Documentatia proiectului

Deep Hallucination Classification

Ene Marius-Andrei, Grupa 242

CNN Sequential Model:

Am folosit libraria keras pentru a implementa acest model.

Pentru a antrena modelul de la zero si a putea clasifica imaginile in diferite clase [0, 1, 2, 3, 4, 5, 6] folosim urmatoarele seturi de date:

- Training images: un np.array care contine imaginile de antrenare, convertite la randul lor intr-un np.arrray.
- Training labels: un np.array in care stocam clasa fiecarei imagine de antrenare
- Validation images: un np.array care contine imaginile de validare. convertite la randul lor intr-un np.array.
- Validation labels: un np.array in care stocam clasa fiecarei imagine de validare
- Test images: un np.array de imagini (convertite intr-un np.array) pe care vom testa modelul creat.

Pentru o invatare mai eficienta:

- Am recodat training_labels in training_labels_one_hot si validation_labels in training_labels_one_hot si validation_labels_one_hot folosind functia **to_categorical** din biblioteca keras, astfel dintr-un vector de clase obtinem o matrice cu len(validation_labels) linii si 7 coloane.

  Ex: 5 -> [0. 0. 0. 0. 0. 1. 0.]

- Am redimensionat valorile pixelilor de la [0, .. , 255] la [0, 1] (nu valori intregi, practic am impartit fiecare pixel la 255)

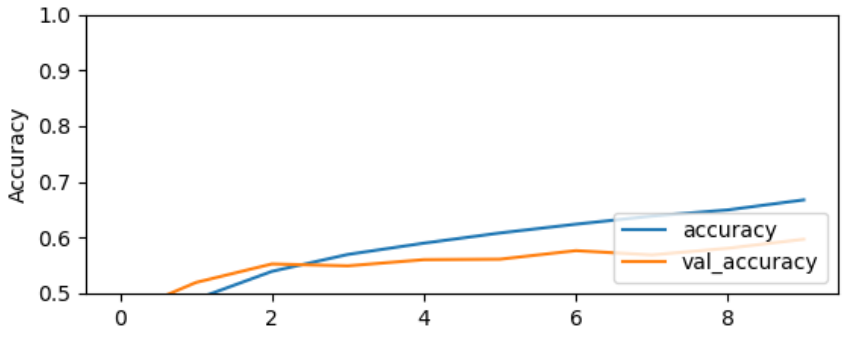
Apoi am creat modelul propriu-zis (adaugand mai multe operatii din cele enumerate mai jos):

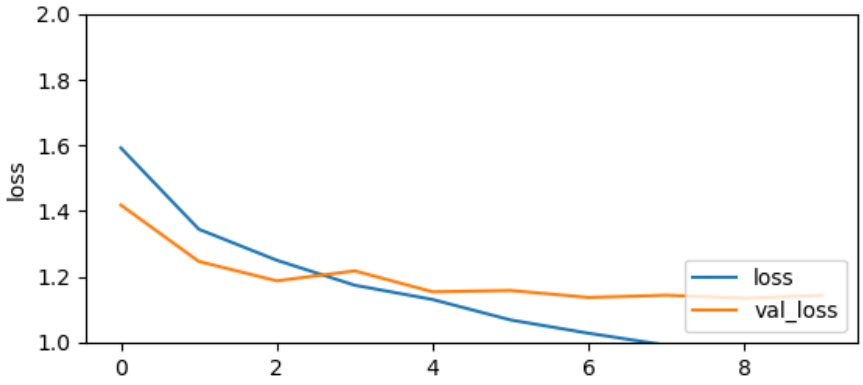
- Am adaugat Conv2D pentru a transforma o imagine in mai multe imagini (cu shape-ul de [16, 16, 3])
- Am folosit MaxPooling2D pentru a pune la maximum valoarea din matricea de dimensiuni, iar acest lucru il folosim si pentru urmatoarele 2 straturi.
- Am adaugat Flatten() pentru a aplatiza dimensiunile imaginii obtinute dupa convolutia acesteia.
- Am adaugat Dense() pentru a-l conecta cu neuronii de la layer-ul anterior
- Am adaugat Dropout pentru a evita overfitting-ul dataset-ului.

