

# Computation of String Art

Adrian-Lucian Mincu

Facultatea de Matematică și Informatică

Martie 26, 2025

# Overview

1. Abstract
2. Obiectivul Lucrării
3. Introducere
4. State of the art
5. Preliminarii
6. Definirea Obiectivului
7. Metodologie
8. Evaluarea Performanței și Analiză
9. AI
10. U-Net
11. GAN
12. Bibliografie

## Abstract

Această prezentare abordează tema computației a ceea ce se numește “String Art”, utilizând diverse metode algoritmice, cât și dezvoltarea unui model de învățare automată (ML) capabil să genereze String Art. O sursă semnificativă de inspirație a fost lucrarea de cercetare numită “String Art: Towards Computational Fabrication of String Images” [Birsak et al., 2018].

În mod normal, artiștii ar trebui să găsească intuitiv un aranjament al aței care să recreeze o imagine dată. De aceea, o să tratez problema găsirii unei căi matematice de a computa string art, reușind să automatizez procesul.

## Obiectivul Lucrării

Scopul intregii lucrări este de a implementa mai multe metode algoritmice care să rezolve această problemă, construind astfel un set de date relevant. Acest dataset va fi utilizat pentru antrenarea unui model de inteligență artificială, care va învăța să reproducă și să optimizeze procesul de generare a acestor modele.

# String Art

Așa cum am spus mai sus, string art este o tehnică de a crea un efect vizual manipulând ata pentru a forma un model și a reproduce o imagine.

Obiectivul este de a replica acest efect, în mod digital, plecând de la o imagine dată ca input.

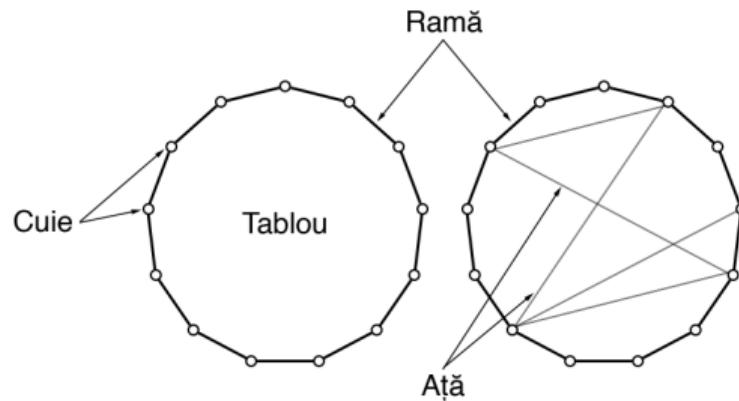


Figure: Imagine care să illustreze fiecare componentă din procesul de string art.

Definiția string art-ului pe care am adresat-o mai devreme nu o să fie aplicată complet când voi recrea imaginile digital, adică unele reguli vor fi relaxate pentru a explora posibilitățile și limitele acestui domeniu.

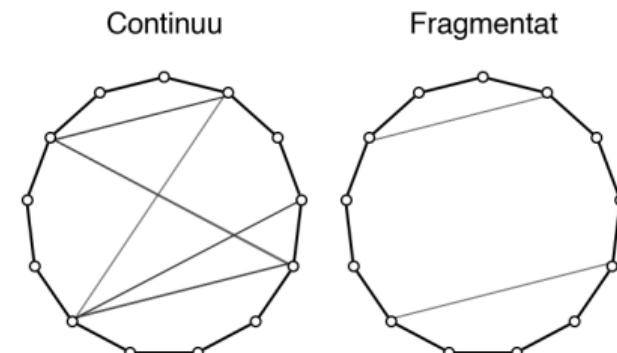


Figure: Imagine care să illustreze diferite tipuri de aranjamente: continuu versus fragmentat.

## State of the art

Până în prezent, soluția abordată a fost cea algoritmică, utilizând fie un algoritm Greedy, fie metoda Celor Mai Mici Patrate (CMMP).  
În continuare, voi prezenta ambele metode în varianta implementată recent de mine.

# Terminologie

Putem defini urmatoarele notații:

$m$  = Numărul de linii din imagine

$n$  = Numărul de coloane din imagine

$y$  = Imaginea inițială alb-negru.  $y \in \mathbb{R}^{m, n}$

$Y$  = Versiunea aplatizată pe rânduri și preprocesată a imaginii.  $Y \in [0, 1]^{m \cdot n} \subset \mathbb{R}^{m \cdot n}$

$p$  = Numărul de cuie folosite

$I = \binom{p}{2}$ , numărul de linii totale care pot fi trase

$M_i$  = O matrice care reprezintă a  $i$ -a linie desenată pe tablou

## Terminologie

$M =$  Pentru fiecare  $M_i$ , coloana corespunzătoare este versiunea aplatizată pe rânduri a  $M_i$ .

$$M \in \mathbb{R}^{m \cdot n, l}$$

Vectorul coloană  $X$  care este output-ul algoritmului nostru, unde fiecare element din el,  $x_i$  reprezintă dacă linia  $i$  va fi trasată sau nu.

$$x_i = \begin{cases} 1 & \text{dacă linia } i \text{ este trasată} \\ 0 & \text{altfel} \end{cases}, X \in \mathbb{R}^l$$

## Funcția Obiectiv

Astfel, obiectivul nostru final este să aflăm valorile elementelor vectorului  $X$  care ne aduc cel mai aproape de imaginea inițială. Deci trebuie să minimizăm:

$$\min \left\| \sum (M_i \cdot x_i) - y \right\|$$

sau

$$\min \|M \cdot X - Y\|$$

# Calitatea Subiectivă a Imaginei

Este important de menționat că există o discrepanță între modul în care calculatorul percepce o imagine calitativă și modul în care acesta este percepță de ochiul uman. În formula de mai sus, calculatorul va considera că imaginea cu cea mai mică eroare este cea de cea mai înaltă calitate.



i03\_24\_5, MOS=0.4, FSIMc=0.693



i03\_13\_4, MOS=2.9, FSIMc=0.671

Fig. 27. Example of contradiction between FSIMc and MOS for the test image # 25.

Figure: Exemplu din lucrarea [et al., 2015] care evidențiază discrepanța între erorile computate de calculator și calitatea percepță.

## Calitatea Subiectivă a Imaginei

Acest fenomen poate fi observat în imaginea de mai sus 3, unde poza din stânga, deși are un scor FSIMc [Zhang et al., ] mai mare (un scor mai mare indicând teoretic o calitate mai bună) scorul MOS<sup>1</sup> este mai mic.

Acest lucru înseamnă că majoritatea oamenilor au percepția imaginea din dreapta ca fiind mai clară și mai calitativă, contrar evaluării calculate algoritmice. Trebuie menționat că FSIMc este o metodă mai avansată decât MSE (folosită în această lucrare) și, cu toate acestea, poate să dea greș în evaluarea calității.

---

<sup>1</sup>Pentru detalii suplimentare despre Scorul Mediu de Opinie (MOS), consultați articolul dedicat Mean Opinion Score (MOS)

Pentru a plasa cuiele echidistant unul față de celălalt pe o formă circulară, am utilizat ecuația parametrică a cercului.

