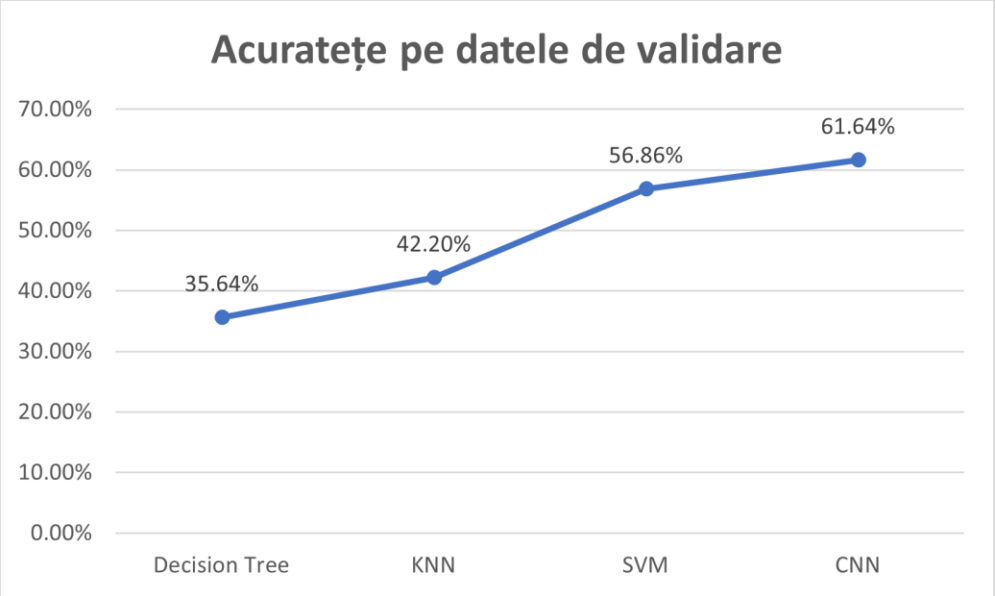
**Toader Vlad-Marian**
**Grupa 242**

# Proiect Inteligență Artificială - Deep Hallucination Classification

Pentru problema de clasificare dată, am încercat să creez patru modele de clasificare, și anume Decision Tree, KNN, SVM și CNN, obținând astfel rezultate din ce în ce mai bune pe datele de validare. Toate modelele și procedeele de procesare a datelor sunt descrise mai jos. Următorul grafic reprezintă evoluția acurateței obținută pe datele de validare, schimbând modelul folosit.




## Procesarea seturilor de date

Pentru a putea citi seturile de date de antrenare, validare și testare, am folosit funcția io.imread, găsită în pachetul skimage. În funcție de modelul ales (detaliate mai jos), am salvat trăsăturile fiecărei poze într-un nparray unidimensional, respectiv de dimensiune 16x16x3.

Astfel, din fiecare imagine obținem un vector (de dimensiuni 768 sau 16x16x3, în funcție de reprezentarea aleasă) cu valorile pixelilor (valori din intervalul (0, 255)). În continuare, am creat

nparray-uri pentru imaginile de antrenare (trainImages), etichetele acestora (trainLabels), pentru imaginile de validare (validationImages) și etichetele acestora (validationLabels), și pentru imaginile de testare (testImages) și numele acestora (imagesName - folosite pentru a crea fișierul de output).

## Normalizarea datelor

În funcție de modelul folosit, normalizarea datelor a fost realizată astfel:

- Pentru clasificatorii de tip Decision Tree, KNN și SVM, normalizarea a fost una standard, și anume am calculat media atributelor imaginilor de antrenare și deviația standard a acestora. Standardizarea este realizată scăzând media și împărțind la deviație fiecare nparray de imagini (de antrenare, validare și testare).
- Pentru clasificatorul de tip CNN, normalizarea a fost realizată împărțind valorile pixelilor la 255, pentru toți cei trei vectori de imagini. Astfel, pixelii vor lua acum valori în intervalul (0, 1), ceea ce conduce la o mai bună acuratețe, deoarece este un interval mai restrâns decât (0, 255).

