Deepfake Image Classification

Roxana Asavei

Iunie 2025

1 Introducere

Obiectivul competiției a fost clasificarea imaginilor generate de diverse modele de inteligență artificială în 5 clase. Clasele nu au o semnificație aparte, fiind numerotate de la 0 la 4.

2 Analiza datelor

Pentru început, am analizat datele pentru a încerca să înțeleg distribuția acestora. Majoritatea imaginilor surprind animale facând diverse activități (jucând șah, mergând pe bicicletă etc.). Totuși, după cum am menționat mai sus, clasele nu au o semnificație anume, astfel încât gruparea imaginilor în clase nu se realizează pe un criteriu ușor identificabil, precum activitatea realizată sau animalul din imagine. În final, am ajuns la concluzia că fiecare clasă corespunde unui anumit model care a generat imaginile.

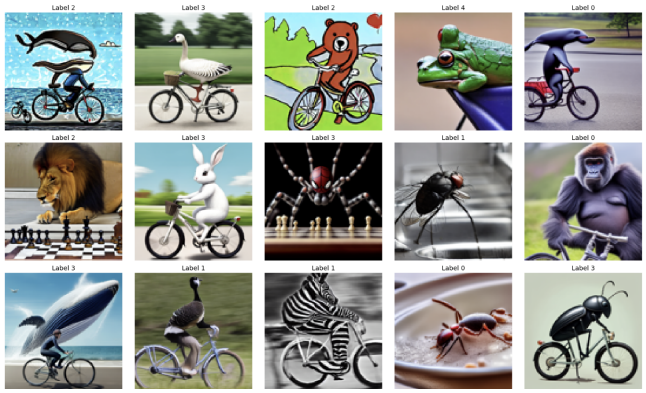
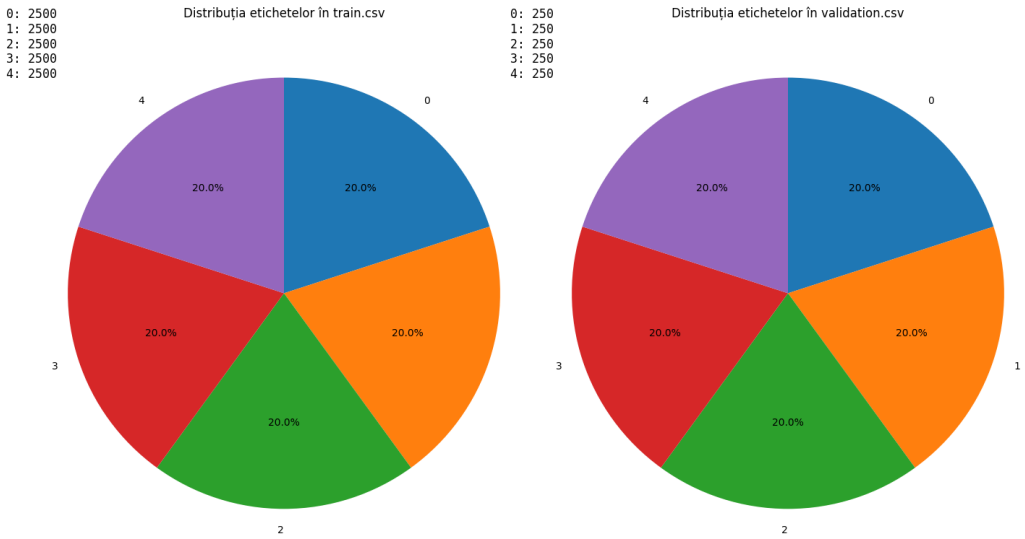


Figure 1: Imagini din training set

Ulterior, am analizat distribuția imaginilor în clase, atât în cazul setului de training, cât și în cazul celui de validare. Am făcut acest lucru pentru a verifica dacă cumva există o situație de class imbalance, ce ar fi dus la un bias

al modelului față de clasa dominantă. Din fericire, nu a fost cazul, deoarece imaginile au fost distribuite uniform în clase. Din cele 12500 de imagini de train, fiecare clasă a avut câte 2500. Similar, din cele 1250 de imagini de validare, fiecărei clase i-au corespuns 250.



(a) Distribuția setului de training (b) Distribuția setului de validare

Figure 2: Distribuția imaginilor pe clase în seturile de training și validare

3 Abordări testate

3.1 KNN

Primul model încercat a fost K-Nearest Neighbors (KNN) care reține pentru fiecare exemplu din train perechea (X, y), unde X - vectorul de features, iar y este labelul asociat. Atunci când primește o nouă imagine neetichetată, modelul va calcula distanța (Manhattan, Euclidiană) de la imaginea curentă până la toate celelalte imagini din train. Va alege cele mai apropiate K imagini, iar cum problema este una de clasificare, imaginea neetichetată va primi labelul majoritar din cadrul celor K vecini.