Pentru un cerc centrat în  $(x_c, y_c)$  cu rază  $r$  și  $p$  cuie:

$$x_k = x_c + r \cdot \cos \frac{2\pi k}{p}$$

$$y_k = y_c + r \cdot \sin \frac{2\pi k}{p}$$

for  $k = 0, 1, 2, \dots, p - 1$

Pentru a trasa linii în reprezentarea matricială, algoritmul Bresenham<sup>2</sup> a fost folosit, determinând punctele necesare care să fie selectate între două cuie alese pentru a avea o reprezentare apropiată de o linie dreaptă desenată.

---

<sup>2</sup>Pentru implementarea detaliată a algoritmului, consultați articolul despre Bresenham's Line Algorithm

## Obținerea Rezultatului

După obținerea vectorului  $X$ , imaginea finală este generată înmulțind matricea de linii cu vectorul coloană  $X$ , rezultând combinațiile de linii care trebuie trasate:  $O = MX$ . Totuși, valorile din  $O$  pot depăși intervalul  $[0, 1]$ .

Pentru a rezolva acestă problemă trebuie introdusă o funcție de limitare [Socha, 2024].  $C = \mathbb{R}^+ \rightarrow [0, 1]$ .  $C(x) = \max(0, \min(x, 1))$ . În plus, pentru a desena imaginea cu culoarea neagră, trebuie inversate valorile culorilor. În final, acestea sunt scalate în intervalul  $[0, 255]$ , specific imaginilor digitale. Formula completă devine:

$$O = (1 - C(MX)) \cdot 255$$

# Metoda celor Mai Mici Pătrate (CMMP)

O primă metodă, care a fost inspirată din videoclipul <sup>3</sup> este cea folosind **CMMP**.

Metoda CMMP minimizează funcția menționată mai sus:  $\min \|M \cdot X - Y\|$  prin vectorul  $X$  astfel încât norma  $l_2$  dintre imaginea originală și cea rezultată să fie minimă.

Metoda CMMP poate fi implementată prin două abordări.

- Reprezentare densă a matricei.
- Reprezentare rară a matricei.

---

<sup>3</sup>Inspirația pentru această metodă poate fi găsită în videoclipul The Mathematics of String Art

## Metoda celor Mai Mici Pătrate (CMMP) Densă

În această abordare, matricea  $M$  este reprezentată ca o matrice densă (plină), cu dimensiunile:  $M \in \mathbb{R}^{m \cdot n, l}$ .

Deși această abordare este simplă și ușor de înțeles, reprezentarea densă este mult mai costisitoare din punct de vedere computațional pentru matrici mai mari, deoarece fiecare element, incluzând și zero-urile, este stocat explicit.

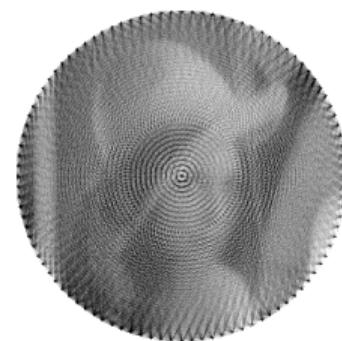


Figure: Imagine rezultată în urma procesului de CMMP cu reprezentare densă a matricii.

## Metoda celor Mai Mici Pătrate (CMMP) Rară

A doua abordare este cea cu reprezentare rară a matricilor<sup>4</sup>.

Știind faptul că vectorii noștri coloană din reprezentarea densă a matricei constituie o matrice aplatizată pe rânduri în care o singură linie a fost trasă, atunci putem spune că matricea este una rară, însemnând că majoritatea elementelor sunt zero.

În acest caz, algoritmii CMMP au implementări speciale [csr, ] care să profite de structura rară a matricii. Sunt mai rapizi și folosesc mai puțină memorie pentru că procesează doar elementele diferite de zero.

---

<sup>4</sup>Pentru detalii despre reprezentarea rară a matricilor, consultați Sparse Matrix

## Metoda celor Mai Mici Pătrate (CMMP) Rară

De exemplu, cel mult  $\max(m, n)$  elemente ale matricei  $M_i$  au valori diferite de 0, datorită algoritmului Bresenham. Observați imaginea de mai jos.

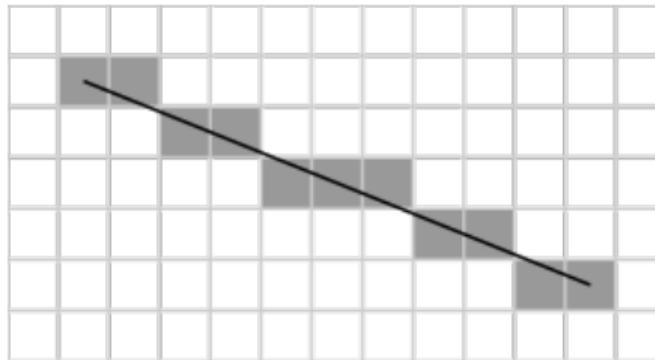


Figure: Imagine care să demonstreze modul de desenare al algoritmului Bresenham.

În exemplul de mai sus, unde  $m = 5$ ,  $n = 1$ , linia trasată are exact 11 elemente diferite de zero.

# Metoda celor Mai Mici Pătrate (CMMP) Rară

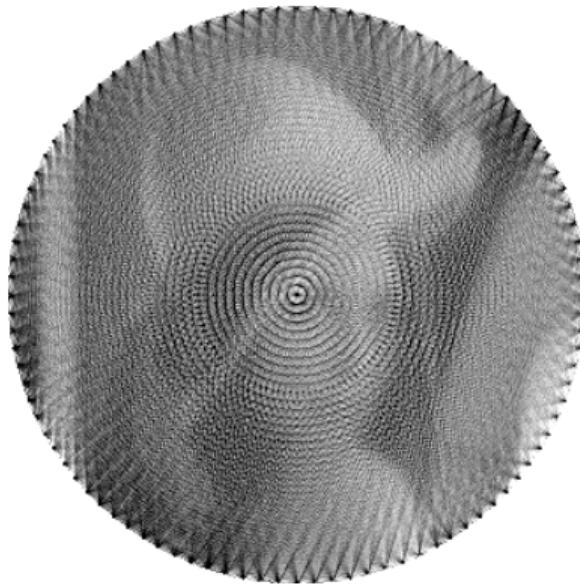


Figure: Imagine rezultată în urma procesului de CMMP cu reprezentare rară a matricii.

## Matching Pursuit Greedy

Metoda Greedy constă în alegerea unei linii la fiecare pas. Această alegere este făcută astfel încât să minimizeze eroarea dintre imaginea originală și imaginea curentă la pasul  $k$ .

Având în vedere timpul de calcul mare, cauzat de necesitatea de a rezolva o problemă de tip CMMP la fiecare pas iterativ, am decis să integrez două euristici:

1. Euristică Random

Procesul este simplu, se selectează random  $k$  linii candidat.

2. Euristică Produs Scalar

În acest caz se computează  $MY$  și se aleg primele  $k$  linii care nu au fost desenate încă și au cel mai mare produs scalar.