Importanța normalizării se poate deduce din datele din tabelul de mai jos, în care se observă diferența dintre acuratețea obținută pe datele de validare atunci când se folosește normalizarea și atunci când datele sunt în formă inițială, în cazul clasificatorilor SVM și CNN.

| **Clasificator** | **Acuratețea fără a folosi normalizarea** | **Acuratețea folosind normalizarea** |
|---|---|---|
| SVM | 0.5601023017902813 | 0.5686274509803921 |
| CNN | 0.6129582524299622 | 0.616368293762207 |

## Clasificatori

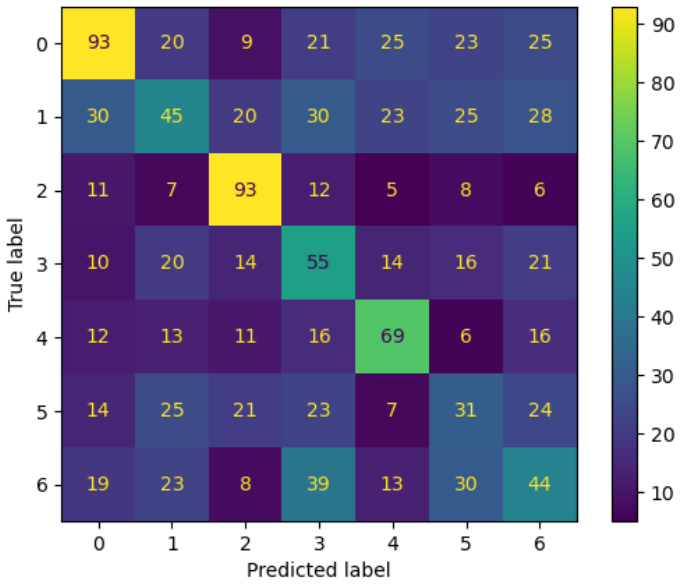
Pentru această problemă de clasificare, am folosit patru modele, cu rezultate din ce în ce mai bune: Decision Tree, KNN, SVM și CNN.

## 1. Decision Tree

Clasificatorii de tip Decision Tree sunt un tip de model care prezic valoarea învățând reguli simple, bazate pe atributele din seturile de date de antrenare.

Astfel, modelul pe care l-am folosit a fost importat din biblioteca sklearn.tree, cu parametrii default. De asemenea, seturile de date au fost standardizate înainte de a fi transmise clasificatorului.

Pe setul de date de validare am obținut o acuratețe de 0.35635123614663256, iar matricea de confuzie arăta astfel:



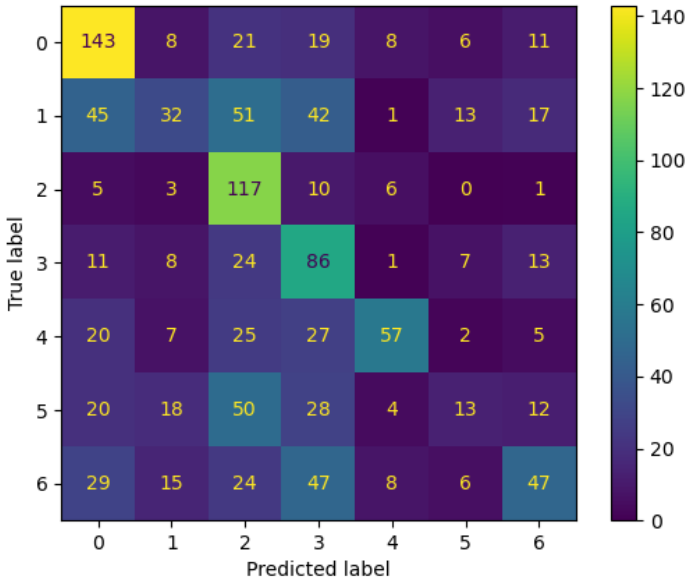

Aceste rezultate nu m-au mulțumit, așa că am trecut la următorul clasificator.

## 2. K Nearest Neighbors (KNN)

Clasificatorii de tip KNN funcționează după următorul principiu: având o valoare nouă, se iau în considerare vecinii cu o valoare cât mai apropiată de aceasta, în funcție de distanța definită. Normalizarea este importantă în cadrul acestui clasificator, deoarece acesta este bazat pe distanțe.

Modelul folosit a fost importat din biblioteca sklearn.neighbors, cu parametrii default. Datele au fost normalizate folosind metoda standard.

