

Deep Hallucination Classification

A. Descrierea proiectului

Tema proiectului a fost clasificarea de imagini color de dimensiune 64x64 pixeli într-o categorie de halucinație, din cele 96 de categorii posibile.

B. Setul de date

- 12.000 poze de antrenare și etichetele asociate;
- 1.000 poze de validare și etichetele asociate;
- 5.000 poze de test.

C. Modele folosite

În cadrul dezvoltării unui model, am încercat mai mulți clasificatori studiați la curs. Toate modelele create și rezultatele acestora pot fi vizualizate [aici](#). În continuare, o să prezint strategia folosită în dezvoltarea unui **SVM** (Mașini cu vectori suport), respectiv **CNN** (Rețele Convoluționale).

D. SVM - Mașini cu vectori suport

O primă abordare care s-a dovedit să dea rezultate bune a fost utilizarea clasificatorului **SVM** (Mașini cu vectori suport). Acesta este un algoritm de învățare automată supravegheat ce are la bază o tehnică de clasificare binară prin găsirea unui hiperplan optim într-un spațiu n-dimensional. Pentru implementarea algoritmului, am folosit din **librăria sklearn** implementarea **SVC** (C-Support Vector Classification), ce are o abordare one-vs-one iar hiperplanul de clasificare găsit este unul liniar.

Citirea imaginilor s-a realizat folosind modulul **PIL**, alături de metoda **numpy.asarray()** pentru a le converti la arrays **numpy** de dimensiune 64x64x3. Apoi, folosind metoda **numpy.ndarray.flatten()** transformăm pozele în array de **dimensiune 12288**. De asemenea, etichetele, sunt salvate într-un array.

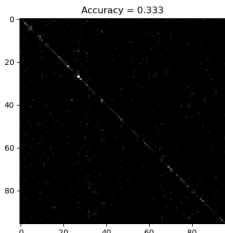
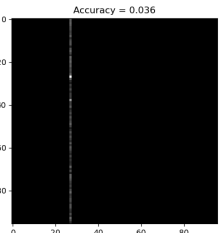
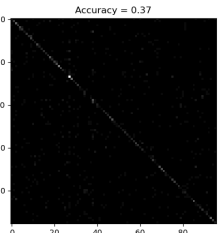
O abordare inițială pentru a normaliza datele, care s-a dovedit a fi eficientă și în cazul altor algoritmi, a fost împărțirea valorilor pozelor (din array) la 255 pentru a avea valori cuprinse în intervalul [0, 1].

Hiperparametrii ajustați pentru acest model au fost:

- ❖ Parametrul de regularizare **C**: stabilește gradul de clasificare greșită a modelului. O valoare mai mare duce la o clasificare mai corectă, cu posibilitate de overfitting. Am încercat mai multe valori pentru acest parametru, până am ajuns la o valoare convenabilă. Alături de un **kernel='linear'**, am obținut o acuratețe de 0.28 cu **C=5**, și 0.284 cu **C=50**.

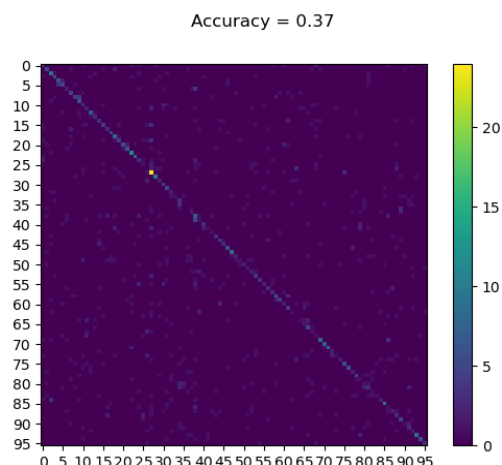
- ❖ Parametrul **kernel**: stabilește funcția kernel folosită. Rolul unei funcții kernel este de a transpune imaginile într-un spațiu separabil. Alături de hiperparametrii $\gamma=0.1$ și $C=2$, am obținut o acuratețe de 0.033 cu un kernel Sigmoid, 0.036 cu un kernel Gaussian radial basis function (RBF) și 0.333 cu un kernel Polynomial.
- ❖ Parametrul **gamma**: a fost folosit în modelele RBF și stabilește curbura dreptei de separare. În testele făcute pe modelele RBF, cea mai bună valoare pentru gamma este 1.