- **Features extraction**
  Pentru extragerea trăsăturilor am folosit ***histograme de culori***, cu câte 3 canale (R, G, B). Pentru fiecare imagine am obținut o histogramă de 512 elemente (8 x 8 x 8), pe care ulterior am normalizat-o și am transformat-o într-un vector cu o singură dimensiune, folosind *flatten*.

- **Alegerea parametrilor**
  Pentru a observa modificarea acurateții în raport cu parametrii modelului, am încercat mai multe valori pentru numărul de vecini la care ne raportăm, dar și două tipuri de distanță: distanța Euclidiană și distanța Manhanttan. Cea mai bună acuratețe pe datele de validare se obtine pentru k = 7 și folosind distanța l1. Evoluția acurateții în funcție de acești parametri poate fi analizată în graficele de mai jos:

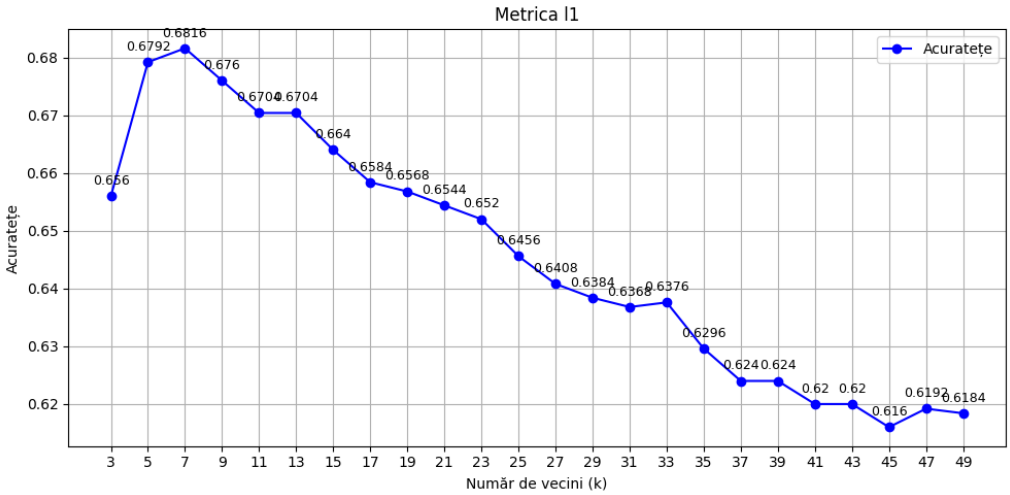


Figure 3: Evoluția acurateții pentru distanța Manhattan

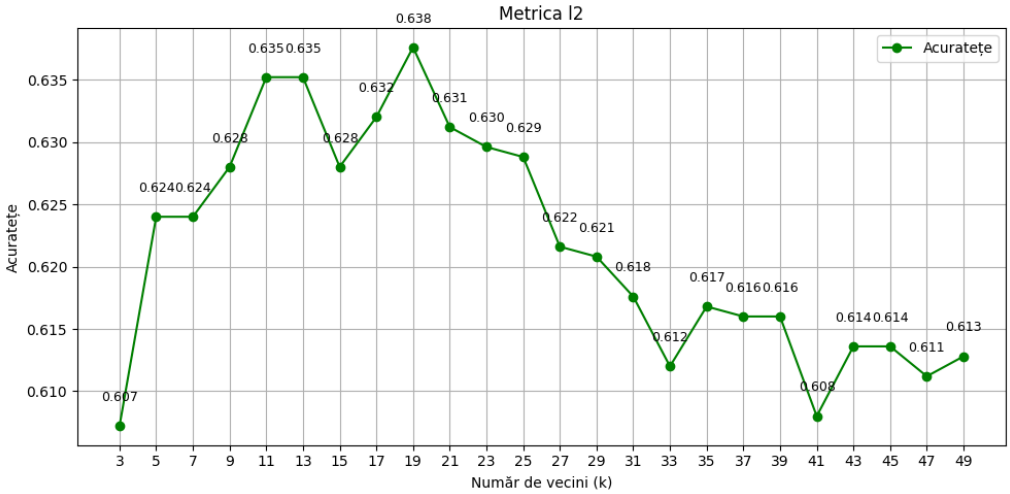


Figure 4: Evoluția acurateții pentru distanța Euclidiană

- **Matricea de confuzie**
  În cele ce urmează, vom analiza matricea de confuzie pentru parametrii pentru care s-a obținut cea mai bună acuratețe pe datele de validare: **k = 7** și **distanța l1**.