Folosind acest model, am obținut pe datele de validare o acuratețe de 0.42199488491048, cu următoarea matrice de confuzie:

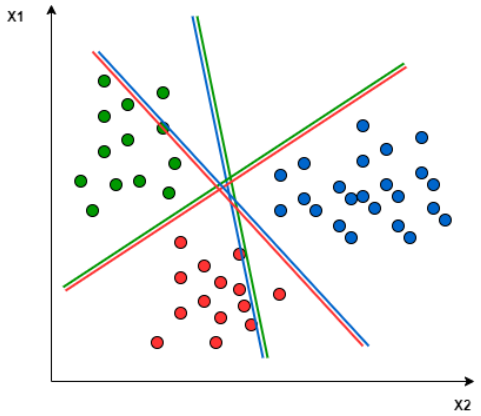


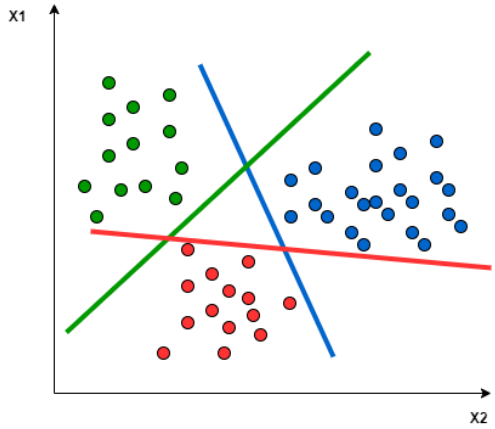

Rezultatele arată un progres față de cele obținute folosind Decision Tree, însă am decis să merg mai departe și să folosesc alți clasificatori.

## 3. Support-vector machine (SVM)

Clasificatorii de tip SVM funcționează astfel: fiecare valoare din setul de date este reprezentată ca un punct într-un spațiu n-dimensional (unde n = numărul de atribute) și scopul este de a găsi hiperplanul care să împartă aceste puncte, astfel încât acesta să aibă margine maximă (distanța maximă între punctele din clase).

În cadrul problemelor de clasificare multiplă, există mai multe abordări, și anume: One-to-One, în care se calculează hiperplane între oricare două clase, și One-to-Rest, în care se calculează hiperplane între fiecare clasă și celelalte.



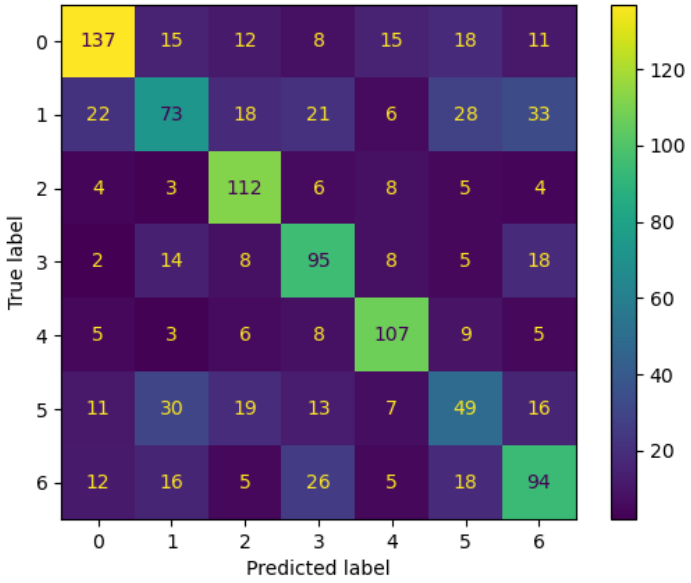

***One-to-One*** ***One-to-Rest***

Implementând acest model din biblioteca sklearn, am decis să folosesc clasificatorul SVC, cu abordarea One-to-One. Parametrii (C - parametrul de penalitate pentru eroare și kernel - tipul de kernel folosit) au fost stabiliți în urma calculării acurateței pe datele de validare și alegerea celor optimi, după cum urmează:

| **Kernel** | **C** | **Acuratețe** |
|---|---|---|
| linear | 1 | 0.45012787723785164 |
| linear | 3 | 0.4381926683716965 |
| rbf | 3 | 0.5601023017902813 |
| rbf | 5 | 0.5686274509803921 |
| rbf | 7 | 0.5575447570332481 |
| poly | 3 | 0.5012787723785166 |

Astfel, am decis să aleg implementarea clasificatorului cu C=5 și kernel rbf. Seturile de date au fost normalizate folosind metoda standard.

Pe datele de validare, am obținut o acuratețe de 0.5686274509803921, cu următoarea matrice de confuzie:

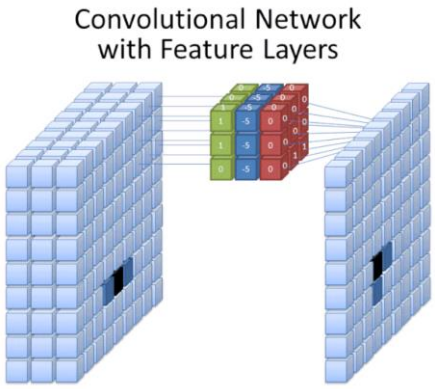



De asemenea, am făcut predicții și pe datele de testare și am creat fișierul de output pentru submisie.