După ce am ajuns la niște hiperparametri convenabili ($C=100$, $\text{kernel}='poly'$), am încercat mai multe metode de **normalizare a datelor**. Prezint mai jos comparația dintre împărțirea datelor la 255, normalizarea L1 și L2.

Împărțire la 255	L1	L2
		
acuratețe=0.333	acuratețe=0.036	acuratețe=0.37

În final, **cele mai bune rezultate** le-am obținut cu o normalizare L2 și hiperparametrii ($C=100$, $\text{kernel}='poly'$), iar tipul de rulare era de aproximativ 40 de minute. Acestea sunt **rezultatele obținute**:

Acuratețea	Precizie	Recall	F1-Score
0.37	0.37	0.37	0.37



Matricea de confuzie - SVM

E. CNN - Rețele Convoluționale

O altă abordare a fost utilizarea unui **CNN** (Rețele Convoluționale). Aceasta este o rețea neurală (convoluțională) folosită în special pentru clasificarea imaginilor. Folosindu-se de straturi de convoluție și funcții de activare, un CNN reușește să recunoască tipare în imaginile date. Pentru implementarea algoritmului am folosit **librăria tensorflow**.

Citirea imaginilor s-a realizat folosind modulul PIL, alături de metoda `numpy.asarray()` pentru a le converti la arrays numpy de **dimensiune 64x64x3**. De asemenea, etichetele, sunt salvate într-un array.

Pentru a obține o acuratețe mai bună, am încercat să folosesc o **augmentare de date** folosind `ImageDataGenerator`. Acesta s-a dovedit a fi mai **ineficient**, atât ca timp de execuție cât și ca acuratețe. Astfel, pentru a preprocesa imaginile, pe lângă imaginile de antrenare, am folosit și un set de **imagini rotite la 30 de grade și cu flip orizontal** obținut tot din imagine de antrenare, dublând setul de date. Astfel, am reușit să obțin o acuratețe mai bună cu aproximativ 10%. De asemenea, **normalizarea** acestora s-a realizat prin împărțirea la 255 a valorilor din array-ul pozelor pentru a avea valori cuprinse în intervalul $[0, 1]$.

Compilarea modelului a fost realizată astfel:

- ❖ **loss**: am folosit drept funcție `loss=SparseCategoricalCrossentropy`. Aceasta este folosită pentru clasificarea în mai multe clase, comparând valorile prezise cu cele reale. De asemenea, acesta acceptă ca input etichete de numere întregi, ci nu etichete reprezentate "one-hot";
- ❖ **optimizer**: am folosit un `optimizer="adam"`, ce are la bază un algoritm de coborâre pe gradient;
- ❖ cât despre **antrenarea modelului**, rezultatele cele mai bune le-am obținut în urma măririi numărului de epoci la 200, și de batches la 256.

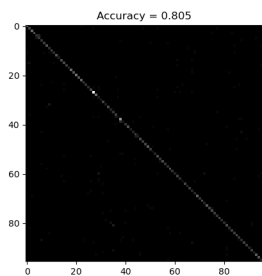
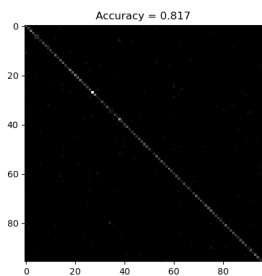
Straturile (layers) folosite în crearea unui model au fost:

- ❖ **Conv2D**: strat ce extrage dintr-o poză caracteristicile cele mai importante (forme, texturi). Acest strat are la bază aplicarea unui filtru prin translatarea unei matrici (kernel) de o dimensiune dată;
 - Numărul de **filtre** are valori de tipul: 32, 64 și 128. După ce am încercat mai multe tipuri de aranjare a straturilor, o abordare ce crește în lățime a dus la o creștere în acuratețe față de o abordare mai adâncă;
 - Dimensiunea de **kernel** găsită ca fiind cea mai optimă este cea de 3x3;
 - Pentru **funcția de activare**, cea mai eficientă s-a dovedit a fi relu;
 - Primul strat are ca **input_shape** poze de (64, 64, 3).
- ❖ **MaxPool2D**: strat ce reduce dimensiunea pozelor. Astfel, următorul strat de convoluție va scana părți mai mari din poze;
- ❖ **Flatten**: transformă inputul într-un array unidimensional pentru a putea fi procesat de layerurile dense;
- ❖ **Dropout**: reduce overfitting prin stabilirea valorii 0 pentru inputuri cu o frecvență dată. Pentru ultimul strat, am încercat să folosesc două tipuri de funcții de activare: softmax și sigmoid. Deși pe datele de validare o abordare folosind sigmoid avea o acuratețe mai mare, pe datele de test, în final, cea mai mare scor l-a obținut modelul cu softmax;
- ❖ **Dense**: strat în care fiecare neuron este conectat de fiecare neuron din stratul precedent (Fully Connected Layer). Am încercat mai multe dimensiuni ale acestor straturi. Un caz remarcabil, care nu făcea overfitting, este folosirea a doua layer Dense (unul de 64, unul de 96), iar între ele un layer Dropout(0.5). Totuși, în soluția finală am folosit trei layer Dense (două de 128 și unul de 96), ce a crescut acuratețea de la 0.796 la 0.805.