## Metoda Greedy Rezultate

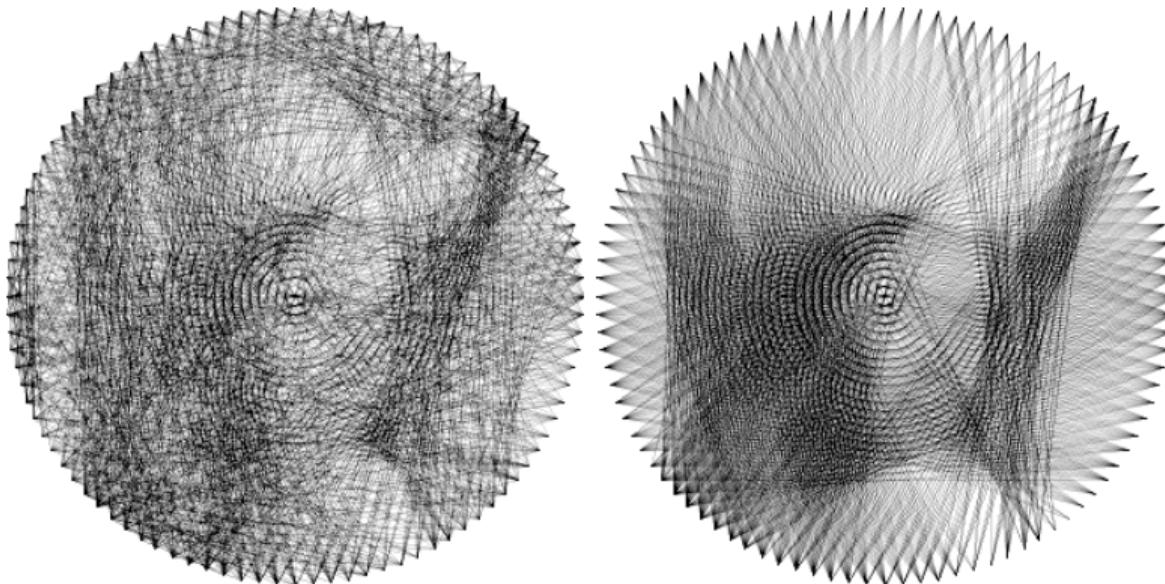


Figure: Imagini rezultate în urma procesului de Greedy cu euristică random (stânga) și euristică produs scalar (dreapta) cu 1000 de linii selectate.

# Orthogonal Matching Pursuit (OMP)

Metoda Orthogonal Matching Pursuit (OMP) este foarte similară cu cea Greedy, având în procesul iterativ același algoritm bazat pe selectia liniei care reduce cel mai mult eroarea.

Spre deosebire de metoda Greedy, OMP actualizează reziduul  $Y$  (initial fiind imaginea originală) prin scăderea proiecției liniei selectate.

Astfel se garantează ca reziduul rămas este ortogonal față de liniile alese anterior.

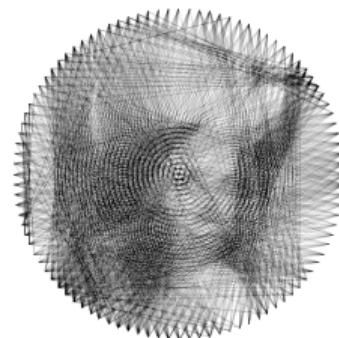


Figure: Imagine rezultată în urma procesului de Orthogonal Matching Pursuit (OMP) cu 1000 de linii selectate.

# Input

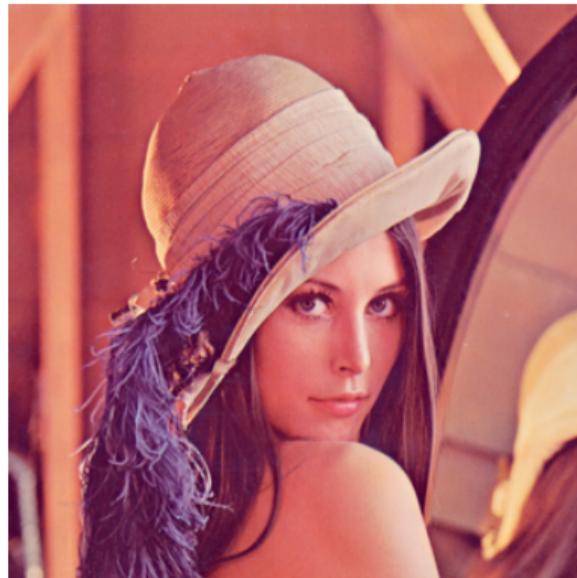


Figure: Imaginea originală.

## Rezultate CMMP

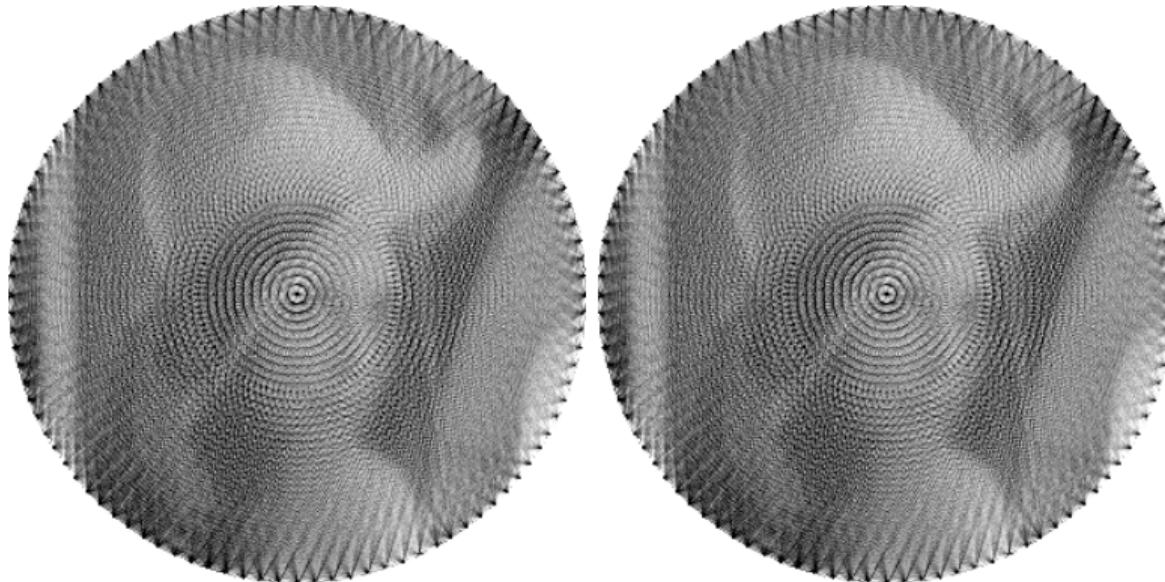


Figure: Imagini rezultate în urma procesului de CMMP cu reprezentări matriciale dense (stânga) și rare (dreapta).