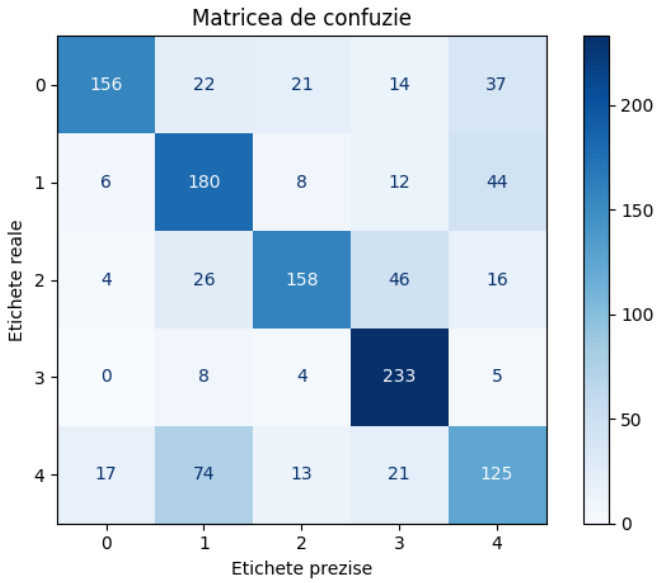


Figure 5: Matricea de confuzie pentru k = 7 și distanța l1

Se poate observa că cele mai mari probleme ale modelului sunt la clasificarea imaginilor din clasa 4, întrucât acolo are cele mai puține etichete prezise corect: 125 / 250. Deci, strict pe clasa 4, modelul are o acuratețe de doar 50%. De asemenea, este vizibilă confuzia modelului între clasele 1 și 4: 29.6% (74 / 250) din imaginile ce aparțin clasei 4 au fost clasificate ca făcând parte din clasa 1.

3.2 CNN

După mai multe încercări cu KNN, am decis trecerea la o rețea neuronală convoluțională.

- **Arhitectura modelelor**
  Pentru realizarea modelelor am folosit 2 librării: **Keras** și **PyTorch**, pentru a vedea dacă există diferențe semnificative de acuratețe. Totuși, păstrând o arhitectură relativ similară în materie de tipurile de straturi, nu am înregistrat diferențe foarte mari.

  Atât la modelele cu Keras, cât și la cele cu Pytorch am folosit următoarele tipuri de layere:

- **Convoluțional:** pentru a aplica asupra imaginilor diverse filtre, rezultând astfel mai multe features maps de pe care va învăța modelul.
- **MaxPooling:** pentru a reduce dimensiunea unui feature map de la un bloc convoluțional la altul. Cel mai frecvent am folosit kernel = (2, 2). Astfel, dintr-o regiune de 2 x 2 va păstra doar valoarea maximă.
- **AveragePooling:** la fel ca MaxPooling, doar că în loc de maxim, face media valorilor din patch-ul curent. Am ales să folosesc acest layer în ultimul bloc de Conv pentru ca valorile tuturor caracteristicilor să influențeze rezultatul.
- **BatchNormalization:** pentru a normaliza rezultatele și a crește viteza de convergență.
- **GlobalAveragePooling:** pentru fiecare feature map calculează media valorilor.
- **Flatten:** pentru a reduce inputul la o singură dimensiune pentru a putea fi procesat de straturile dense.
- **Dense:** straturi complete de neuroni, care învață pe baza caracteristicilor extrase de blocurile convulaționale.
- **Dropout:** pentru a reduce overfittingul și pentru a crește capacitatea modelului de generalizare, am ales să adaug Dropout, atât între straturile dense, de la final, dar am experimentat și cu Dropout între blocurile convoluționale.

Ca funcții de activare, am folosit:

- **ReLU:** $ReLU(x) = \max(0, x)$
- **Softmax:** $softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$

- **Modelul cu Keras**

  - **Features extraction**
    Pentru că s-au dovedit a fi eficiente, am păstrat histogramele de culori. Astfel, fiecare poză a fost reprezentată sub forma unui obiect de dimensiune 3 x 224 x 224, câte o matrice de 224 x 224 pentru fiecare canal R, G, B. Normalizarea imaginilor s-a realizat prin împărțirea acestora la 255.
  - **Structura**
    După ce am experimentat cu mai multe modele, am ajuns la concluzia că cel mai potrivit pentru acest task este un model cu 3 blocuri *Conv2D - MaxPooling2D - BatchNormalization*, urmat de un bloc *Conv2D - AveragePooling2D - BatchNormalization*, cu GlobalAveragePooling și un strat dens. Astfel, am ajuns la un model cu următoarea structură:

