine) Aceasta este motorul principal al aplicației. Funcția `run_el()` implementează algoritmul descris în Capitolul 4.

- **Flux de date:** Imaginea încărcată în memorie este salvată virtual într-un obiect `io.BytesIO` la o calitate specificată de utilizator (ex: 95%). Această operațiune simulează scrierea pe disc, folosind algoritmul JPEG să aplică quantizarea.
- **Calculul diferenței:** Imaginea este reîncărcată imediat din buffer și comparată cu originalul folosind funcția `ImageChops.difference()`.
- **Normalizare și Amplificare:** Deoarece diferențele brute sunt minime, am implementat un algoritm de auto-scalare a luminozității bazat pe valoarea maximă a erorii (extrema), urmat de o amplificare a saturăției culorilor pentru a evidenția artefactele.

6.2.2. Modulul de generare a falsurilor ("Digital Lab") Pentru a testa eficiența ELA, aplicația include un set de unelte pentru crearea falsurilor:

- **Decupare asistată de AI:** Clasa `LassoSelectorApp` integrează modelul `rembg`. La apăsarea unui singur buton, imaginea este trimisă către rețea neuronală, care returnează o mască alfa precisă, permitând extragerea subiectului (oameni, mașini, animale) fără intervenție manuală.
- **Manipulare geometrică:** Obiectele extrase pot fi rotite (0-360 grade) și scalate bicubic direct pe canvas, permitând integrarea realistă în scenă.
- **Compresia Locală (Zonală):** O inovație a acestei aplicații este capacitatea de a selecta o zonă rectangulară și de a-i aplica un factor de calitate diferit de restul imaginii. Acest lucru se realizează prin decuparea regiunii, comprimarea ei într-un buffer separat și lipirea înapoi, creând o inconsistență matematică invizibilă ochiului liber.

6.2.3. Modulul de simulare a degradării Pentru a studia fenomenul de stabilizare a erorii, am implementat funcția de **Compresie Succesivă**. Aceasta utilizează o buclă (for loop) care rezolvă imaginea de n ori (unde n este setat de utilizator, între 1 și 50). Rezultatul demonstrează practic cum o imagine virală își pierde detaliile fine și cum "semnătura" ELA se estompează în timp.

6.3. Interfața grafică și Experiența Utilizatorului (UI/UX)

S-a acordat o atenție deosebită aspectului vizual, trecând de la designul standard Tkinter la o interfață modernă, inspirată de aplicațiile profesionale de editare foto (Adobe Photoshop).

- **Tema "Dark Mode":** Utilizarea unei palete de culori închise (#1e1e1e, #252526) pentru a reduce obsoala ochilor și a pune accent pe imaginea analizată.
- **Componente personalizate:** Deoarece Tkinter nu oferă natively butoane cu colțuri rotunjite, a fost creată clasa `RoundedButton` care moștenește obiectul `Canvas`. Aceasta desenează vectorial forma butonului și gestionează evenimentele de mouse (Hover, Click) pentru a oferi un feedback vizual fluid.
- **Vizualizare Comparativă:** Fereastra de rezultat ELA include o funcționalitate interactivă "Press-to-Hold". Utilizatorul poate să apasă click stânga pe harta de eroare pentru a vedea instantaneu imaginea originală color. Această suprapunere rapidă permite corelarea directă a zonelor "strălucitoare" din ELA cu obiectele din imaginea reală.

