Oprea Laurentiu-Cristian
Gr 242
AI Documentation

# Deep Hallucination Classification

**Scopul** lucrarii este de a antrena un model cat mai precis pentru clasificarea in 7 (etichetate de la 0 la 6) clase a unor imagini '.png' obtinute prin algoritmi deep hallucination.

**Datele de intrare** cuprind 3 fisiere .txt (test.txt, train.txt si validation.txt) si doua foldere continand imaginile .png pentru testare respectiv pentru antrenare si validare.

Indiferent de modelul folosit, primul pas a fost prelucrarea datelor. Incep prin a incarca fisierele text (numele pozelor respectiv label-urile lor, sau doar numele lor in cazul fisierului test.txt) , incarcandu-le intr-un np.array cu functia loadtxt. Fisierele de train si validation vin sub formatul "id,label", asadar il split-uim mai departe in alte doua liste ( de id-uri si de label-uri). Acum ca avem aceste date citite, incarcam si imaginile intr-un np.array. Functia de incarcat imaginile ramane la latitudinea programatorului, asadar eu am ales functia imread() din libraria matplotlib, pentru ca are avantajul de a citi imaginile deja scalate (pixelii au valori float de la 0-negru la 1-alb, in loc de 0-negru si 255-alb), si nu trebuie sa mai parcurg si acel pas in preprocesarea datelor. Pentru a putea antrena modelele, acum trebuie aplatizata matricea imaginilor, transformand matricea de antrenament din shape (8000, 16, 16, 3) in shape (8000, 768), avand acum pe fiecare rand o alta imagine, avand primele 256 de elemente ale liniei valorile pixelilor RED, urmatoarele 256 pentru pixelii GREEN si ultimele 256 pentru pixelii BLUE. Acest proces de aplatizare il aplicam inainte de antrenarea modelelor KNN si SVM, dar nu si CNN pentru ca acolo folosim un model ce functioneaza si cu parametrii de shape (size, n, n, 3).

**KNN Model**

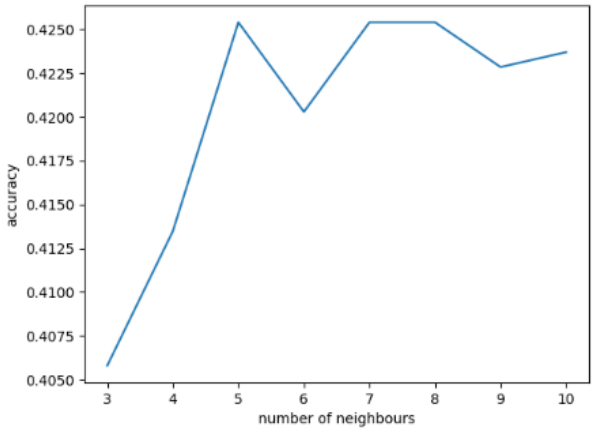
Acum ca avem datele pregatite, incepem antrenarea diferitelor modele. Primul model incercat este un K nearest neighbours , caruia ii implementez clasificatorul prin intermediul unei clase. Asupra unei imagini de test, clasificatorul realizeaza urmatorii pasi: Calculeaza toate distantele dintre imagine si imaginile de antrenament prin formula - $L2(X, Y) = \sqrt{\sum_{i=1}^{n} (X_i - Y_i)^2}$ , alege cele mai apropiate K distante, afla label-urile fata de care s-au calculat acele distante, aplica vector de frecventa pe aceste labeluri iar labelul cu cele mai multe aparitii este predictia pentru imaginea de test.

Clasificam apoi toate imaginile de validare, iar acuratetea o calculam printr-un for ce compara predictiile cu validation_labels.

Modelul obtine o acuratete de 0.43 pe setul de validare , pentru K=5. Dupa repetate rulari observ ca in K=5 acuratetea modelului este maxima.

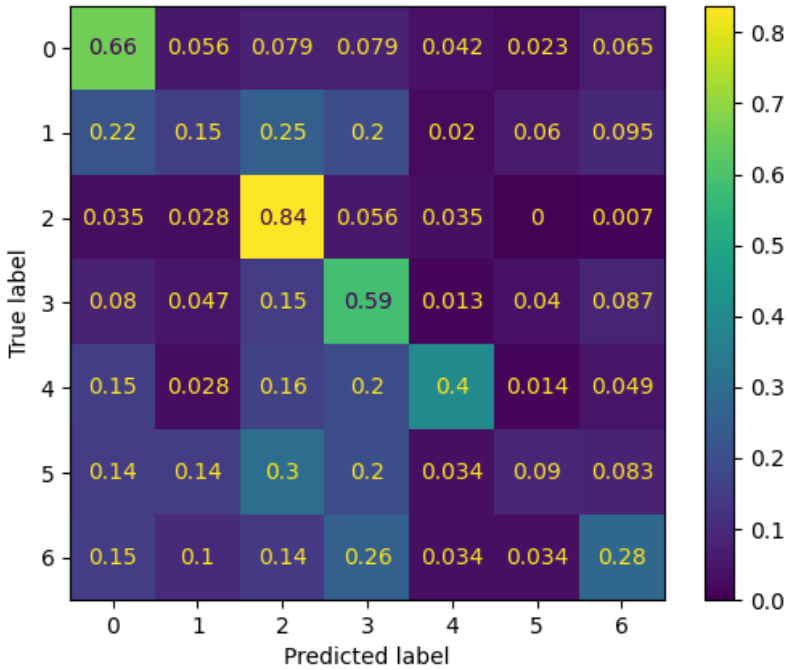
Graficul acuratetii:

