Deepfake Classification

Mădălin Ioana, Grupa 244

9 iunie 2025

Cuprins

1 Introducere

Am avut de creat un model care sa clasifice imagini generate artificial. Datasetul pe care l-am folosit avea imagini de 100×100 pixeli, iar taskul era sa le clasificam in 5 categorii diferite (etichete de la 0 la 4, reprezentand 5 tipuri diferite de deepfake-uri).

Am incercat doua abordari complet diferite: una simpla cu K-Nearest Neighbors (KNN) si una mai avansata cu o retea neurala convolutionala cu blocuri reziduale, pentru a observa care merge mai bine si de ce.

Proiectul demonstreaza importanta alegerii arhitecturii potrivite si evidentiaza limitarile metodelor traditionale fata de abordarile moderne de deep learning.

2 Datasetul si preprocesarea datelor

Datasetul este organizat in trei seturi: antrenare, validare si test. Fiecare imagine are dimensiunea de 100×100 pixeli si poate apartine uneia dintre cele 5 categorii.

2.1 Structura datasetului

- **Setul de antrenare:** folosit pentru invatarea parametrilor modelului
- **Setul de validare:** folosit pentru evaluarea performantei in timpul antrenarii
- **Setul de test:** folosit pentru evaluarea finala si generarea predictiilor

2.2 Preprocesarea pentru KNN

Pentru abordarea KNN am facut preprocesarea simpla:

- Am incarcat imaginile in format RGB (3 canale)
- Am transformat fiecare imagine intr-un vector de dimensiune $100 \times 100 \times 3 = 30.000$
- Am normalizat valorile impartind la 255 pentru a avea pixeli in intervalul [0,1]
- Nu am aplicat augmentari pentru KNN deoarece algoritmul se bazeaza pe comparatii exacte

2.3 Preprocesarea pentru CNN

Pentru reteaua neurala convolutionala am folosit o abordare mai sofisticata:

- Am incarcat imaginile in format RGBA (4 canale) pentru a include si informatia despre transparenta
- Am aplicat normalizarea cu $\mu = 0.5$ si $\sigma = 0.5$ pentru toate cele 4 canale
- Pentru setul de antrenare am aplicat augmentari: flip orizontal aleator si rotire cu ± 10 grade
- Pentru validare si testare am folosit doar normalizarea

Decizia de a folosi canalul alpha (RGBA in loc de RGB) s-a bazat pe intuitia ca informatia despre transparenta ar putea contine semne vizuale utile pentru detectarea deepfake-urilor.

3 Prima abordare: K-Nearest Neighbors (KNN)

Am inceput cu algoritmul K-Nearest Neighbors ca baseline, fiind o metoda simpla si rapida de implementat pentru clasificare.

3.1 Principiul algoritmului

KNN clasifica o imagine noua pe baza votului majoritatii dintre cei k vecini cei mai apropiati din setul de antrenare. Pentru fiecare imagine de test:

1. Calculez distanta fata de toate imaginile din setul de antrenare
2. Selectez cei k vecini cei mai apropiati
3. Clasa finala este cea mai frecventa dintre acesti k vecini

3.2 Implementarea KNN

Am implementat algoritmul de la zero cu urmatorii pasi:

1. Incarcarea imaginii in format RGB
2. Transformarea matricei $100 \times 100 \times 3$ intr-un vector de dimensiune 30.000
3. Normalizarea valorilor la intervalul [0,1]
4. Calcularea distantei euclidiene fata de toate exemplele din antrenare

Distanta euclidiana se calculeaza ca:

$$d(x_1, x_2) = \sqrt{\sum_{i=1}^{30000} (x_{1i} - x_{2i})^2}$$

3.3 Testarea valorilor pentru k

Am experimentat cu mai multe valori pentru parametrul k:

Tabela 1: Rezultatele KNN pentru diferite valori ale lui k

k	**Acuratete (%)**	**Observatii**
3	29.8	Varianta mare, sensibil la outlieri
5	29.7	Echilibru intre bias si varianta
7	27.6	Mai stabil, dar performanta scazuta
9	29.1	Incepe sa supraaplatizeze
11	28.8	Performanta in scadere