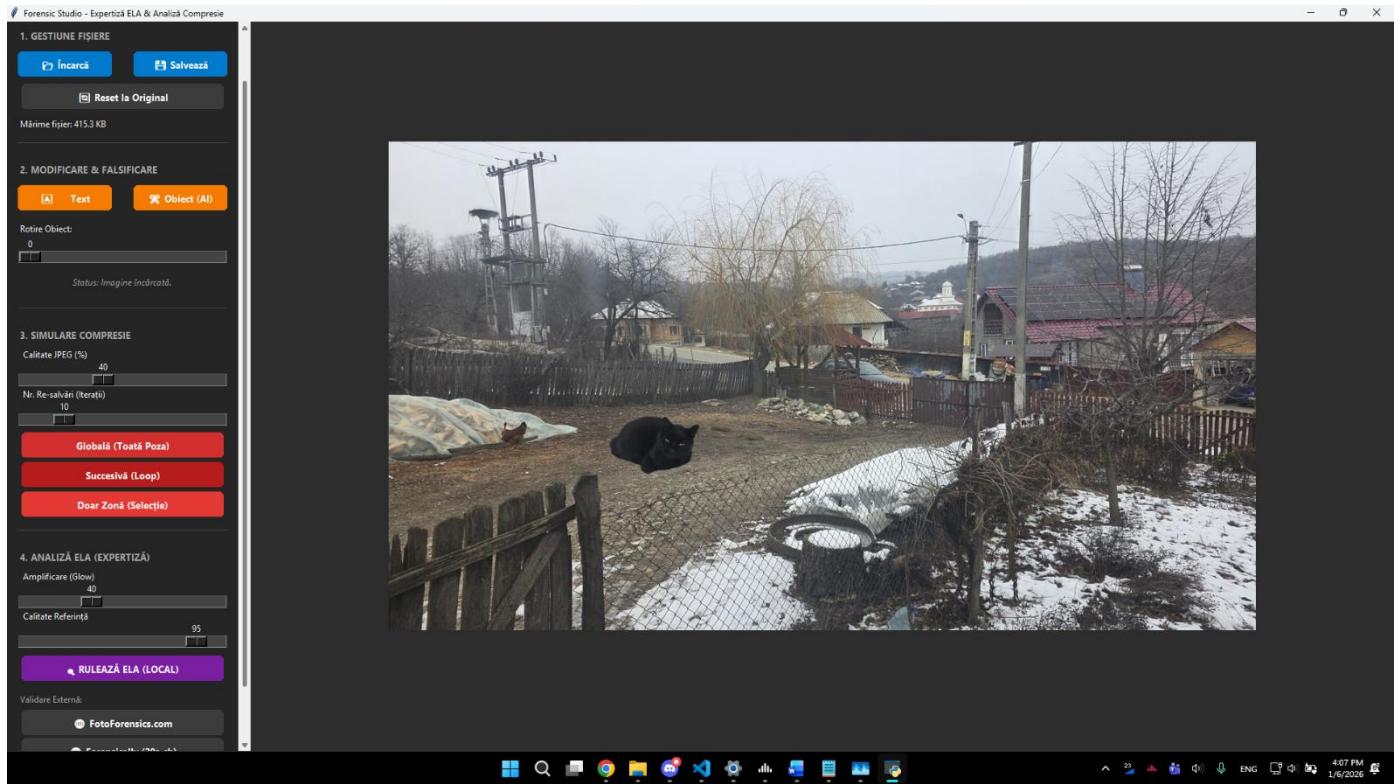


Figura 14. Interfata aplicatiei **Forensic ELA Expert**

6.4. Validare și Interconectare Aplicația nu funcționează într-o bulă izolată. Pentru a asigura rigoarea științifică, am integrat butoane de legătură directă către platformele externe de validare (*FotoForensics* și *Forensically*), încurajând utilizatorul să verifice rezultatele obținute local.

CAPITOLUL 7. CONCLUZII

Prezenta lucrare a avut ca obiectiv principal dezvoltarea și validarea unei soluții software integrate, "Forensic ELA Expert", dedicată combaterii fenomenului de falsificare a imaginilor digitale. Plecând de la studiul teoretic al compresiei JPEG și al algoritmului Error Level Analysis (ELA), proiectul a reușit să transpună concepte matematice abstracte într-o aplicație practică, accesibilă și eficientă.

7.1. Sinteza realizărilor

Principala contribuție a acestei lucrări constă în abordarea duală a problemei: aplicația dezvoltată funcționează simultan ca un laborator de **generare a falsurilor** și ca un instrument de **detectie**.

- Prin integrarea bibliotecilor de Inteligență Artificială (rembg), am demonstrat cât de ușor pot fi realizate astăzi manipulări convingătoare vizual (Splicing), subliniind necesitatea unor unelte automate de verificare.
- Implementarea modulelor de **Compresie Locală** și **Compresie Succesivă** a permis simularea unor scenarii complexe, oferind o înțelegere profundă a modului în care algoritmul JPEG distrugă sau conservă informația în funcție de setările de cuantizare.
- Din punct de vedere tehnic, aplicația se remarcă printr-o arhitectură modulară și o interfață grafică modernă, depășind limitările vizuale ale bibliotecilor standard prin implementarea unor elemente de design personalizate (Dark Mode, butoane rotunjite, interacțiune dinamică pe Canvas).

7.2. Concluzii experimentale

Testele efectuate în Capitolul 5, atât pe imagini proprii, cât și pe seturi de date standardizate, au validat eficacitatea metodei ELA ca instrument de triere preliminară (screening). Concluziile cheie sunt:

1. **Eficiență în detecția inserțiilor:** ELA este extrem de sensibilă la inconistențele dintre un obiect inserat și fundal, mai ales dacă sursele au istorice de compresie diferite. Obiectele "străine" apar cu o semnătură spectrală distinctă (strălucire intensă).
2. **Sensibilitatea la compresia locală:** Alterarea calității unei zone specifice este imediat vizibilă prin ELA, care evidențiază discrepanța dintre blocurile de 8x8 pixeli degradate și restul imaginii.
3. **Limita "Stabilizării Erorii":** Experimentele de re-salvare succesivă au confirmat vulnerabilitatea metodei. O imagine falsificată, dacă este re-salvată de suficiente ori (peste 10-15 iterații), atinge un echilibru al erorii care maschează urmele manipulării, transformând ELA într-o unealtă ineficientă în aceste cazuri specifice.

7.3. Contribuția personală

Pe lângă implementarea algoritmului clasic ELA, contribuția personală se reflectă în:

- Dezvoltarea algoritmului de **Compresie Zonală**, care permite studiul izolat al artefactelor JPEG.
- Crearea sistemului de vizualizare comparativă "**Click-and-Hold**", care facilitează corelarea instantanee între artefactele din harta de eroare și elementele vizuale din imaginea originală.
- Optimizarea experienței de utilizare (UX) prin feedback vizual și design ergonomic.

7.4. Direcții viitoare de dezvoltare

Deși aplicația "Forensic ELA Expert" este funcțională și robustă, domeniul expertizei criminalistice este vast. Direcțiile viitoare de extindere a proiectului includ:

- **Analiza Metadatelor (EXIF):** Integrarea unui modul care să extragă și să analizeze datele despre modelul camerei, data capturii și software-ul care a procesat ultima dată imaginea.
- **Analiza Zgomotului (Noise Analysis):** Implementarea unor algoritmi care să detecteze inconsistentele în zgomotul senzorului ISO, o metodă complementară ELA care funcționează bine chiar și pe imaginile re-salvate.
- **Detectie Automată cu Deep Learning:** Antrenarea unei rețele neuronale convoluționale (CNN) care să primească harta ELA ca intrare și să clasifice automat imaginea ca "Autentică" sau "Falsă", eliminând subiectivismul interpretării umane.

În concluzie, proiectul demonstrează că, deși nu există o "glonț de argint" în detectarea falsurilor, combinația dintre înțelegerea matematică a compresiei și instrumente software bine proiectate poate deconspira majoritatea manipulărilor digitale întâlnite în practică.