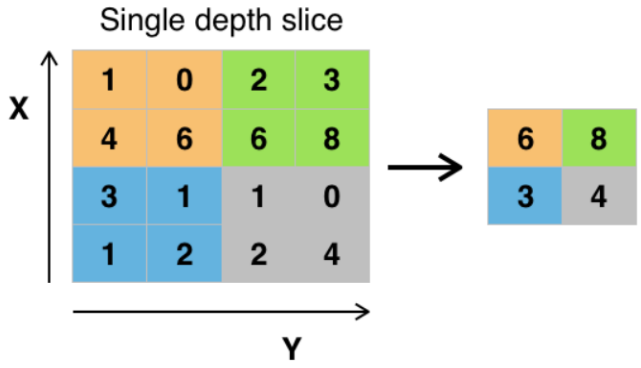
Deși rezultatele au fost cele mai bune obținute până acum, am decis să mai încerc un model, cu care am obținut rezultate și mai satisfăcătoare.

## 4. Convolutional neural network (CNN)

Rețelele neuronale convoluționale sunt rețele neuronale artificiale care folosesc straturi convoluționale, care extrag trăsături din imagini, aplicând o serie de filtre și creând o serie de hărți de atribute.



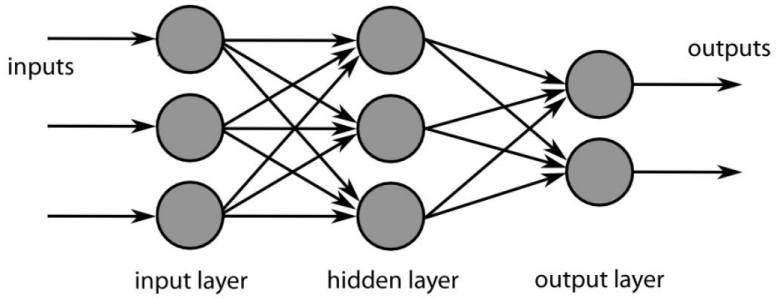
După ce hărțile atributelor au fost create, acestea sunt trecute printr-o funcție de activare, care ajută rețeaua să învețe trăsături non-lineare. Apoi, există un strat de pooling, care are rolul de a comprima imaginea, făcând-o mai mică, abstractizând astfel părțile neimportante din imagine.




Următoarele tipuri de straturi neuronale folosite sunt cele de dropout, care au rolul de a preveni overfitting-ul, renunțând astfel la o parte din conexiuni dintre straturi.

Straturile de batch normalization au rolul de a normaliza intrările de la o serie de straturi la altele.

Ultimele straturi ale CNN sunt straturile conectate complet, care primesc o serie de intrări și au rolul de a le combina în alte atribute care vor ajuta la clasificare.




Ultimul strat conține 7 neuroni (unul pentru fiecare clasă), și are rolul de a face clasificarea finală, având funcția de activare softmax, care are rolul de a selecta neuronul cu cea mai mare probabilitate.

Pentru a implementa acest tip de clasificator, m-am folosit de librăria open-source Tensorflow. Seturile de date au fost normalizate împărțind fiecare atribut (cu valori în intervalul

(0, 255)) la 255, obținând astfel un nou interval (0, 1), mult mai restrâns decât primul, ceea ce conduce la o acuratețe mai mare.

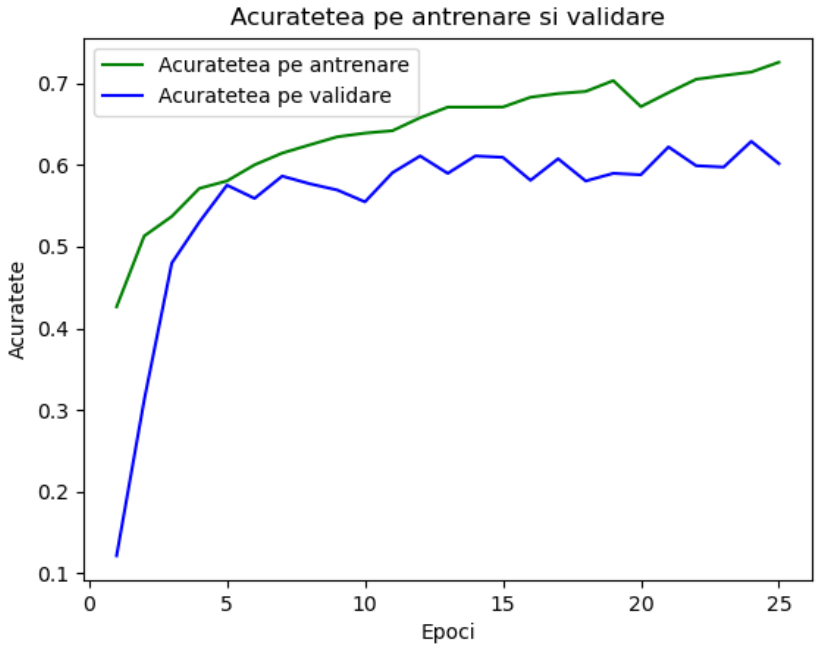
Tot pentru o mai bună acuratețe, am transformat etichetele imaginilor de antrenare și validare în vectori, folosind procedeul de one-hot encoding, unde fiecare etichetă este transformată într-un vector de dimensiune 7 (deoarece sunt 7 clase), în care maximul se găsește pe poziția corespunzătoare clasei din care face parte fiecare imagine.

După mai multe încercări de configurări alemodelului, atât din punct de vedere al numărului de straturi, cât și al tipului acestora, am ajuns la o configurație care a dat rezultate satisfăcătoare, și anume:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 16, 16, 32)        896
_________________________________________________________________
dropout (Dropout)            (None, 16, 16, 32)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 16, 16, 32)        128
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 8, 8, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 8, 8, 64)          0
_________________________________________________________________
batch_normalization_1 (Batch (None, 8, 8, 64)          256
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
dropout_2 (Dropout)          (None, 8, 8, 128)         0
_________________________________________________________________
batch_normalization_2 (Batch (None, 8, 8, 128)         512
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dropout_3 (Dropout)          (None, 8192)              0
_________________________________________________________________
dense (Dense)                (None, 32)                262176
_________________________________________________________________
dropout_4 (Dropout)          (None, 32)                0
_________________________________________________________________
batch_normalization_3 (Batch (None, 32)                128
_________________________________________________________________
dense_1 (Dense)              (None, 7)                 231
=================================================================
Total params: 356,679
Trainable params: 356,167
Non-trainable params: 512
```

Pentru compilarea modelului, am folosit optimizatorul Adam, și funcția de pierdere Sparse Categorical Crossentropy, deoarece problema dată este o problemă de clasificare multiplă.

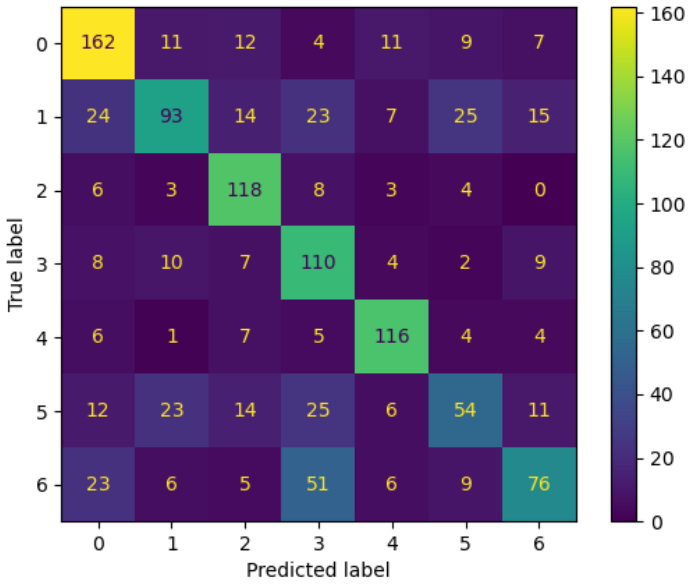
Am ales să antrenez modelul pe datele de antrenare pe parcursul a 25 de epoci, iar evoluția acurateței pe datele de antrenare și testare se poate observa în graficul următor:




Predicțiile întoarse de acest model atât pe datele de validare, cât și pe cele de testare, sunt întoarse sub forma de one-hot vectors, astfel că am fost nevoit să reconstruiesc etichetele sub formă de numere, cu ajutorul funcției argmax din numpy.

Am creat de asemenea și fișierul de output pentru submisie, folosindu-mă și de numele imaginilor de testare reținute la început.

Folosind acest clasificator, am obținut o acuratețe de 0.616368293762207 pe datele de validare, cu următoarea matrice de confuzie:




Astfel, modelul pentru care am obținut cea mai bună acuratețe, atât pe datele de validare, cât și pe 25% din datele de testare (conform Kaggle) este CNN.