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 222, 222, 32)	896
max_pooling2d (MaxPooling2D)	(None, 111, 111, 32)	0
batch_normalization (BatchNormalization)	(None, 111, 111, 32)	128
conv2d_1 (Conv2D)	(None, 109, 109, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 54, 54, 64)	0
batch_normalization_1 (BatchNormalization)	(None, 54, 54, 64)	256
conv2d_2 (Conv2D)	(None, 52, 52, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 26, 26, 128)	0
batch_normalization_2 (BatchNormalization)	(None, 26, 26, 128)	512
conv2d_3 (Conv2D)	(None, 24, 24, 256)	295,168
average_pooling2d (AveragePooling2D)	(None, 12, 12, 256)	0
batch_normalization_3 (BatchNormalization)	(None, 12, 12, 256)	1,024
global_average_pooling2d (GlobalAveragePooling2D)	(None, 256)	0
flatten (Flatten)	(None, 256)	0
dense (Dense)	(None, 512)	131,584
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 5)	2,565

Figure 6: Structura modelului

Ca optimizer, am încercat atât Adam, cât și SGD, cu un learning rate de 0.01 și un momentum de 0.9. Cu structura descrisă mai sus, în combinație cu Adam am reușit să ating o acuratețe de 90.24% pe validare, în timp ce SGD a atins doar 88.72%. Din acest motiv, am ales să rămân la Adam.

În primele epoci (30) de antrenament nu am folosit data augumentation, iar acuratețea pe datele de train a crescut rapid. Totuși, și acuratețea pe validarea a înregistrat salturi seminificative.

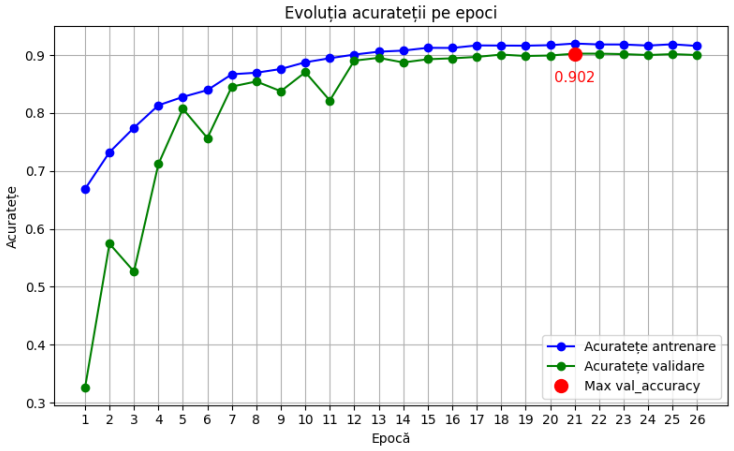


Figure 7: Evoluția acurateții în primele 30 de epoci

Ulterior, am continuat antrenarea acestui model aplicând diferite augumentări pe datele de train: rotații cu maxim 15 grade, shiftări oe lungime sau lățime cu 10%, zoom cu 10%, flip orizontal. De asemenea, am experimentat cu diferite batch size-uri: inițial am încercat cu 32, iar ulterior am testat și 16, dar și 64. În cazul modelului meu, batch_size = 32 s-a dovedit a fi cel mai potrivit.

După mai multe sesiuni de antrenament, în care am modificat batch size-ul, learning rate-ul optimizerului, callback-ul de ReduceLROnPlateau, parametrii de la data augmentation, am ajuns la un model care oferă pe datele de test o acuratețe de 91.52%. În plus, pe datele de validare, matricea de confuzie a modelului arată astfel:

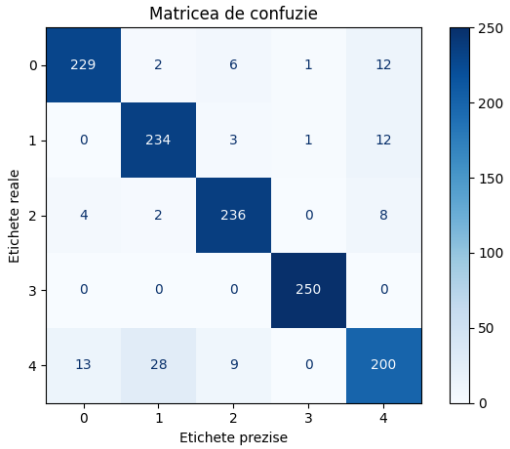


Figure 8: Evoluția acurateții în primele 30 de epoci

La fel ca la KNN, se observă o confuzie destul de mare a modelului atunci când vine vorba de imaginile cu labelul 4, ce sunt clasificate ca având labelul 1.