BIBLIOGRAFIE

A. Articole și Lucrări Științifice (Teorie)

1. **Krawetz, N.**, "A Picture's Worth: Digital Image Analysis and Forensics", Hacker Factor Solutions, 2007. (Lucrarea fundamentală care a definit metoda ELA).
2. **Wallace, G. K.**, "The JPEG still picture compression standard", IEEE Transactions on Consumer Electronics, vol. 38, no. 1, 1992. (Sursa standard pentru funcționarea compresiei JPEG și DCT).
3. **Dong, J., Wang, W., & Tan, T.**, "CASIA Image Tampering Detection Evaluation Database", Institute of Automation, Chinese Academy of Sciences, 2013. (Sursa setului de date folosit pentru testare).
4. **Farid, H.**, "Photo Forensics", MIT Press, 2016. (Carte de referință pentru toate tipurile de manipulare digitală).
5. **Voicu Ioan**, "Tehnici de verificare a autenticității imaginilor", Suport de curs/laborator, 2025. (Documentul PDF pe care l-am folosit ca bază teoretică).
6. Tehnici de Compresie a Semnalelor Multimedia Lucrare de laborator Disponibil la:
http://telecom/etc.tuiasi.ro/pns/cc/lab_cc/L04_05_TCSM_JPEG.pdf

B. Documentație Tehnică și Librării Software (Implementare)

6. **Python Software Foundation**, "Python 3.10 Documentation - Tkinter (GUI programming)". Disponibil la:
<https://docs.python.org/3/library/tkinter.html>
7. **Clark, A. (Pillow)**, "Pillow: The Friendly PIL Fork Documentation". Disponibil la: <https://pillow.readthedocs.io/>
8. **Daniel Gatis (Rembg)**, "Rembg: Tool to remove images background", GitHub Repository. Disponibil la: <https://github.com/danielgatis/rembg> (Libraria AI folosită pentru decupare).
9. **Qin, X. et al.**, "U2-Net: Going Deeper with Nested U-Structure for Salient Object Detection", Pattern Recognition, 2020. (Modelul neuronal din spatele Rembg).

C. Resurse Web și Instrumente de Validare

10. **FotoForensics**, "Error Level Analysis Tutorial & Tool". Disponibil online la:
<http://fotoforensics.com/>
11. **Jonas Wagner (29a.ch)**, "Forensically - Free Online Photo Forensics Tools".
Disponibil online la: <https://29a.ch/photo-forensics/>

Cod

```
import tkinter as tk
from tkinter import filedialog, messagebox, simpledialog, Scale, HORIZONTAL,
Toplevel
from PIL import Image, ImageTk, ImageChops, ImageEnhance, ImageDraw, ImageFont,
ImageOps
import io
import platform
import sys
import subprocess
import webbrowser

# --- CONFIGURARE TEMĂ MODERNĂ ---
COLORS = {
    "bg_main": "#1e1e1e",           # Fundal Canvas
    "bg_panel": "#252526",          # Fundal Sidebar (gri inchis)
    "text_main": "#ffffff",          # Text Alb
    "text_dim": "#aaaaaa",           # Text Gri
    "accent": "#007acc",             # Albastru VS Code
    "accent_hover": "#0098ff",
    "danger": "#d32f2f",             # Rosu
    "danger_hover": "#f44336",
    "success": "#2e7d32",            # Verde
    "success_hover": "#4caf50",
    "warning": "#f57c00",             # Portocaliu
    "warning_hover": "#ff9800",
    "purple": "#7b1fa2",              # Mov
    "purple_hover": "#9c27b0",
    "gray_btn": "#3c3c3c",            # Butoane Neutre
    "gray_hover": "#505050"
}

FONT_MAIN = ("Segoe UI", 10)
FONT_BOLD = ("Segoe UI", 10, "bold")

# --- CLASA BUTON ROTUNJIT (FIX BUG DUBLU CLICK) ---
class RoundedButton(tk.Canvas):
    def __init__(self, master, text, command, bg_color, hover_color, width=120,
height=35, corner_radius=15, **kwargs):
        # Setam fundalul canvas-ului sa fie acelasi cu al parintelui
        parent_bg = kwargs.pop('bg_parent', COLORS["bg_panel"])
```

```

        super().__init__(master, width=width, height=height, bg=parent_bg,
highlightthickness=0, **kwargs)

        self.command = command
        self.bg_color = bg_color
        self.hover_color = hover_color
        self.text_str = text
        self.radius = corner_radius
        self.W = width
        self.H = height

        # Desenam forma initiala
        self.rect_id = self.create_rounded_rect(0, 0, self.W, self.H,
self.radius, fill=bg_color, outline="")

        # Adaugam textul - ATENTIE: state='disabled' lasa click-ul sa treaca prin
text direct la buton
        self.text_id = self.create_text(self.W/2, self.H/2, text=self.text_str,
fill="white", font=FONT_BOLD, state="disabled")

        # Bindings - DOAR PE CANVAS (eliminam tag_bind care cauza dublura)
        self.bind("<Enter>", self.on_enter)
        self.bind("<Leave>", self.on_leave)
        self.bind("<Button-1>", self.on_click)
        self.bind("<ButtonRelease-1>", self.on_release)

    def create_rounded_rect(self, x1, y1, x2, y2, r, **kwargs):
        points = (x1+r, y1, x1+r, y1, x2-r, y1, x2-r, y1, x2, y1, x2, y1+r, x2,
y1+r, x2, y2-r, x2, y2-r, x2, y2, x2-r, y2, x2-r, y2, x1+r, y2, x1+r, y2, x1, y2,
x1, y2-r, x1, y2-r, x1, y1+r, x1, y1+r, x1, y1)
        return self.create_polygon(points, smooth=True, **kwargs)

    def on_enter(self, event):
        self.itemconfig(self.rect_id, fill=self.hover_color)
        self.config(cursor="hand2")

    def on_leave(self, event):
        self.itemconfig(self.rect_id, fill=self.bg_color)
        self.config(cursor="")

    def on_click(self, event):
        # Efект vizual de apasare
        self.move(self.text_id, 1, 1)

    def on_release(self, event):