Modelul CNN care mi-a generat cea mai bună acuratețe este cel descris mai jos. Acesta a fost creat după încercarea a mai multor hiperparametri și căutarea celor optimi. Toate încercările pot fi accesate în partea C a documentației.

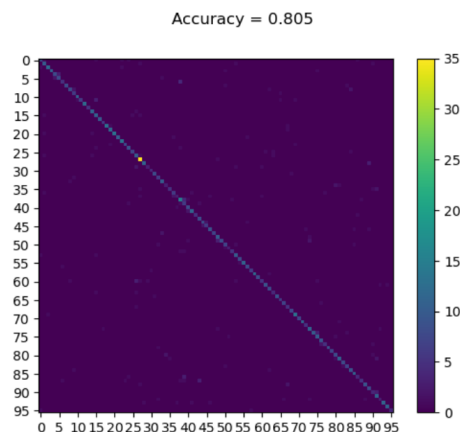
Strat	Hiperparametri
Conv2D	filters=32, kernel=(3, 3), activation="relu", input_shape=(64, 64, 3)
MaxPool2D	pool_size=(2, 2)
Conv2D	filters=64, kernel=(3, 3), activation="relu"
MaxPool2D	pool_size=(2, 2)
Dropout	0.5
Conv2D	filters=128, kernel=(3, 3), activation="relu"
MaxPool2D	pool_size=(2, 2)

Dropout	0.5
Flatten	default
Dense	units=128
Dense	units=128
Dropout	0.5
Dense	units=96, activation="softmax"

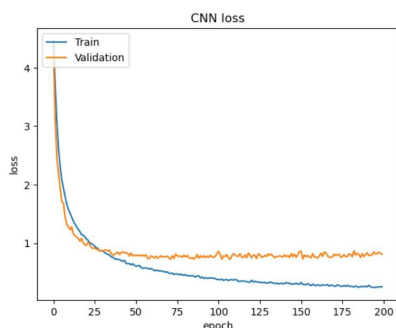
Activarea de pe ultimul strat	Softmax	Sigmoid
Acuratețe pe validare	0.805	0.817
Acuratețe pe test	0.81733	0.81466
Matricea de confuzie		

În final, **cele mai bune rezultate** le-am obținut cu modelul prezentat mai sus, pe cazul softmax. Acestea sunt **rezultatele obținute**:

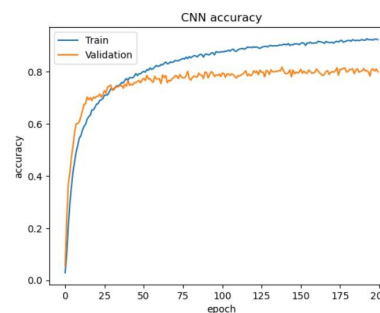
Acuratețea	Precizie	Recall	F1-Score
0.805	0.805	0.805	0.805



Matricea de confuzie - CNN softmax



Graficul train-validation - Loss



Graficul train-validation - Acuratețe

F. Concluzie

În urma tuturor modelelor încercate, cel cu acuratețea cea mai mare o fost implementarea unui CNN, menționat în capitolul anterior, cu o **acuratețe finală de 0.81733**.

G. Bibliografie

- Laboratoarele și Cursurile de Inteligență Artificială FMI
- <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>
- <https://www.tensorflow.org/tutorials/images/cnn>
- <https://ro.myservername.com/what-is-support-vector-machine-machine-learning>
- <https://stackoverflow.com/questions/66785014/how-to-plot-the-accuracy-and-loss-from-this-keras-cnn-model>