Este obtinut prin calcularea acuratetii pentru [3, 4, .., 10] vecini, iar plotul este facut cu matplotlib




Matricea de confuzie:

Este obtinuta prin aplicare functiei ConfusionMatrixDisplay din libraria sklearn.metrics asupra predictiilor

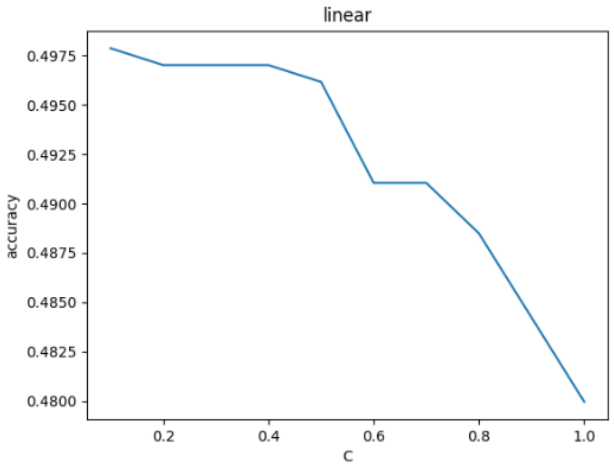


**SVM Model**

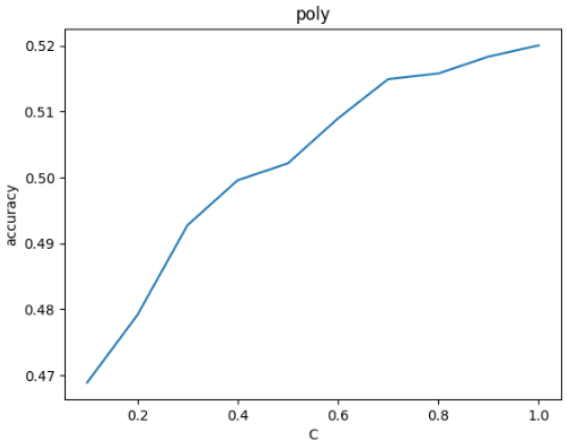
Al doilea model utilizat este un Support Vector Machine(SVM), implementat cu ajutorul librariei sklearn. Initializam un SVC(classifier) caruia ii pasam parametrii doriti (tipul de kernel si parametrul C), si apoi il antrenam cu datele de antrenare. Pentru a realiza predictiile apelam SVC.predict, iar pentru a masura acuratetea folosim functia predefinita in sklearn.metrics, accuracy_score().

<u>Graficul acuratetii in functie de kernel si **C**:</u>

- Pentru kernel liniar:
  Observam ca valoarea optima este C = 0.1 pentru kernelul liniar



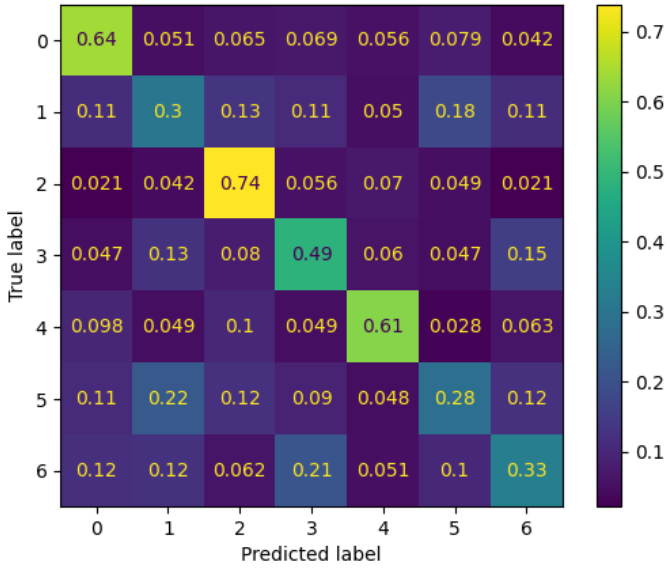

- Pentru kernel poly:
  Observam ca valoarea optima este C = 1 pentru kernelul poly




Asadar acuratetea maxima de 0.52 o obtinem antrenand modelul SVM pe un kernel poly cu C=1.