```

```

        self.move(self.text_id, -1, -1)
        if self.command:
            self.command()

# =====
# CLASA LASSO & AI
# =====
class LassoSelectorApp:
    def __init__(self, master_root):
        self.top = Toplevel(master_root)
        self.top.title("Unealtă Selectie (AI & Lasso)")
        self.top.geometry("1000x750")
        self.top.configure(bg=COLORS["bg_main"])
        self.top.grab_set()

        self.source_image = None
        self.tk_source = None
        self.result_image = None
        self.points = []

    # Bara de sus
    btn_frame = tk.Frame(self.top, bg=COLORS["bg_panel"], pady=10)
    btn_frame.pack(fill=tk.X)

    # Butoane Stanga
    f_left = tk.Frame(btn_frame, bg=COLORS["bg_panel"])
    f_left.pack(side=tk.LEFT, padx=10)
    RoundedButton(f_left, "1. Încarcă Sursă", self.load_image,
COLORS["accent"], COLORS["accent_hover"], width=130).pack(side=tk.LEFT, padx=5)
    RoundedButton(f_left, "Reset", self.reset_selection, COLORS["gray_btn"],
COLORS["gray_hover"], width=80).pack(side=tk.LEFT, padx=5)

    self.lbl_info = tk.Label(btn_frame, text="(Încarcă imaginea -> Alege
metoda)", fg=COLORS["text_dim"], bg=COLORS["bg_panel"], font=FONT_MAIN)
    self.lbl_info.pack(side=tk.LEFT, padx=15)

    # Butoane Dreapta
    f_right = tk.Frame(btn_frame, bg=COLORS["bg_panel"])
    f_right.pack(side=tk.RIGHT, padx=10)
    RoundedButton(f_right, " Detectie AI", self.auto_remove_background,
COLORS["purple"], COLORS["purple_hover"], width=120).pack(side=tk.LEFT, padx=5)
    RoundedButton(f_right, " Decupare Manuală", self.finish_crop_lasso,
COLORS["success"], COLORS["success_hover"], width=150).pack(side=tk.LEFT, padx=5)

```

```

        self.canvas = tk.Canvas(self.top, bg="#2d2d2d", cursor="crosshair",
highlightthickness=0)
        self.canvas.pack(fill=tk.BOTH, expand=True)

        self.canvas.bind("<ButtonPress-1>", self.start_draw)
        self.canvas.bind("<B1-Motion>", self.draw_motion)
        self.canvas.bind("<ButtonRelease-1>", self.stop_draw)
        self.current_line = None
        self.img_offset_x = 0
        self.img_offset_y = 0

    def load_image(self):
        filename = filedialog.askopenfilename(filetypes=[("Images", "*.jpg *.jpeg
*.png")])
        if filename:
            self.source_image = Image.open(filename).convert("RGBA")
            self.source_image.thumbnail((900, 700))
            self.tk_source = ImageTk.PhotoImage(self.source_image)
            self.canvas.delete("all")
            cx = self.canvas.winfo_width() // 2
            cy = self.canvas.winfo_height() // 2
            self.canvas.create_image(cx, cy, image=self.tk_source,
anchor="center", tags="img_src")
            self.img_offset_x = cx - (self.source_image.width // 2)
            self.img_offset_y = cy - (self.source_image.height // 2)
            self.reset_selection()
            self.lbl_info.config(text="Imagine încărcată.")

    def start_draw(self, event):
        self.points = [(event.x, event.y)]
        self.current_line = self.canvas.create_line(event.x, event.y, event.x,
event.y, fill="#e74c3c", width=2, tags="lasso_line")

    def draw_motion(self, event):
        if self.points:
            self.points.append((event.x, event.y))
            coords = []
            for p in self.points: coords.extend(p)
            self.canvas.coords(self.current_line, *coords)

    def stop_draw(self, event):
        if len(self.points) > 2:
            self.canvas.create_line(self.points[-1][0], self.points[-1][1],
self.points[0][0], self.points[0][1], fill="#e74c3c", width=2, tags="lasso_line")

```

```

def reset_selection(self):
    self.canvas.delete("lasso_line")
    self.points = []

def finish_crop_lasso(self):
    if not self.source_image or len(self.points) < 3:
        messagebox.showwarning("Info", "Desenează un contur!")
        return
    mask = Image.new("L", self.source_image.size, 0)
    draw_mask = ImageDraw.Draw(mask)
    real_points = []
    for px, py in self.points:
        real_points.append((px - self.img_offset_x, py - self.img_offset_y))
    draw_mask.polygon(real_points, fill=255, outline=255)
    result = Image.new("RGBA", self.source_image.size, (0,0,0,0))
    result.paste(self.source_image, (0,0), mask)
    bbox = mask.getbbox()
    if bbox: self.result_image = result.crop(bbox)
    else: self.result_image = result
    self.top.destroy()

def auto_remove_background(self):
    if not self.source_image:
        messagebox.showwarning("Eroare", "Încarcă întâi o imagine!")
        return
    self.lbl_info.config(text="⏳ Inițializare AI...", fg="#f1c40f")
    self.top.update()
    try:
        from rembg import remove
    except ImportError:
        self.lbl_info.config(text="⬇️ Instalare librării AI...", fg="#e74c3c")
        self.top.update()
        try:
            subprocess.check_call([sys.executable, "-m", "pip", "install",
"rembg", "onnxruntime"])
            from rembg import remove
        except Exception as e:
            messagebox.showerror("Eroare", str(e))
            return

    self.lbl_info.config(text="⏳ Procesare AI...", fg="#f1c40f")
    self.top.update()
    try:
        result = remove(self.source_image)

```

```

bbox = result.getbbox()
if bbox: self.result_image = result.crop(bbox)
else: self.result_image = result
self.top.destroy()
except Exception as e:
    messagebox.showerror("Eroare AI", str(e))
    self.lbl_info.config(text="Eroare.", fg="red")

# =====
# CLASA PRINCIPALA (UI MODERN & ROUNDED)
# =====

class ForensicELAApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Forensic Studio - Expertiză ELA & Analiză Compresie")
        self.root.geometry("1280x900")
        self.root.configure(bg=COLORS["bg_main"])

        self.base_image = None
        self.display_image = None
        self.tk_image = None
        self.bg_coords = (0, 0)
        self.overlay_image = None
        self.original_overlay_image = None
        self.overlay_id = None
        self.drag_data = {"x": 0, "y": 0}
        self.interaction_mode = None
        self.text_to_place = ""
        self.current_scale = 1.0
        self.zone_start = None
        self.zone_rect_id = None

    # --- LAYOUT PRINCIPAL ---

    # 1. SIDEBAR (STÂNGA)
    sidebar_outer = tk.Frame(root, width=340, bg=COLORS["bg_panel"])
    sidebar_outer.pack(side=tk.LEFT, fill=tk.Y)

    self.sb_canvas = tk.Canvas(sidebar_outer, width=320,
    bg=COLORS["bg_panel"], highlightthickness=0)
    self.sb_canvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

    sb_scroll = tk.Scrollbar(sidebar_outer, orient="vertical",
    command=self.sb_canvas.yview)

```

```

        sb_scroll.pack(side=tk.RIGHT, fill=tk.Y)
        self.sb_canvas.configure(yscrollcommand=sb_scroll.set)

    controls = tk.Frame(self.sb_canvas, bg=COLORS["bg_panel"], padx=15,
pady=15)
        self.canvas_window = self.sb_canvas.create_window((0, 0),
window=controls, anchor="nw")

    controls.bind("<Configure>", lambda e:
self.sb_canvas.configure(scrollregion=self.sb_canvas.bbox("all")))
        self.sb_canvas.bind("<Configure>", lambda e:
self.sb_canvas.itemconfig(self.canvas_window, width=e.width))
        self._setup_mousewheel(sidebar_outer)

# 2. ZONA CANVAS (DREAPTA)
canvas_container = tk.Frame(root, bg="#333")
canvas_container.pack(side=tk.RIGHT, expand=True, fill=tk.BOTH)

self.canvas = tk.Canvas(canvas_container, bg="#2d2d2d", cursor="cross",
highlightthickness=0)
self.canvas.pack(expand=True, fill=tk.BOTH, padx=2, pady=2)

self._setup_canvas_bindings()

# --- POPULARE SIDEBAR (WIDGETS ROTUNJITE) ---

# TITLU
tk.Label(controls, text="PANOU CONTROL", font=("Segoe UI", 16, "bold"),
bg=COLORS["bg_panel"], fg=COLORS["accent"]).pack(anchor="w", pady=(0, 20))

# SECTIUNEA 1: GESTIUNE
self._add_header(controls, "1. GESTIUNE FIŞIERE")

row1 = tk.Frame(controls, bg=COLORS["bg_panel"])
row1.pack(fill=tk.X, pady=5)
RoundedButton(row1, "📁 Încarcă", self.load_base_image,
COLORS["accent"], COLORS["accent_hover"], width=130).pack(side=tk.LEFT,
padx=(0,5))
        RoundedButton(row1, "💾 Salvează", self.save_image, COLORS["accent"],
COLORS["accent_hover"], width=130).pack(side=tk.RIGHT, padx=(5,0))

# Buton Reset pe toata latimea (frame wrapper)
f_res = tk.Frame(controls, bg=COLORS["bg_panel"])
f_res.pack(fill=tk.X, pady=5)

```

```

        RoundedButton(f_res, "☒ Reset la Original", self.reset_to_original,
COLORS["gray_btn"], COLORS["gray_hover"], width=280).pack()

        self.lbl_size_comp = tk.Label(controls, text="Mărime fișier: - KB",
bg=COLORS["bg_panel"], fg=COLORS["text_main"], font=("Segoe UI", 9))
        self.lbl_size_comp.pack(anchor="w", pady=5)

        self._add_separator(controls)

# SECTIUNEA 2: FALSIFICARE
self._add_header(controls, "2. MODIFICARE & FALSIFICARE")

row2 = tk.Frame(controls, bg=COLORS["bg_panel"])
row2.pack(fill=tk.X, pady=5)
RoundedButton(row2, "Ⓐ Text", self.mode_add_text, COLORS["warning"],
COLORS["warning_hover"], width=130).pack(side=tk.LEFT, padx=(0,5))
        RoundedButton(row2, "ⓧ Obiect (AI)", self.open_lasso_tool,
COLORS["warning"], COLORS["warning_hover"], width=130).pack(side=tk.RIGHT,
padx=(5,0))

        tk.Label(controls, text="Rotire Obiect:", bg=COLORS["bg_panel"],
fg=COLORS["text_main"], font=("Segoe UI", 9)).pack(anchor="w", pady=(10,0))
        self.rotate_slider = Scale(controls, from_=0, to=360, orient=HORIZONTAL,
bg=COLORS["bg_panel"], fg=COLORS["text_main"],
troughcolor="#444", highlightthickness=0,
command=self.on_rotate_slide)
        self.rotate_slider.pack(fill=tk.X)
        self.rotate_slider.config(state=tk.DISABLED)

        self.lbl_status = tk.Label(controls, text="Status: Așteptare...", 
bg=COLORS["bg_panel"], fg=COLORS["text_dim"], font=("Segoe UI", 9, "italic"),
wraplength=260, justify="left")
        self.lbl_status.pack(fill=tk.X, pady=10, ipady=5)

        self._add_separator(controls)

# SECTIUNEA 3: COMPRESIE
self._add_header(controls, "3. SIMULARE COMPRESIE")

        self.compress_slider = Scale(controls, from_=1, to=100,
orient=HORIZONTAL, bg=COLORS["bg_panel"], fg=COLORS["text_main"],
troughcolor="#444", highlightthickness=0,
label="Calitate JPEG (%)")
        self.compress_slider.set(70)
        self.compress_slider.pack(fill=tk.X)

```

```

        self.successive_slider = Scale(controls, from_=1, to=50,
orient=HORIZONTAL, bg=COLORS["bg_panel"], fg=COLORS["text_main"],
troughcolor="#444", highlightthickness=0,
label="Nr. Re-salvări (Iterații)")
        self.successive_slider.set(1)
        self.successive_slider.pack(fill=tk.X)

        f_comp = tk.Frame(controls, bg=COLORS["bg_panel"])
        f_comp.pack(fill=tk.X, pady=5)
        # Butoane compresie unul sub altul dar centrate
        RoundedButton(f_comp, "Globală (Toată Poza)",
self.apply_compression_global, COLORS["danger"], COLORS["danger_hover"],
width=280).pack(pady=3)
        RoundedButton(f_comp, "Succesivă (Loop)",
self.apply_successive_compression, "#b71c1c", "#d32f2f", width=280).pack(pady=3)
        RoundedButton(f_comp, "Doar Zonă (Selectie)",
self.mode_local_compression, "#e53935", "#ef5350", width=280).pack(pady=3)

        self._add_separator(controls)

        # SECTIUNEA 4: ELA
        self._add_header(controls, "4. ANALIZĂ ELA (EXPERTIZĂ)")

        self.elascale = Scale(controls, from_=10, to=100, orient=HORIZONTAL,
bg=COLORS["bg_panel"], fg=COLORS["text_main"],
troughcolor="#444", highlightthickness=0,
label="Amplificare (Glow)")
        self.elascale.set(40)
        self.elascale.pack(fill=tk.X)

        self.elaq = Scale(controls, from_=50, to=100, orient=HORIZONTAL,
bg=COLORS["bg_panel"], fg=COLORS["text_main"],
troughcolor="#444", highlightthickness=0,
label="Calitate Referință")
        self.elaq.set(95)
        self.elaq.pack(fill=tk.X)

        f_elas = tk.Frame(controls, bg=COLORS["bg_panel"])
        f_elas.pack(fill=tk.X, pady=10)
        RoundedButton(f_elas, "RULEAZĂ ELA (LOCAL)", self.run_elas,
COLORS["purple"], COLORS["purple_hover"], width=280).pack()

# --- BUTOANE LINK WEB ---

```

```

        tk.Label(controls, text="Validare Externă:", bg=COLORS["bg_panel"],
fg=COLORS["text_dim"], font=("Segoe UI", 9)).pack(anchor="w", pady=(5,0))

        f_web = tk.Frame(controls, bg=COLORS["bg_panel"])
        f_web.pack(fill=tk.X, pady=2)
        RoundedButton(f_web, "🌐 FotoForensics.com", self.open_fotoforensics,
COLORS["gray_btn"], COLORS["gray_hover"], width=280).pack(pady=2)
        RoundedButton(f_web, "🌐 Forensically (29a.ch)", self.open_forensically,
COLORS["gray_btn"], COLORS["gray_hover"], width=280).pack(pady=2)

        tk.Label(controls, bg=COLORS["bg_panel"], height=3).pack()

# --- UI HELPERS ---
def _add_header(self, parent, text):
    tk.Label(parent, text=text, font=("Segoe UI", 10, "bold"),
bg=COLORS["bg_panel"], fg=COLORS["text_dim"]).pack(anchor="w", pady=(15, 5))

def _add_separator(self, parent):
    tk.Frame(parent, height=1, bg="#444").pack(fill=tk.X, pady=10)

def _setup_mousewheel(self, widget):
    widget.bind('<Enter>', lambda e: self._bind_wheel())
    widget.bind('<Leave>', lambda e: self._unbind_wheel())

def _bind_wheel(self):
    self.sb_canvas.bind_all("<MouseWheel>", self._on_mousewheel)
    if platform.system() == "Linux":
        self.sb_canvas.bind_all("<Button-4>", self._on_mousewheel)
        self.sb_canvas.bind_all("<Button-5>", self._on_mousewheel)

def _unbind_wheel(self):
    self.sb_canvas.unbind_all("<MouseWheel>")
    if platform.system() == "Linux":
        self.sb_canvas.unbind_all("<Button-4>")
        self.sb_canvas.unbind_all("<Button-5>")

def _on_mousewheel(self, event):
    if platform.system() == "Windows":
        self.sb_canvas.yview_scroll(int(-1*(event.delta/120)), "units")
    else:
        if event.num == 4: self.sb_canvas.yview_scroll(-1, "units")
        elif event.num == 5: self.sb_canvas.yview_scroll(1, "units")

def _setup_canvas_bindings(self):
    self.canvas.bind("<Button-1>", self.on_canvas_click)

```

```

    self.canvas.bind("<B1-Motion>", self.on_canvas_drag)
    self.canvas.bind("<ButtonRelease-1>", self.on_canvas_release)
    self.canvas.bind("<Button-3>", self.on_right_click)
    if platform.system() == "Linux":
        self.canvas.bind("<Button-4>", self.on_mouse_wheel_zoom)
        self.canvas.bind("<Button-5>", self.on_mouse_wheel_zoom)
    else:
        self.canvas.bind("<MouseWheel>", self.on_mouse_wheel_zoom)

# --- FUNCTIONALITY ---

def update_size_label(self):
    if self.display_image:
        buf = io.BytesIO()
        self.display_image.save(buf, "JPEG", quality=95)
        s = buf.tell() / 1024.0
        self.lbl_size_comp.config(text=f"Mărime fișier: {s:.1f} KB")

def apply_transformations(self):
    if self.original_overlay_image and self.overlay_id:
        img = self.original_overlay_image.copy()
        angle = self.rotate_slider.get()
        if angle != 0: img = img.rotate(angle, expand=True,
resample=Image.BICUBIC)
        if self.current_scale != 1.0:
            new_w = int(img.width * self.current_scale)
            new_h = int(img.height * self.current_scale)
            img = img.resize((max(1, new_w), max(1, new_h)), Image.LANCZOS)
        self.overlay_image = img
        coords = self.canvas.coords(self.overlay_id)
        cx, cy = coords[0], coords[1]
        self.tk_overlay = ImageTk.PhotoImage(self.overlay_image)
        self.canvas.itemconfig(self.overlay_id, image=self.tk_overlay)
        self.canvas.coords(self.overlay_id, cx, cy)

def on_rotate_slide(self, val):
    if self.interaction_mode == "move_object": self.apply_transformations()

def on_mouse_wheel_zoom(self, event):
    if self.interaction_mode == "move_object" and
self.original_overlay_image:
        delta = 0
        if platform.system() == "Linux":
            if event.num == 4: delta = 1
            elif event.num == 5: delta = -1

```

```

        else: delta = event.delta
        if delta > 0: self.current_scale *= 1.1
        elif delta < 0: self.current_scale *= 0.9
        self.current_scale = max(0.1, min(self.current_scale, 3.0))
        self.lbl_status.config(text="Status: Scalare
{self.current_scale*100}%.0f")
        self.apply_transformations()

def open_lasso_tool(self):
    if not self.display_image:
        messagebox.showwarning("Atentie", "Incarca intai imaginea de baza!")
        return
    lasso_app = LassoSelectorApp(self.root)
    self.root.wait_window(lasso_app.top)
    if lasso_app.result_image:
        self.overlay_image = lasso_app.result_image
        self.place_overlay_on_canvas()
    else:
        self.lbl_status.config(text="Status: Selectie anulata.")

def place_overlay_on_canvas(self):
    if self.overlay_image.width > 500 or self.overlay_image.height > 500:
        self.overlay_image.thumbnail((500, 500))
    self.original_overlay_image = self.overlay_image.copy()
    self.current_scale = 1.0
    self.rotate_slider.set(0)
    self.tk_overlay = ImageTk.PhotoImage(self.overlay_image)
    cx = self.canvas.winfo_width() // 2
    cy = self.canvas.winfo_height() // 2
    self.overlay_id = self.canvas.create_image(cx, cy, image=self.tk_overlay,
tags="overlay", anchor="center")
    self.interaction_mode = "move_object"
    self.rotate_slider.config(state=tk.NORMAL)
    self.lbl_status.config(text="Status: Obiect activ. Scroll pt zoom, Slider
pt rotire.")

def load_base_image(self):
    fn = filedialog.askopenfilename(filetypes=[("Images", "*.jpg *.png")])
    if fn:
        self.base_image = Image.open(fn).convert("RGB")
        self.base_image.thumbnail((1200, 900))
        self.display_image = self.base_image.copy()
        self.update_canvas(self.display_image)
        self.update_size_label()
        self.lbl_status.config(text="Status: Imagine incarcata.")

```

```

        self.interaction_mode = None
        self.rotate_slider.set(0)
        self.rotate_slider.config(state=tk.DISABLED)

    def update_canvas(self, pil_img):
        self.display_image = pil_img
        self.tk_image = ImageTk.PhotoImage(pil_img)
        self.canvas.delete("all")
        cx, cy = self.canvas.winfo_width(), self.canvas.winfo_height()
        if cx<10: cx=800
        if cy<10: cy=600
        ix, iy = max(0, (cx-pil_img.width)//2), max(0, (cy-pil_img.height)//2)
        self.canvas.create_image(ix, iy, image=self.tk_image, anchor="nw",
tags="background")
        self.bg_coords = (ix, iy)

    def mode_add_text(self):
        if not self.display_image: return
        t = simpledialog.askstring("Text", "Text de inserat:")
        if t: self.text_to_place = t; self.interaction_mode = "text";
self.lbl_status.config(text="Status: Click pe imagine pt text.")

    def mode_local_compression(self):
        if not self.display_image: return
        self.interaction_mode = "select_zone"
        self.lbl_status.config(text="Status: Trage cu mouse-ul un DREPTUNGHI
pentru a comprima zona interioară.")

    def on_canvas_click(self, e):
        if self.interaction_mode == "text":
            rx, ry = int(e.x - self.bg_coords[0]), int(e.y - self.bg_coords[1])
            if 0 <= rx < self.display_image.width and 0 <= ry <
self.display_image.height:
                draw = ImageDraw.Draw(self.display_image)
                try: font = ImageFont.truetype("arial.ttf", 50)
                except: font = ImageFont.load_default()
                draw.text((rx, ry), self.text_to_place, fill="red", font=font)
                self.update_canvas(self.display_image)
                self.interaction_mode = None
                self.update_size_label()
        elif self.interaction_mode == "move_object":
            item = self.canvas.find_closest(e.x, e.y)
            if "overlay" in self.canvas.gettags(item):
                self.drag_data["item"] = item; self.drag_data["x"] = e.x;
self.drag_data["y"] = e.y

```

```

        elif self.interaction_mode == "select_zone":
            self.zone_start = (e.x, e.y)
            if self.zone_rect_id: self.canvas.delete(self.zone_rect_id)
            self.zone_rect_id = self.canvas.create_rectangle(e.x, e.y, e.x, e.y,
outline="red", width=2, dash=(4, 4))

    def on_canvas_drag(self, e):
        if self.interaction_mode == "move_object" and "item" in self.drag_data:
            dx, dy = e.x - self.drag_data["x"], e.y - self.drag_data["y"]
            self.canvas.move(self.drag_data["item"], dx, dy)
            self.drag_data["x"], self.drag_data["y"] = e.x, e.y
        elif self.interaction_mode == "select_zone" and self.zone_start:
            x1, y1 = self.zone_start
            self.canvas.coords(self.zone_rect_id, x1, y1, e.x, e.y)

    def on_canvas_release(self, e):
        if self.interaction_mode == "select_zone" and self.zone_start:
            x1, y1 = self.zone_start
            x2, y2 = e.x, e.y
            if self.zone_rect_id: self.canvas.delete(self.zone_rect_id)
            self.zone_rect_id = None; self.zone_start = None

            left_canvas, top_canvas = min(x1, x2), min(y1, y2)
            right_canvas, bottom_canvas = max(x1, x2), max(y1, y2)
            if (right_canvas - left_canvas) < 5 or (bottom_canvas - top_canvas) <
5: return

            img_left = max(0, int(left_canvas - self.bg_coords[0]))
            img_top = max(0, int(top_canvas - self.bg_coords[1]))
            img_right = min(self.display_image.width, int(right_canvas -
self.bg_coords[0]))
            img_bottom = min(self.display_image.height, int(bottom_canvas -
self.bg_coords[1]))

            self.apply_compression_local(img_left, img_top, img_right,
img_bottom)
            self.interaction_mode = None
            self.lbl_status.config(text="Status: Compresie locală aplicată.")

    def on_right_click(self, e):
        if self.interaction_mode == "move_object" and self.overlay_image:
            ox, oy = self.canvas.coords(self.overlay_id)
            top_left_x = ox - (self.overlay_image.width / 2)
            top_left_y = oy - (self.overlay_image.height / 2)
            paste_x = int(top_left_x - self.bg_coords[0])