## Rezultate Matching Pursuit

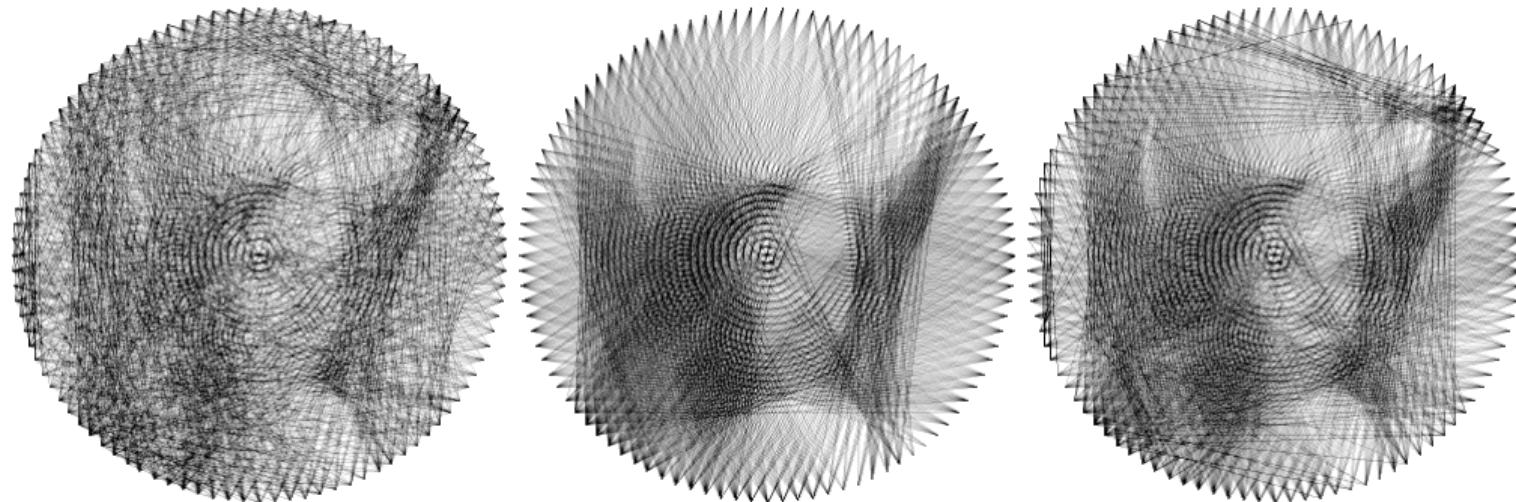


Figure: Imagini rezultate în urma procesului de Greedy cu euristică random (stânga) și euristică produs scalar (mijloc) și OMP (dreapta), cu 1000 de linii selectate.

## Observații vizuale

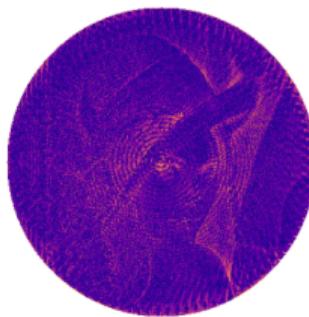
Se poate observa încă de la început că cele mai bune rezultate pentru ochiul uman sunt cele care folosesc metoda CMMP. Acest lucru se întâmplă din cauză că ele utilizează toate liniile posibile și ajustează intensitatea lor între 0 (alb) și 1 (negru).

Pe de altă parte, abordările Greedy, care selectează un număr limitat de linii (în acest caz, 1000), produc reconstrucții mai puțin netede. Euristica random (stânga) tinde să fie mai haotică, alegând linii din diferite regiuni ale imaginii. În schimb, euristica produsului scalar (mijloc) se concentrează pe anumite zone, ducând la rezultate mai localizate, dar mai puțin variate.

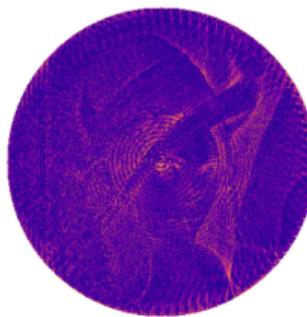
Metoda OMP pare să fie foarte apropiată cu cea de Greedy care folosește euristica produs scalar. Doar că din cauza diferențelor în implementare, se poate observa că OMP are o alegere mai diversă, mai punctual, selectează și liniile de pe margine.

# Imagini Diferență

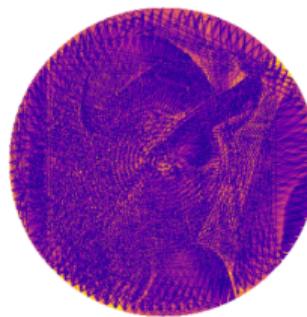
Imaginile diferență evidențiază discrepanțele dintre imaginea originală și rezultatul fiecărei metode.



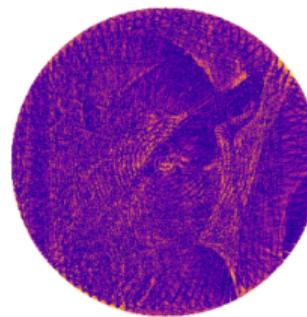
least\_squares  
matrix\_representation: dense  
RMS: 0.6681



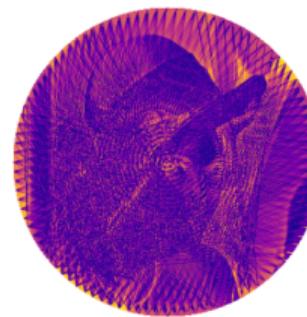
least\_squares  
matrix\_representation: sparse  
RMS: 0.6681



matching\_pursuit  
number\_of\_lines: 1000  
mp\_method: orthogonal  
RMS: 0.7437



matching\_pursuit  
number\_of\_lines: 1000  
mp\_method: greedy  
selector\_type: random  
RMS: 0.7406



matching\_pursuit  
number\_of\_lines: 1000  
mp\_method: greedy  
selector\_type: dot-product  
RMS: 0.7737

Figure: Diferența imaginilor rezultate până la imaginea originală.

# Timp de Computare

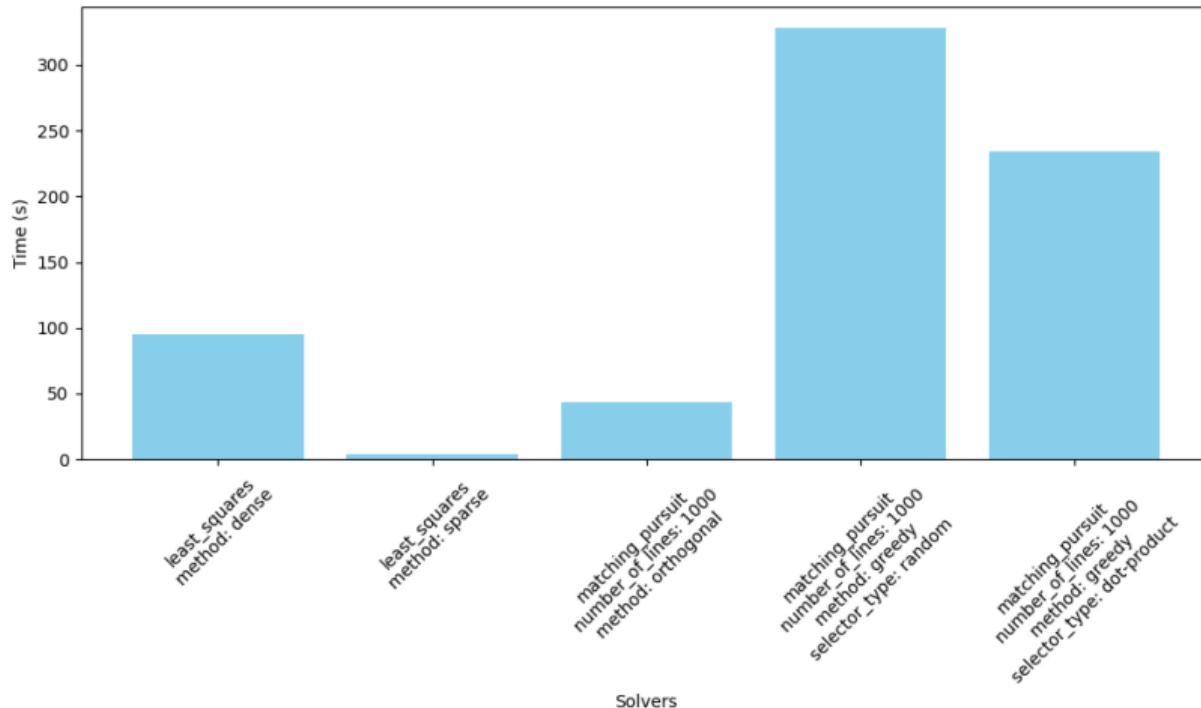


Figure: Imagine care ilustrează timpul necesar pentru computarea rezultatului în funcție de fiecare metodă.

## Timp de Computare

În ceea ce privește timpul de computare, metoda CMMP rară este cea mai eficientă, finalizându-se în doar câteva secunde. Metoda CMMP densă, deși necesită mai mult timp, rămâne relativ rapidă în comparație cu algoritmii matching pursuit.

Metodele Greedy, în special euristică random, prezintă cele mai lungi timpuri de calculare. Acest lucru se datorează faptului că acestea trebuie să calculeze repetat soluția de tip CMMP pentru fiecare linie candidat și să o selecteze pe cea care minimizează eroarea la fiecare pas.

Datorită metodei mai eficiente de selecție a liniei din cadrul abordării OMP, se observă că acest algoritm este semnificativ mai rapid decât metoda Greedy și, surprinzător, mai rapid decât CMMP dens. Cu toate acestea, OMP nu este încă la fel de rapid ca CMMP rar.

# Utilizarea Maximă a Memoriei

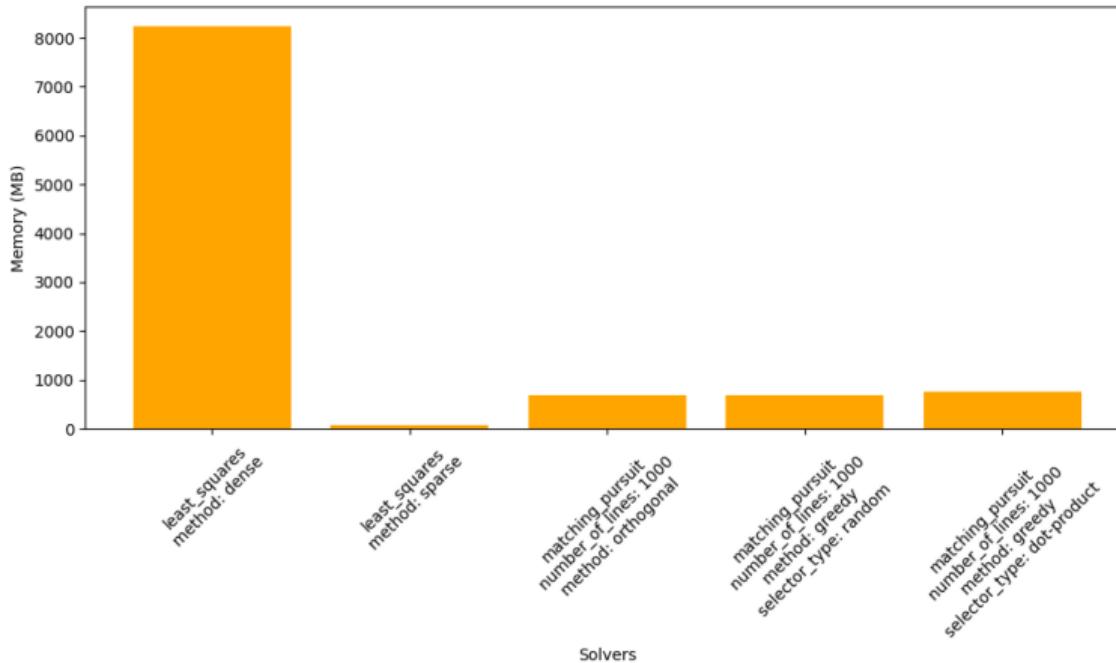


Figure: Imagine care ilustrează memoria maximă utilizată în timpul computării rezultatului în funcție de fiecare metodă.

# Utilizarea Maximă a Memoriei

Utilizarea maximă a memoriei oferă o perspectivă interesantă asupra eficienței algoritmilor. Toate metodele, cu excepția abordării CMMP densă, utilizează mai puțin de 1GB de memorie.

Metoda densă necesită mult mai multă memorie deoarece trebuie să stocheze o matrice mare de dimensiune:  $m = 108.900$ ,  $n = 4.950$ . În schimb, metoda rară trebuie doar să stocheze elementele nenule, reprezentând o soluție mai eficientă din punct de vedere al memoriei.

Iar abordările OMP și Greedy păstrează, în cel mai rău caz, o matrice de dimensiune:  $m = 108.900$ ,  $n = 1000$ . Unde 1000 e numarul de linii care trebuie desenate.

# Optimizarea String Art cu Inteligență Artificială

Având algoritmi definiți mai sus, putem genera dataset-ul nostru de imagini de intrare și imagini țintă. Acestea pot fi apoi utilizate pentru antrenarea unui model de AI care să prezică liniile necesare pentru a recrea imaginea dorită.

Output-ul poate fi generat în două moduri:

1. Prezicerea liniilor trasate

Având o configurație de cuie dispuse pe un cerc, modelul poate fi antrenat să prezică perechile de puncte care formează liniile corecte, de exemplu: (C0, C1), (C23, C49) etc.

2. Prezicerea intregii imagini

În această abordare, modelul generează direct imaginea finală desenată, eliminând necesitatea unui proces de post-procesare.

# Procesul de Generare Date

**Date de intrare:** ImageNet-Sketch<sup>5</sup>

- **50.000** imagini de tip sketch
- **1000** foldere (câte o clasă per folder), fiecare cu aproximativ **50** de imagini

**Prepararea Datelor:**

1. **Extragerea imaginilor:** procesare separată pentru fiecare folder
2. **Filtrarea rezoluțiilor:** eliminarea imaginilor cu laturi mai mici de 256px
3. **Redimensionarea imaginilor:** toate imaginile sunt scalate la 256x256 pixeli

---

<sup>5</sup>Link către Github Repository pentru ImageNet-Sketch

## Preprocesarea Imagineilor

**Problema:** Unele imagini, precum imaginea de input *Lena*, au adesea un ton estompat sau tern, rezultând într-un aspect aproape gri care afectează acuratețea detectării de contururi de către algoritmi.

**Observație:** Metodele de preprocesare nu ar trebui să depindă de o variabilă “magică” și ar trebui să asigure aceeași tonalitate pentru fiecare imagine.

**Soluție:** Am explorat două metode de preprocesare care urmăresc să standardizeze tonalitatea imaginilor. Aceste metode sunt **consistentă**: sporesc contrastul imaginilor fade și îl reduc pentru cele prea întunecate, aducând fiecare imagine la un nivel tonal similar.