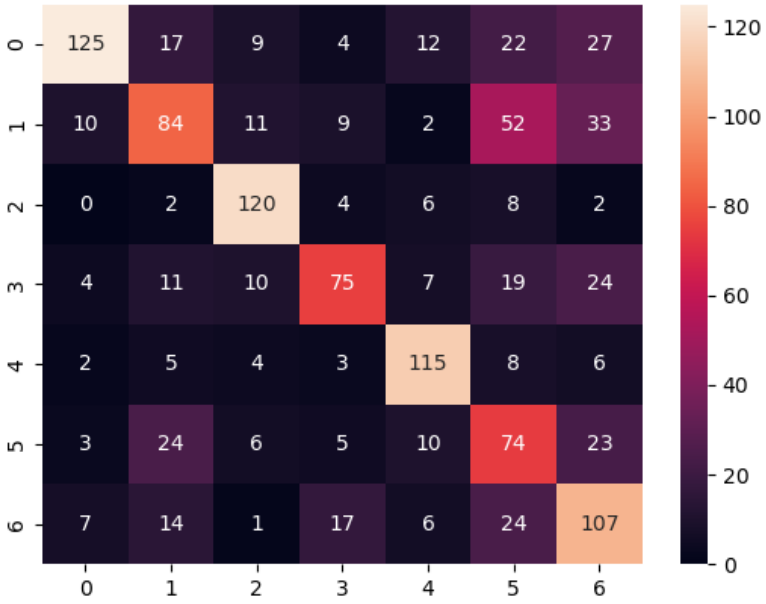
Pentru a definii pierderile, optimizarile si metrics am compilat modelul folosind functia **compile** din libraria keras. Apoi, folosind functia **fit** am antrenat modelul pe datele de antrenare, cu optiniunea de a se verifica pe datele de validare.

Am testat modelul pentru 10, 50 si 100 de epoci si am obtinut urmatoarele rezultate:

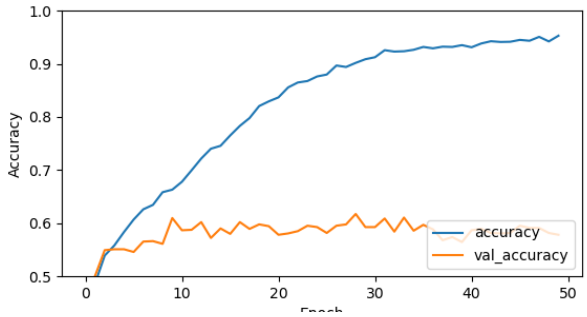
10 Epocs: Loss:1.143410325050354, Accuracy:0.5967604517936707

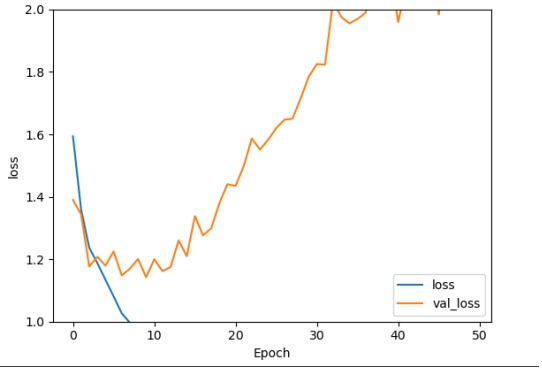


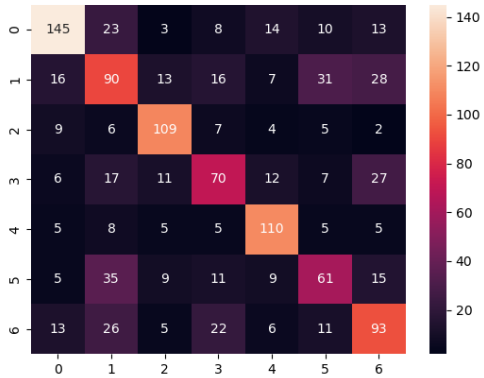

2.0
1.8
1.6
1.4
1.2
1.0
loss
0
2
4
6
8
loss
val_loss


0
1
2
3
4
5
6
125 17 9 4 12 22 27
10 84 11 9 2 52 33
0 2 120 4 6 8 2
4 11 10 75 7 19 24
2 5 4 3 115 8 6
3 24 6 5 10 74 23
7 14 1 17 6 24 107
0 1 2 3 4 5 6
120
100
80
60
40
20
0

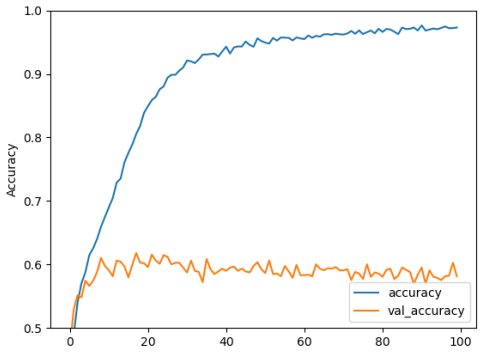
50 Epocs: Loss:2.280545234680176, Accuracy:0.5780051350593567