```

```

        paste_y = int(top_left_y - self.bg_coords[1])

        if self.overlay_image.mode == 'RGBA':
            self.display_image.paste(self.overlay_image, (paste_x, paste_y),
self.overlay_image)
        else:
            self.display_image.paste(self.overlay_image, (paste_x, paste_y))

        self.canvas.delete(self.overlay_id)
        self.overlay_id = None; self.overlay_image = None;
self.original_overlay_image = None
        self.interaction_mode = None
        self.rotate_slider.config(state=tk.DISABLED)
        self.update_canvas(self.display_image)
        self.update_size_label()
        self.lbl_status.config(text="Status: Obiect fixat.")

    def apply_compression_global(self):
        if not self.display_image: return
        q = self.compress_slider.get()
        buf = io.BytesIO()
        self.display_image.save(buf, "JPEG", quality=q)
        s = len(buf.getvalue())/1024.0
        self.lbl_size_comp.config(text=f"Mărime fișier: {s:.1f} KB")
        buf.seek(0)
        self.display_image = Image.open(buf).convert("RGB")
        self.update_canvas(self.display_image)
        messagebox.showinfo("Compresie", f"Comprimat Global la Q={q}. Mărime:
{s:.1f} KB")

    def apply_successive_compression(self):
        if not self.display_image: return
        iterations = self.successive_slider.get()
        quality = self.compress_slider.get()

        self.lbl_status.config(text=f"▣ Se comprimă succesiv de {iterations}
ori...", fg=COLORS["warning"])
        self.root.update()

        current_img = self.display_image
        for i in range(iterations):
            buffer = io.BytesIO()
            current_img.save(buffer, format="JPEG", quality=quality)
            buffer.seek(0)
            current_img = Image.open(buffer).convert("RGB")

```

```

        self.display_image = current_img
        self.update_canvas(self.display_image)
        self.update_size_label()

        messagebox.showinfo("Succes", f"Imaginea a fost re-salvată de
{iterations} ori la calitatea {quality}%.")
        self.lbl_status.config(text=f"Status: Compresie Succesivă ({iterations}x)
finalizată.", fg=COLORS["text_dim"])

def apply_compression_local(self, x1, y1, x2, y2):
    try:
        crop = self.display_image.crop((x1, y1, x2, y2))
        quality = self.compress_slider.get()
        buffer = io.BytesIO()
        crop.save(buffer, format="JPEG", quality=quality)
        buffer.seek(0)
        degraded_crop = Image.open(buffer).convert("RGB")
        self.display_image.paste(degraded_crop, (x1, y1))
        self.update_canvas(self.display_image)
        self.update_size_label()
    except Exception as e:
        messagebox.showerror("Eroare", f"Nu s-a putut comprima zona: {e}")

def run_ela(self):
    if not self.display_image: return

    orig = self.display_image.convert("RGB")
    ela_q = self.elas_quality.get()
    user_scale = self.elas_scale.get()

    buf = io.BytesIO()
    orig.save(buf, "JPEG", quality=ela_q)
    buf.seek(0)
    recomp = Image.open(buf).convert("RGB")

    ela_img = ImageChops.difference(orig, recomp)
    extrema = ela_img.getextrema()
    max_diff = max([ex[1] for ex in extrema])
    if max_diff == 0: max_diff = 1
    scale_factor = (255.0 / max_diff) * (user_scale / 20.0)

    # Amplificare pixel-level (mai precis)
    r, g, b = ela_img.split()
    scale_func = lambda i: i * scale_factor

```

```

r = r.point(scale_func)
g = g.point(scale_func)
b = b.point(scale_func)
ela_img = Image.merge("RGB", (r, g, b))
ela_img = ImageEnhance.Color(ela_img).enhance(3.0)

top = Toplevel(self.root)
top.title(f"Rezultat ELA - Mod Comparativ")
top.geometry("1000x850")
top.configure(bg="#000")

tk_ela = ImageTk.PhotoImage(ela_img)
tk_orig = ImageTk.PhotoImage(orig)

lbl_img = tk.Label(top, image=tk_ela, bg="#000", bd=0, cursor="hand2")
lbl_img.pack(expand=True)
lbl_img.img_ela = tk_ela
lbl_img.img_orig = tk_orig

def show_original(event): lbl_img.configure(image=lbl_img.img_orig)
def show_ela(event): lbl_img.configure(image=lbl_img.img_ela)

lbl_img.bind("<ButtonPress-1>", show_original)
lbl_img.bind("<ButtonRelease-1>", show_ela)

info_text = (f"Eroare Max: {max_diff} | Amplificare:\n"
{scale_factor:.1f}\n"
f"COMPARARE: Ține CLICK STÂNGA apăsat pe imagine pentru a
vedea ORIGINALUL.")
tk.Label(top, text=info_text, bg="#000", fg="#aaa", font=("Segoe UI",
10)).pack(pady=10)

def reset_to_original(self):
    if self.base_image:
        self.display_image = self.base_image.copy()
        self.update_canvas(self.display_image)
        self.canvas.delete("overlay")
        self.rotate_slider.config(state=tk.DISABLED)
        self.update_size_label()

def save_image(self):
    if self.display_image:
        f = filedialog.asksaveasfilename(defaultextension=".jpg",
filetypes=[("JPEG", "*.jpg")])
        if f: self.display_image.save(f, quality=95)

```

```
def open_fotoforensics(self):
    webbrowser.open("http://fotoforensics.com/")

def open_forensically(self):
    webbrowser.open("https://29a.ch/photo-forensics/#error-level-analysis")

if __name__ == "__main__":
    root = tk.Tk()
    app = ForensicELAApp(root)
    root.mainloop()
```