# Histogram Equalization

Această metodă funcționează prin redistribuirea valorilor histogramei unei imagini pe întregul spectru de intensități, “egalizând“ astfel contrastul imaginii.

Pentru mai multe detalii, consultați tutorialul oficial OpenCV.



Figure: Diferența dintre imaginea originală *Lena* și varianta procesată cu histogram equalization.

# Grayscale Quantization

**Grayscale Quantization** reduce numărul de nivele de intensitate dintr-o imagine, transformând un spectru continuu de tonuri de gri într-unul discret, bazat pe praguri.

$$Q(x) = \lfloor \frac{x}{\Delta} \rfloor * \Delta + \frac{\Delta}{2}$$

$$\Delta = \frac{255}{L}$$

$L$  = Numărul dorit de nivele de intensitate



Figure: Diferența dintre imaginea originală *Lena* și varianta procesată grayscale quantization cu 2 nivele.

# Analiză Comparativă

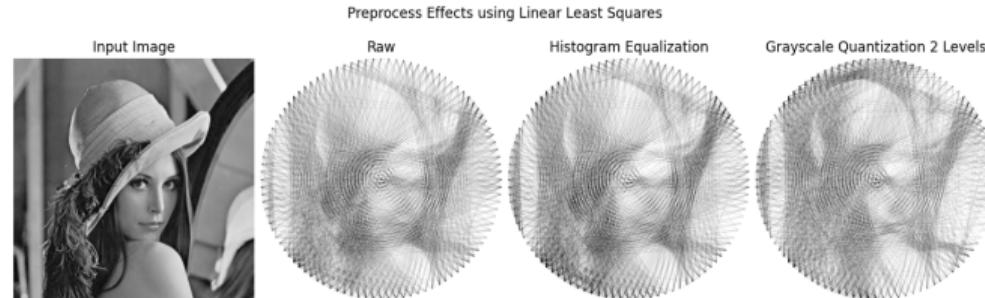
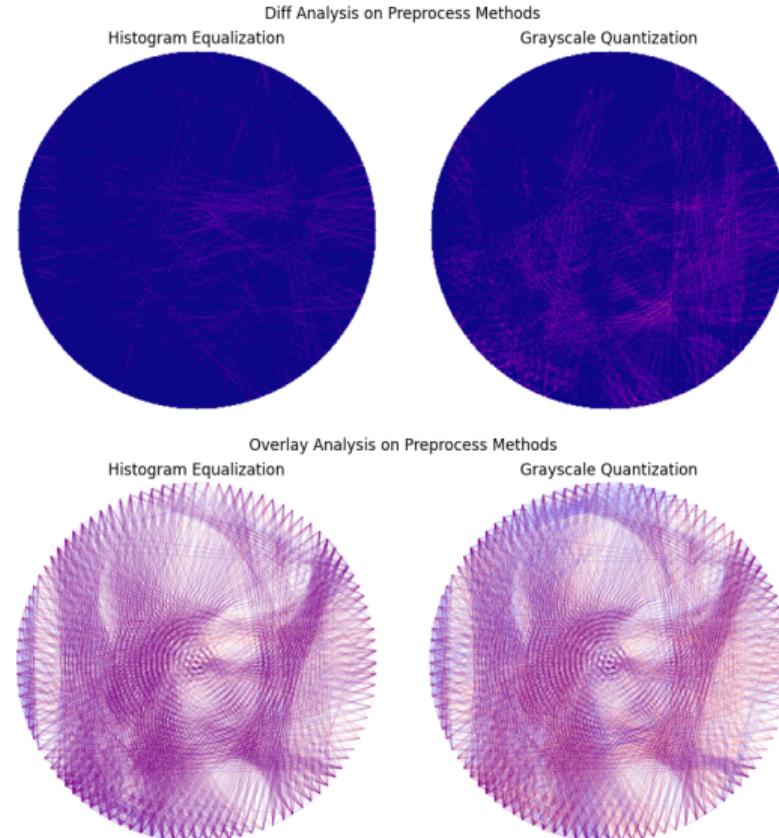


Figure: Efectul preprocesării folosind cele două metode.

**Grayscale Quantization**, fiind o metodă mai agresivă, facilitează detectarea formelor datorită naturii binare a rezultatului. În schimb, **Histogram Equalization** produce o ajustare mai subtilă, păstrând mai multe detalii din imaginea originală.

# Analiză Comparativă — Diferențe și Suprapunerি



# Computarea Label-urilor

Metodele prezentate au fost implementate in pachetul Python: Procedural Computing of String-Art<sup>6</sup>

Acest pachet Python a fost utilizat pentru generarea label-urilor asociate fiecărei imagini.

## Configurația utilizată:

- CROP\_MODE = "first-half"
- NUMBER\_OF\_PEGS = 100
- RASTERIZATION = "xiaolin-wu"
- solver = "least-squares"
- matrix\_representation = "sparse"

---

<sup>6</sup>Link către Github Repository Procedural Computing of String-Art

## Creare Datasetului

**Metodă de generare:** S-a utilizat metoda **CMMP** datorită vitezei ( 5s/imagină), dar pentru toate cele **50.000** de imagini din ImageNet-Sketch, procesul ar fi durat aproximativ **70 de ore** de compute.

**Soluție adoptată:** S-a folosit platforma Kaggle, însă limita de **12 ore compute/session** a impus crearea unui subset. Astfel a fost generat un dataset de **1000 de imagini label-uite**: *String-Art AI 1000*

**Încărcarea datelor:** Folosind un DataLoader custom, s-a realizat un split **train/val/test** de **70% / 15% / 15%**. Codul este disponibil aici: *StringArt DataLoader Example*

## U-Net - Motivare

Deși arhitectura **U-Net** a fost concepută inițial pentru sarcini de segmentare a imaginilor, s-a dovedit a fi o alegere potrivită și pentru problema *StringArt*.

Capacitatea U-Net de a genera imagini de ieșire cu aceeași dimensiune ca imaginea de intrare, împreună cu conexiunile *skip* care păstrează detalii spațiale importante, o face adecvată pentru acest tip de aplicație.

## Arhitectura

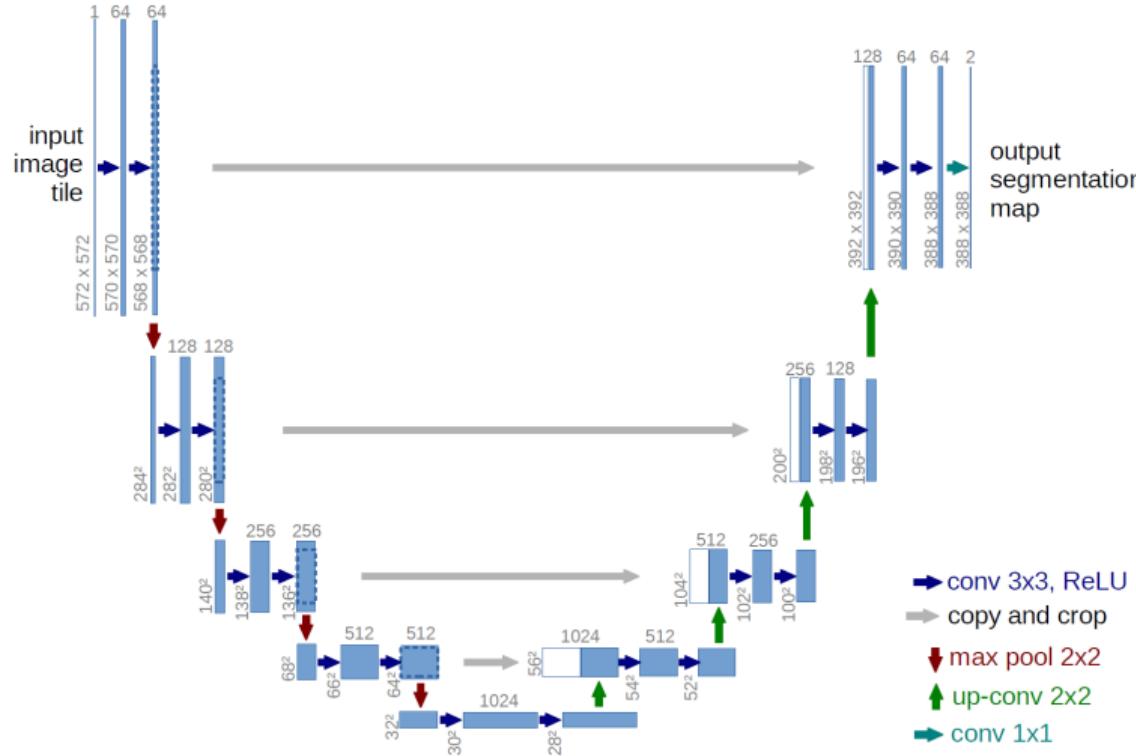


Figure: Arhitectura modelului U-Net.

Singura modificare față de designul original al U-Net constă într-un traseu conlovuțional cu profunzime redusă. În loc de configurația standard  $1024 \rightarrow 512 \rightarrow 256 \rightarrow 128 \rightarrow 64$ , s-a folosit o variantă mai ușoară:  $256 \rightarrow 128 \rightarrow 64 \rightarrow 32 \rightarrow 16$ .

Această modificare a fost necesară deoarece arhitectura originală nu generaliza eficient pe setul relativ mic de antrenare, format din doar 1000 de imagini.

Imaginile sunt scalate în intervalul [0, 1] prin împărțirea valorilor pixelilor la 255.

## **Informații despre imagini:**

- Dimensiune intrare: 256x256
- Canale: 1 (grayscale)
- Dimensiune ieșire: 256x256

# Loss Function

Înțial, s-a folosit **MSE** (Mean Squared Error) ca loss function. Totuși, aceasta nu reușea să surprindă corect tiparele structurale specifice *StringArt*, în special dispunerea cuielor și conexiunile dintre ele (liniile trasate de la un cui la altul).

Pentru a reda mai bine similitudinea structurală dintre imagini, s-a utilizat **SSIM** (*Structural Similarity Index*)<sup>7</sup>.

Pentru mai multe detalii, consultați: U-Net Loss Function

---

<sup>7</sup>Link către SSIM Index

# Training

**Optimizer** : AdamW **Learning Rate** : 1e-4

**Learning Rate Scheduler** A fost utilizat scheduler-ul ReduceLROnPlateau, configurat astfel:

- Factor: 0.5 (înjumătățește rata de învățare atunci când este declanșat)
- Patience: 10 epoci fără îmbunătățire a pierderii pe setul de validare

**Batch Size și Pași de Acumulare:** Din cauza limitărilor GPU (Tesla T4 pe Kaggle), s-a utilizat un batch size mic de 16. Pentru a simula un batch mai mare și a stabiliza antrenarea, s-a folosit acumularea gradientului cu 4 pași ( $Batch\ Simulat = 4 \times 16 = 64$ ).

## Early Stopping și Salvarea Modelului:

A fost implementată o logică personalizată pentru:

- Oprirea timpurie a antrenării dacă nu se observă îmbunătățiri timp de 20 de epoci consecutive.
- Salvarea modelului ori de câte ori se atinge o nouă valoare minimă pentru pierderea pe setul de validare.

# Rezultate

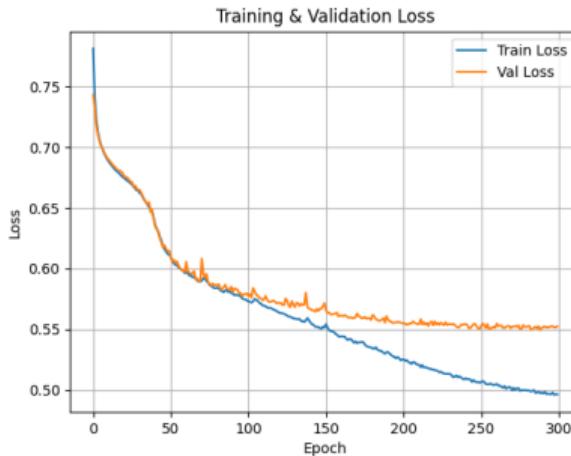


Figure: Training vs. Validation Loss.

Atât pierderea la antrenare, cât și cea la validare scad simultan, ceea ce indică o învățare sănătoasă. Totuși, pierderea pe setul de validare stagnează în jurul valorii de 0.55, în timp ce pierderea la antrenare continuă să scadă până la aproximativ 0.5.

# Rezultate

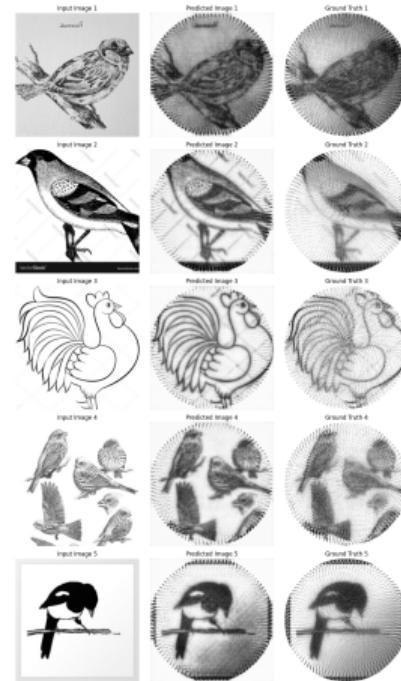


Figure: Predictions.

# Concluzii

## Puncte forte:

- Modelele circulare de cuie sunt bine replicate.
- Rezultatele generate sunt vizual apropriate de imaginile țintă (*ground truth*).

## Puncte slabe:

- Logica de trasare a liniilor (conectarea cuielor) nu este bine învățată.
- Unele cuie sunt aliniate greșit sau lipsesc complet.

GAN-urile sunt recunoscute pentru capacitatea lor de a genera imagini realiste. Pornind de la această idee, am emis ipoteza că o arhitectură GAN ar putea învăța tiparele structurale ale *StringArt* și ar putea genera ieșiri corespunzătoare pentru imaginile de intrare.

## Generator:

Generatorul este același model U-Net utilizat anterior, fără modificări sau adăugări.

## Discriminator:

Discriminatorul este o rețea complet conoluțională, formată din trei straturi de conoluție cu dimensiunile caracteristicilor 64, 128 și 256, respectiv:

### Codul modelului

```
1 self.model = nn.Sequential(  
2     nn.Conv2d(in_channels, 64, 4, 2, 1),  
3     nn.LeakyReLU(),  
4     nn.Conv2d(64, 128, 4, 2, 1),  
5     nn.BatchNorm2d(128),  
6     nn.LeakyReLU(),  
7     nn.Conv2d(128, 1, 4, 1, 1),  
8     nn.Sigmoid(),  
9 )
```

## Loss Function

Așa cum s-a observat în slide-ul 46, metrica Structural Similarity Index (SSIM) funcționează semnificativ mai bine la păstrarea caracteristicilor structurale comparativ cu funcția de pierdere bazată pe Mean Squared Error (MSE).

Funcția finală de pierdere combină:

- O pierdere Binary Cross-Entropy (BCE) între predictia discriminatorului pentru imaginile generate și eticheta lor adevărată (fals)
- Plus o pierdere SSIM care încurajează similaritatea pe pixeli între imaginea generată și imaginea de referință.

Pierdea totală poate fi exprimată astfel:

$$\text{Loss} = \text{Loss}_{\text{discriminator}} + \lambda \cdot \text{Loss}_{\text{SSIM}}$$

Unde:

- $\lambda$  este un coeficient care controlează echilibrul între realism (de a păăli discriminatorul) și acuratețe (similaritatea structurală cu imaginea etichetă).
- Valoarea lui  $\lambda$  a fost aleasă să fie 50 pentru acest setup de antrenament.

# Rezultate

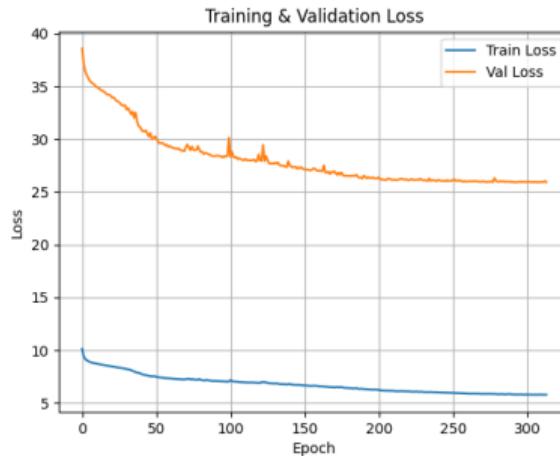


Figure: Training vs. Validation Loss.

Antrenamentul a fost oprit în jurul epocii 300 de early stopping. Atât pierderea pe setul de antrenament, cât și cea pe setul de validare prezintă o scădere constantă și corelată până la convergență.

# Rezultate

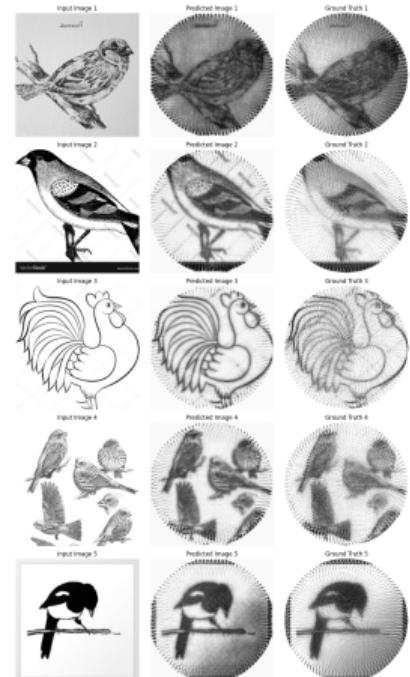
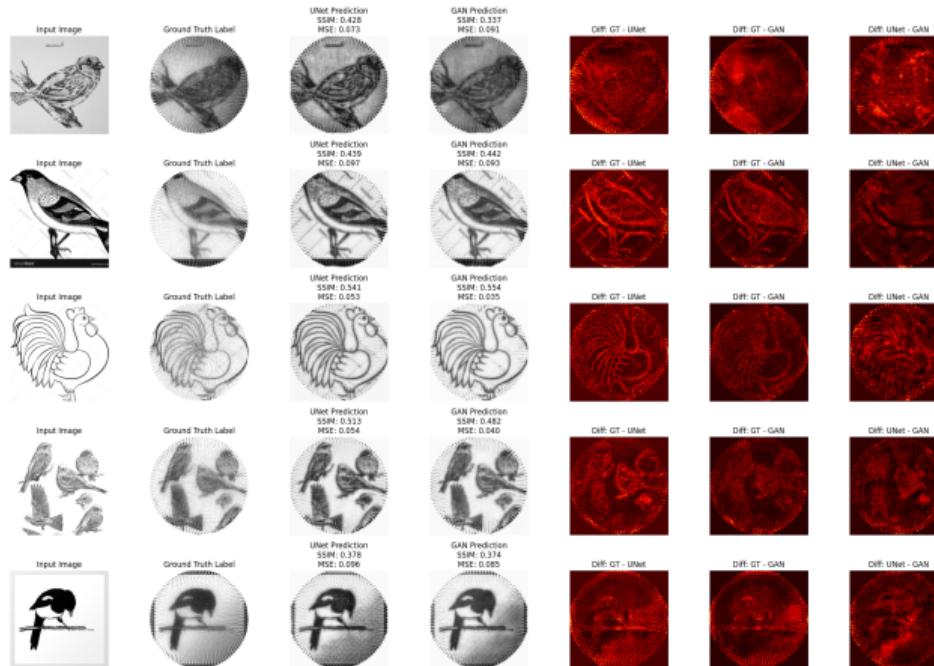


Figure: Predictions.

# Analiză Comparativă vs. U-Net



## Analiză Comparativă vs. U-Net

Din grafic observăm că valorile SSIM sunt în general similare între cele două modele, cu câteva excepții notabile. De exemplu, în prima imagine, U-Net-ul simplu depășește semnificativ GAN-ul (scoruri SSIM de 0.428 vs. 0.337).

Totuși, în ceea ce privește MSE, GAN-ul obține constant valori mai mici în majoritatea cazurilor, indicând o corespondență pixel cu pixel mai apropiată de imaginea de referință.

Aceasta este confirmată și de imaginile de diferență: ieșirile GAN-ului prezintă mai puține și mai puțin intense zone luminoase comparativ cu cele ale U-Net-ului, sugerând o acuratețe mai fină în reconstrucție.

Un alt avantaj vizibil este capacitatea GAN-ului de a reda mai bine decupările circulare din imagini, în timp ce U-Net-ul tinde să păstreze mai multe artefacte pătrate.

## Concluzii

Per ansamblu, deși predicțiile U-Net și GAN pot părea similare la un nivel general, o analiză vizuală mai atentă relevă faptul că ieșirile U-Net prezintă frecvent regiuni negre mai închise.

Așadar, deși metricele cantitative pot sugera o performanță comparabilă, analiza subiectivă indică faptul că GAN-ul este superior.

# Bibliografie

-  [csr\\_matrix.](#)  
Accessed: 2025-01-16.
-  Birsak, M., Rist, F., Wonka, P., and Musalski, P. (2018).  
String art: Towards computational fabrication of string images.  
*ResearchGate*.
-  et al., N. P. (2015).  
Image database tid2013: Peculiarities, results and perspectives.  
*ScienceDirect*, page 76.
-  Socha, R. (2024).  
Computational string art.  
*KTH*, page 5.
-  Zhang, L., Mou, X., and Zhang, D.  
Fsim: A feature similarity index for image quality assessment.  
*PolyU*.