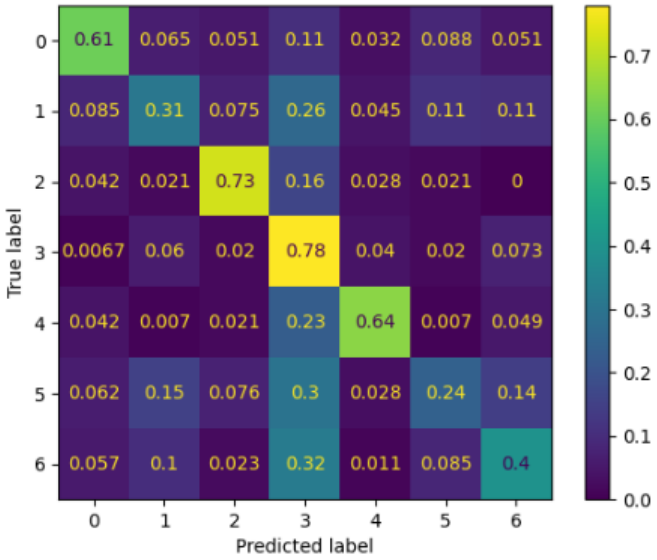
Matricea de confuzie :

Aceasta matrice este obtinuta pentru SVM cu kernel liniar si C=1 (overfit, acuratete minima).




Aceasta matrice este obtinuta pentru SVM cu kernel poly si C=1 (acuratete maxima).

Diferenta principala intre cei doi kerneli este ca cel polinomial a identificat mult mai precis imaginilie din categoria 3.




**CNN Model**

Ultimul si cel mai precis model antrenat este un Convolutional Neural Network, implementat cu ajutorul modelului Sequential() din keras.models. Layerele utilizate (Convolutional, Dense, Dropout, Flatten, Pooling) sunt implementate cu ajutorul librariei keras.layers.

Citirea imaginilor se face de data aceasta, fata de celelalte modele, fara aplatizarea acestora in np.arrayul de imagini, pentru ca modelul este scris in asa fel incat sa proceseze si imagini (size, size, 3) , fata de celelalte modele unde era nevoie de (size*size*3, ).

Layerele sunt adaugate manual in modelul Sequential.

Layout-ul final al straturilor convolutionale este urmatorul:

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 16, 16, 100)       2800

 dropout (Dropout)           (None, 16, 16, 100)       0

 conv2d_1 (Conv2D)           (None, 16, 16, 150)       135150

 max_pooling2d (MaxPooling2D  (None, 8, 8, 150)        0
 )

 dropout_1 (Dropout)         (None, 8, 8, 150)         0

 conv2d_2 (Conv2D)           (None, 8, 8, 200)         120200

 max_pooling2d_1 (MaxPooling  (None, 4, 4, 200)        0
 2D)