3.4 Limitarile KNN pentru detectarea deepfake-urilor

Rezultatele slabe ale KNN (sub 30% acuratete) se datoreaza urmatoarelor limitari:

- **Reprezentarea bruta:** KNN foloseste doar valorile pixelilor, fara sa inteleaga structura imaginii

- **Aspectul dimensionalitatii:** In spatii de dimensiuni mari, toate punctele par la distante similare
- **Lipsa invariantei:** Doua deepfake-uri similare dar cu mici translatii vor fi considerate foarte diferite

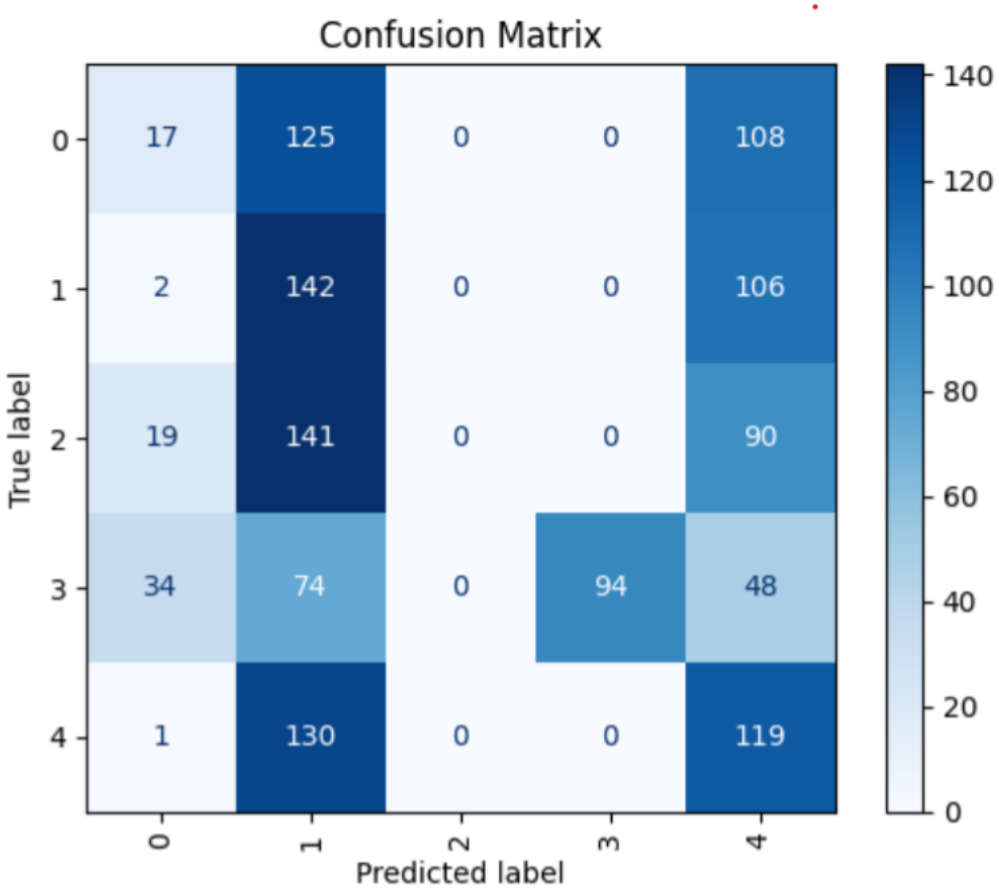

Figura 1: Matricea de confuzie pentru modelul KNN cu k=5

4 A doua abordare: CNN cu blocuri reziduale

Avand in vedere performanta slaba a KNN, am trecut la o arhitectura de retea neurala convolutionala cu blocuri reziduale.

4.1 Motivatia pentru blocurile reziduale

Blocurile reziduale rezolva problema gradient vanishing prin introducerea conexiunilor skip (shortcut connections). Aceasta permite antrenarea retelelor mai adanci si imbunatateste propagarea gradientului.

4.2 Arhitectura detaliata a modelului

Reteaua mea are urmatoarea structura:

Tabela 2: Arhitectura detaliata a CNN-ului

Strat	**Input**	**Output**	**Parametri**
Stem Conv	$4 \times 100 \times 100$	$64 \times 100 \times 100$	kernel 3×3, padding=1
Stem BN+ReLU	$64 \times 100 \times 100$	$64 \times 100 \times 100$	-
Stage 1 Block 1	$64 \times 100 \times 100$	$128 \times 50 \times 50$	stride=2
Stage 1 Block 2	$128 \times 50 \times 50$	$128 \times 50 \times 50$	stride=1
Stage 2 Block 1	$128 \times 50 \times 50$	$256 \times 25 \times 25$	stride=2
Stage 2 Block 2	$256 \times 25 \times 25$	$256 \times 25 \times 25$	stride=1
Stage 3 Block 1	$256 \times 25 \times 25$	$512 \times 12 \times 12$	stride=2
Stage 3 Block 2	$512 \times 12 \times 12$	$512 \times 12 \times 12$	stride=1
Global Avg Pool	$512 \times 12 \times 12$	$512 \times 1 \times 1$	-
Flatten	$512 \times 1 \times 1$	512	-
FC1 + ReLU	512	128	+ Dropout 0.3
FC2	128	5	clasele finale

4.3 Implementarea blocurilor reziduale

Fiecare bloc rezidual contine:

- Doua convolutii 3×3 consecutive cu BatchNorm si ReLU
- O conexiune skip care aduna inputul direct la output
- Pentru cazurile in care dimensiunile nu se potrivesc, folosesc o convolutie 1×1 pentru adaptare

Ecuatia matematica pentru un bloc rezidual este:

$$y = F(x, \{W_i\}) + x$$

unde $F(x, \{W_i\})$ reprezinta transformarile invatate, iar x este inputul original.

4.4 Avantajele arhitecturii alese

- **Propagarea gradientului:** Conexiunile skip permit gradientului sa se propage direct
- **Identitatea:** Reteaua poate invata functia identitate daca este optima
- **Eficienta:** Arhitectura este relativ compacta, dar eficienta
- **Generalizare:** BatchNorm si Dropout ajuta la regularizare

5 Augmentarea datelor si CutMix

Pentru a imbunatati generalizarea modelului, am folosit mai multe tehnici de augmentare a datelor.

5.1 Augmentari traditionale

Pentru setul de antrenare am aplicat:

- **RandomHorizontalFlip:** flip orizontal cu probabilitatea 0.5
- **RandomRotation:** rotire aleatoare cu ± 10 grade
- **Normalizare:** cu $\mu = 0.5$ si $\sigma = 0.5$ pentru toate canalele

5.2 CutMix

CutMix este o tehnica de augmentare care combina parti din doua imagini diferite. Procesul functioneaza astfel:

1. Se aleg doua imagini random din batch: (x_1, y_1) si (x_2, y_2)
2. Se genereaza un factor λ din distributia Beta(1,1)
3. Se calculeaza dimensiunile unui dreptunghi bazat pe λ
4. Se inlocuieste o portiune dreptunghiulara din prima imagine cu cealalta
5. Labelul final devine: $\lambda \cdot y_1 + (1 - \lambda) \cdot y_2$

Avantajele CutMix:

- Creste diversitatea datelor de antrenare
- Previne overfittingul prin regularizare implicita
- Imbunatateste robustetea modelului

Am aplicat CutMix in 50% din batch-uri in timpul antrenarii.

6 Antrenarea modelului CNN

Procesul de antrenare a fost optimizat prin multiple experimente pentru gasirea configuratiei optime.

6.1 Configuratia de optimizare

Tabela 3: Parametrii de antrenare

Parametru	**Valoare**	**Justificare**
Loss Function	CrossEntropyLoss	Standard pentru clasificarea pe mai multe clase
Label Smoothing	0.1	Previne overconfidence
Optimizer	AdamW	Combina Adam cu weight decay
Learning Rate	3e-4	Echilibru intre viteza si stabilitate
Weight Decay	1e-4	Regularizare L2
Batch Size	64	Optimizare GPU si stabilitate
Epoci	100	Convergenta completa fara overfitting

6.2 Strategia de learning rate

Am folosit OneCycleLR scheduler cu urmatoarele caracteristici:

- **Faza de warm-up:** 10% din antrenare cu LR crescator
- **Faza de scadere:** 90% din antrenare cu scadere cosinusoidala
- **Max LR:** 3e-4, acelasi cu LR initial

Aceasta strategie permite convergenta rapida si stabila, evitand blocarea in minime locale.

6.3 Tehnici de regularizare

Pentru prevenirea overfittingului am combinat mai multe tehnici:

- **Dropout (0.3):** in stratul de clasificare
- **Weight Decay (1e-4):** regularizare L2 in optimizer
- **Label Smoothing (0.1):** pentru prevenirea overconfidence
- **Data Augmentation:** inclusiv CutMix

7 Experimentele si tuningul hiperparametrilor

Am efectuat multiple experimente pentru optimizarea performantei modelului.

7.1 Optimizarea learning rate-ului

Tabela 4: Impactul learning rate-ului asupra performantei

Learning Rate	Acuratete Validare	Observatii
1e-4	91.2%	Convergenta lenta
3e-4	94.3%	Optim, convergenta rapida si stabila
1e-3	92.8%	Oscilatii in antrenare

7.2 Analiza dropoutului

Tabela 5: Efectul dropoutului asupra generalizarii

Dropout Rate	Acc. Antrenare	Acc. Validare	Overfitting
0.0	98.1%	91.5%	Ridicat
0.3	95.8%	94.3%	Moderat
0.5	93.2%	93.1%	Scazut, dar invatare lenta

7.3 Impactul canalului Alpha (RGBA vs RGB)

O descoperire importanta a fost superioritatea procesarii RGBA fata de RGB:

Tabela 6: Comparatia RGB vs RGBA

Format	Acuratete Validare	Imbunatatire
RGB (3 canale)	92.1%	Baseline
RGBA (4 canale)	94.3%	+2.2%

Aceasta imbunatatire sugereaza ca informatia despre transparenta contine semne vizuale valoroase pentru detectarea deepfake-urilor.

8 Rezultatele finale si analiza performantei

Comparatia finala intre cele doua abordari arata diferente dramatice de performanta.

8.1 Comparatia generala

Tabela 7: Comparatia finala intre modele

Model	Acuratete	Timp Antrenare
KNN (k=5)	29.7%	2 minute
CNN	94.3%	45 minute

8.2 Analiza pe clase

Performanta modelului variaza intre clase:

- **Clasa 0:** 92.8% recall (232/250) – detectare buna, dar cu unele confuzii cu clasa 4
- **Clasa 1:** 94.8% recall (237/250) – performanta ridicata, cateva confuzii cu clasa 4
- **Clasa 2:** 94.0% recall (235/250) – modelul detecteaza bine, dar confuziile sunt similare cu clasa 4
- **Clasa 3:** 100% recall (250/250) – cea mai buna performanta, fara nicio eroare
- **Clasa 4:** 72.8% recall (182/250) – cea mai slaba performanta, multe instante clasificate gresit ca 0, 1 sau 2

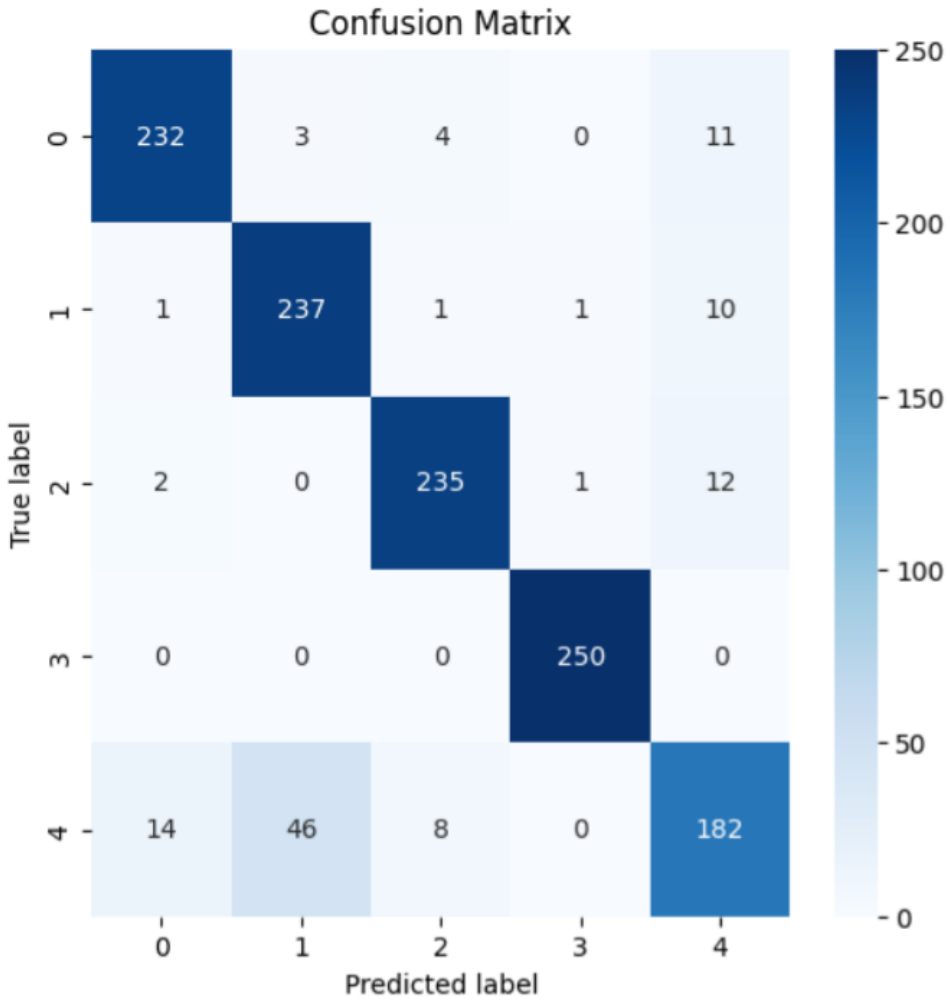

Figura 2: Matricea de confuzie pentru modelul CNN cu blocuri reziduale

9 Predictia pe setul de test si Test Time Augmentation

Pentru evaluarea finala am folosit strategii avansate de predictie.

9.1 Test Time Augmentation (TTA)

Am aplicat TTA pentru imbunatatirea predictiilor:

1. Predictie pe imaginea originala
2. Predictie pe imaginea cu flip orizontal
3. Media probabilitatilor (dupa aplicarea softmax)

Listing 1: Pseudocod pentru TTA

```
pentru fiecare imagine_test:
    # Predictii multiple
    prob1 = softmax(model(imagine_originala))
    prob2 = softmax(model(flip_orizontal(imagine_originala)))

    # Media probabilitatilor
    prob_finala = (prob1 + prob2) / 2

    # Predictia finala
    clasa_prezisa = argmax(prob_finala)
```

TTA a adus o imbunatatire de aproximativ 0.3-0.5% in acuratetea finala.

10 Concluzii

Acest proiect a demonstrat clar superioritatea metodelor moderne de deep learning fata de algoritmii traditionali pentru taskurile complexe.

10.1 Aspecte importante

1. **Demonstrarea eficacitatii CNN-urilor:** o diferenta de peste 64% in acuratete fata de KNN
2. **Importanta canalului Alpha:** o imbunatatire de 2.2% prin folosirea RGBA
3. **Optimizarea hiperparametrilor:** gasirea configuratiei optime prin multiple experimente
4. **Implementarea tehnicilor moderne:** CutMix, Label Smoothing, Test Time Augmentation

10.2 Impactul practic

Realizarea proiectului m-a invatat ca alegerea modelului potrivit face o diferenta enorma. KNN e prea simplu pentru taskuri vizuale complexe, in timp ce CNN-urile pot sa invete reprezentari utile. In general, sunt multumit de rezultate. Am invatat mult despre preprocesarea datelor, arhitecturile CNN si tehnicile de regularizare. Diferenta de performanta intre KNN si CNN arata clar de ce deep learningul a devenit atat de popular in computer vision.