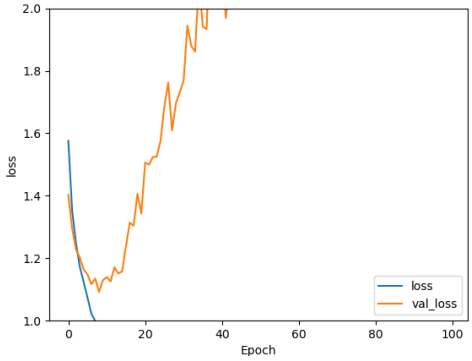


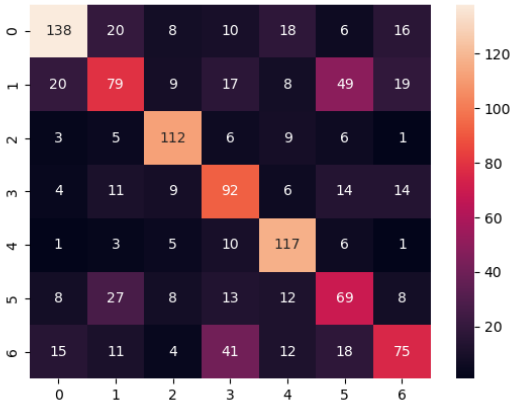




100 Epocs: Loss:2.971266746520996, Accuracy:0.5814151763916016







Am incercat sa pun diferite valori pentru Convolution, Dense, Batch in speranta de a obtine o acuratete mai buna, dar nu s-a intamplat. (Se poate vedea in cod).

SVM Model:

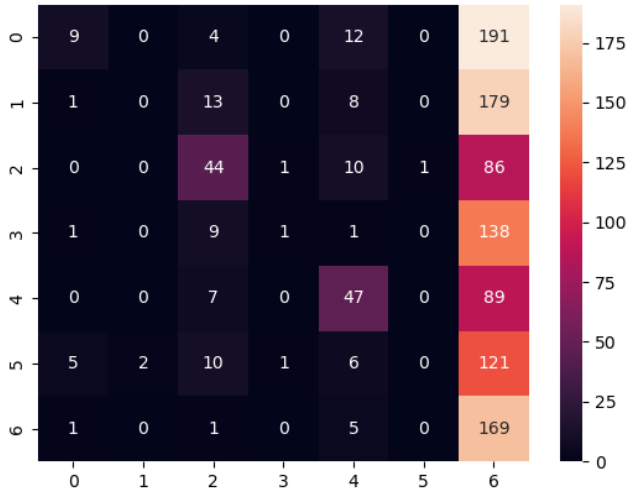
La fel ca in modelul anterior realizat aceleasi operatii pentru datele de antrenare, singura diferenta este modul in care antrenam AI-ul.

Pentru inceput, folosing biblioteca sklearn am normalizat datele in 3 tipuri: Standard, L1, L2.

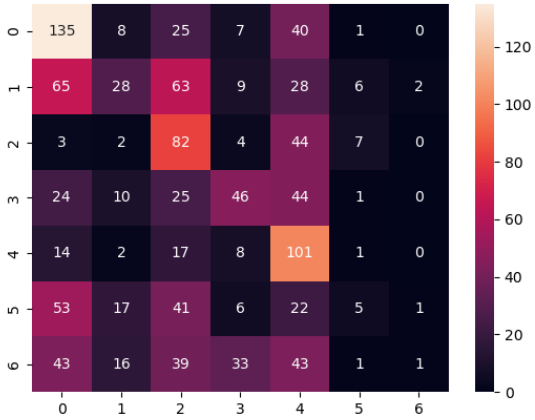
Apoi am creat un model folosind functia SVC (Support Vector Classifier) in care am dat fit datelor de antrenare (images + labels).

Folosind functia **predict** am verificat acuratetea modelului pe datele de validare si am obtinut pentru cele 3 tipuri de normalizare:

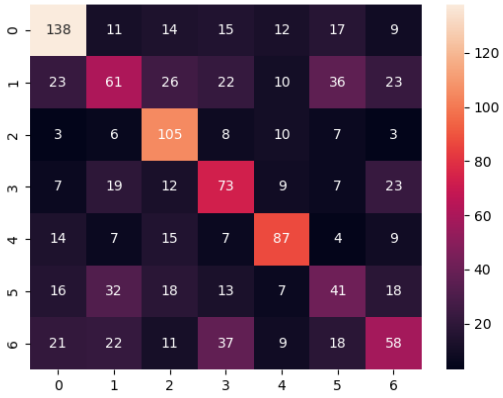
L2: Accuracy: 0.23017902813299232 si matricea de confuzie:



L1: Accuracy: 0.3393009377664109 si matricea de confuzie:



Standard: Accuracy: 0.4799658994032396 si matricea de confuzie:



Dupa cum se observa acuratetea noului model este mai mica decat a celui anterior.