 dropout_2 (Dropout)         (None, 4, 4, 200)         0

 flatten (Flatten)           (None, 3200)              0

 dense (Dense)               (None, 300)               960300

 dropout_3 (Dropout)         (None, 300)               0

 dense_1 (Dense)             (None, 100)               30100

 dropout_4 (Dropout)         (None, 100)               0

 dense_2 (Dense)             (None, 7)                 707

=================================================================
Total params: 1,249,257
Trainable params: 1,249,257
Non-trainable params: 0
```

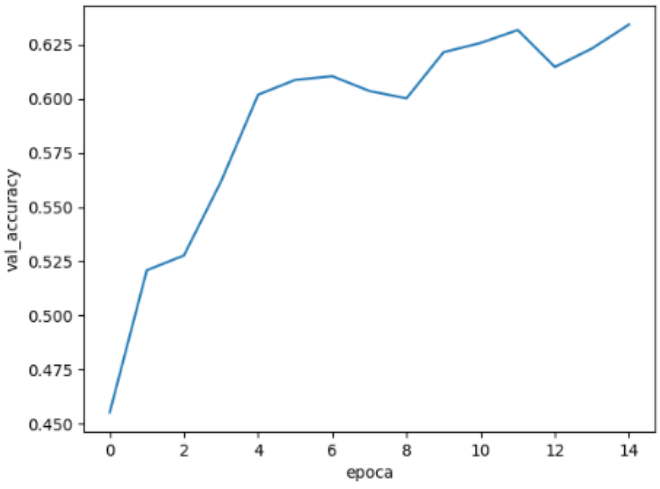
Procesul prin care am ajuns la aceste straturi a fost un trial and error, unde dupa cateva layouturi am observat ca acuratetea este maxima atunci cand am 3 straturi convolutionale, intrucat mai multe ar fi dus la overfitting iar mai putine nu ar identifica optim trasaturile cheie ale imaginii. Initial realizam un pooling dupa fiecare convolutional, dar am decis ca mai bine renunt la poolingul de dupa primul convolutional, caci imaginea s-ar micsora prea mult pana in final (ar fi ajuns 2x2) , si ar intoarce predictii foarte imprecise. Crestem numarul de filtre de la un strat convolutional la altul (100, 150, 200) pentru a delimita cat mai bine trasaturile imaginii. Fiecare strat este urmat de cate un dropout , pentru ca asa previn overfittingul cand am multiple straturi convolutionale. Se aplica apoi un Flatten pentru a obtine un vector liniar din pool, si a putea mai apoi sa-l pasam straturilor Dense. Straturile Dense sunt straturi complete unde se intampla de fapt procesul de invatare. Unitatile/Dimensiunile straturilor dense au fost  la randul lor alese prin trial and error, si am observat ca o acuratete optima se obtine undeva in jurul valorii de 1.2 milioane de parametri antrenabili. De la un strat Dense la altul avem din ce in ce mai putine unitati (300, 100, 7).

Compilarea modelului se face cu unul dintre optimizerii ‘Adam’ sau ‘Adamax’ si cu functia de pierdere Sparse categorical crossentropy ( o functie probabilistica ce accepta labeluri Integer).

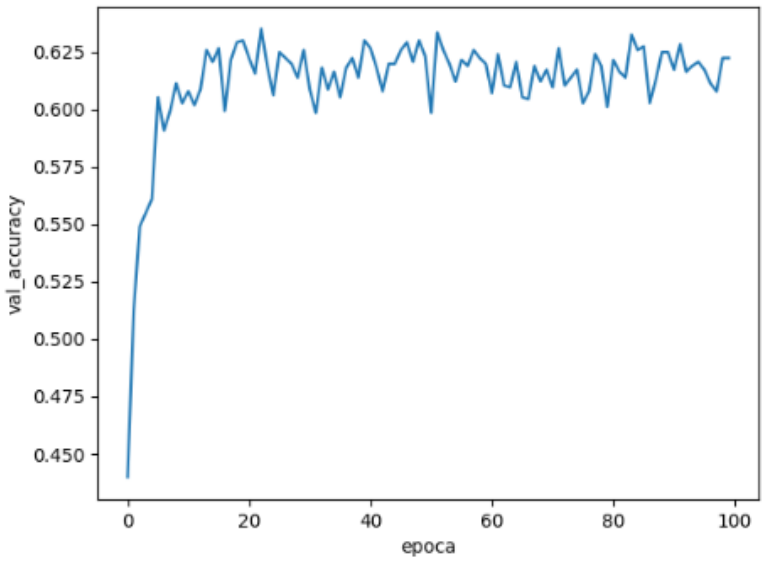
Antrenarea modelului se face pe un numar de 15 epoci, valoare gasita a fi optima prin testare (se vede mai bine in graficul acuratetii).

<u>Graficul acuratetii in functie de numarul de epoci:</u>

CNN rulat pe 15 epoci:



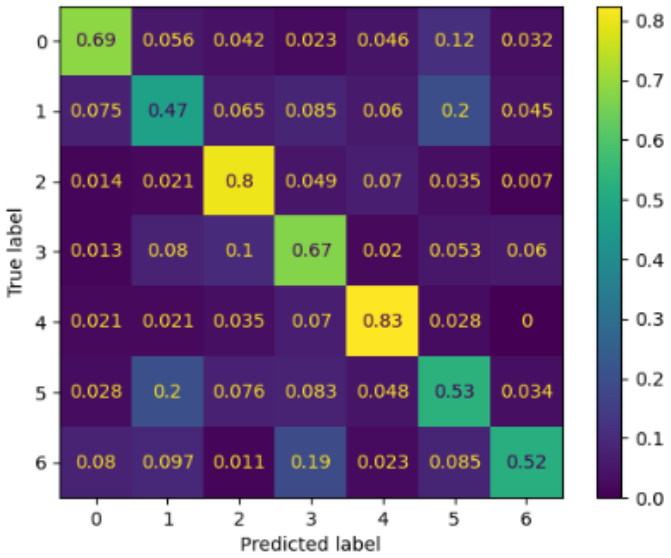
CNN rulat pe 100 de epoci:




Se poate observa din acest grafic ca maximul acuratetii este atins in zona 15-25 de epoci.

<u>Matricea de confuzie:</u>

CNN rulat pe 15 epoci:




CNN rulat pe 100 de epoci: