Broscoțeanu Daria-Mihaela
Grupa 243

Deep Hallucination Classification

I. Descrierea Proiectului

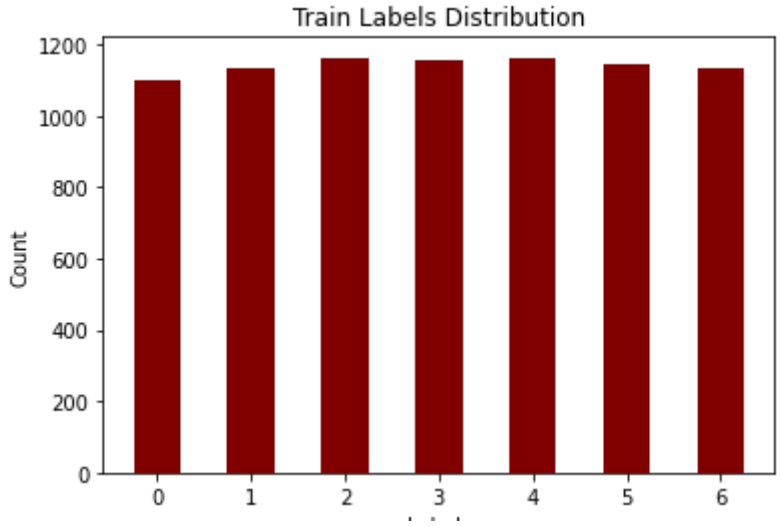
Proiectul presupune clasificarea unor imagini de dimensiune 16 x 16 (halucinații) în 7 clase distincte, numerotate de la 0 la 6.

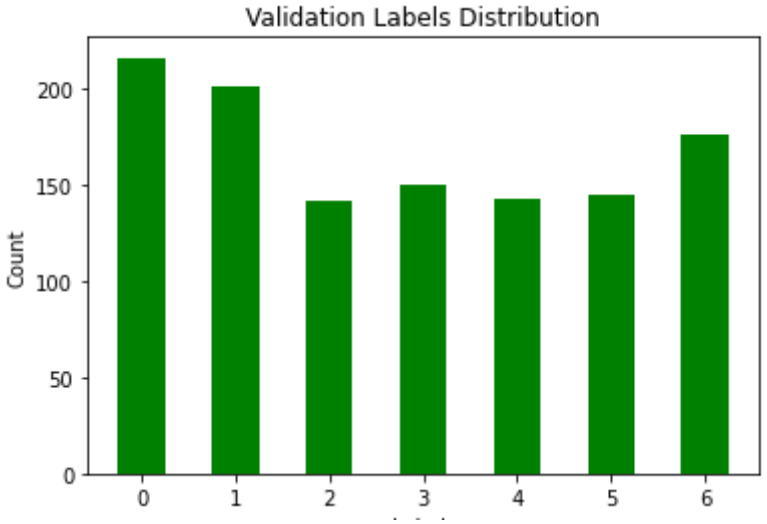
II. Setul de date

Pentru a putea clasifica aceste imagini, primim drept input următoarele seturi de date:

- 8000 de imagini de train, însoțite de label-urile corespunzătoare
- 1173 de imagini de validare, însoțite de label-urile corespunzătoare
- 2819 imagini de test

Distribuția label-urilor pe seturile de date de train și validare:





Broscoțeanu Daria-Mihaela
Grupa 243

III. Abordări

În rezolvarea acestei probleme am abordat mai mulți algoritmi pe care îi voi prezenta în continuare:

- KNN
- CNN

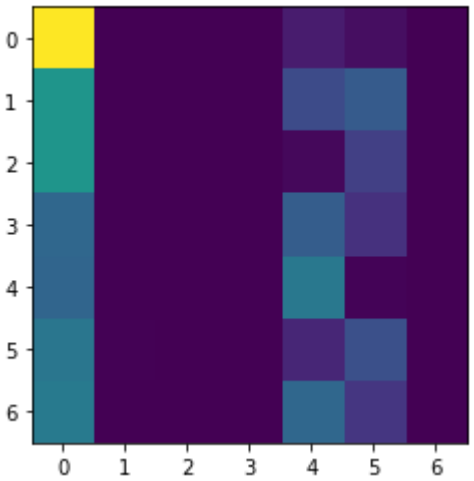
A. KNN

Într-o primă abordare a problemei, am optat pentru KNN. KNN este un algoritm simplu de clasificare care are rolul de a clasifica puncte în funcție de primii k cei mai apropiați vecini ai acestuia. Acești k vecini vor participa la un vot care va stabili prin vot majoritar în ce categorie putem încadra elementul pe care vrem să îl clasificăm.

Fiind primul algoritm pe care l-am încercat, am optat pentru o citire a datelor fără însă a le procesa prea mult. M-am folosit de PIL pentru a citi aceste date și de a le adăuga noise datelor de train, apelând la GaussianBlur(3) din ImageFilter pentru a adăuga acest noise. Am lucrat astfel cu np array-uri care cuprindeau pentru fiecare pixel valorile RGB cu valori între 0 și 255.

Preluând algoritmul prezentat în laborator, am implementat atât o normalizare a datelor care permite normalizarea după metricile L1 și L2. Cum setul meu de date de train avea în acest moment 16000 de imagini (atât cele originale, cât și imaginile peste care am adăugat noise), am ales ca număr de vecini care să participe la votul de determinare 400 - întrucât am citit că valoarea optimă ar fi rădăcina numărului de date din setul de train.

Parametrii pe care i-am folosit pentru aceasta clasificare au fost metrică L2 și 400 ca număr de vecini, însă aceștia au dus la o acuratețe de aproximativ 0.40. Acuratețe pe care aș fi putut să o îmbunătățesc dacă augmentam setul de date corespunzător și preprocesăm imaginile într-o manieră mai corectă, ceea ce am și realizat, însă pentru următorul model.



Matricea de confuzie pentru KNN

Broscoțeanu Daria-Mihaela
Grupa 243

B.CNN

CNN = rețele neuronale convolutionale care cuprind straturi de convoluție peste care se aplica anumite funcții de activare. Pentru implementarea modelului meu m-am folosit de keras din tensorflow.

1. Procesarea și augumentarea datelor

În cadrul abordării cu modelul CNN, am făcut procesări asupra setului de date de antrenare, cât și de validare.

Într-o primă etapă, am efectuat doar următoarele procesări și augumentari, folosindu-mă de funcțiile din OpenCv:

- valorile RGB ale pixelilor le-am redus la intervalul [0,1] de la intervalul [0,255]
- am rotit imaginile în diferite maniere și le-am adăugat setului de dat de train cu label-ul imaginii de la care am plecat

Abordarea acesta care transforma adăuga în citire şi imaginile rotite în diferite maniere a dus însă la o creştere a overfitting-ului, ceea ce m-a determinat să trec la realizarea acestor operaţii cu următoarea abordare.

În algoritmul final, am efectuat următoarele procesări și argumentări, folosindu-mă de funcțiile din keras.preprocessing.image.ImageDataGenerator, împărțind imaginile în batch-uri de 32:

- valorile RGB ale pixelilor le-am redus la intervalul [0,1] de la intervalul [0,255]
- featurewise_std_normalization
- horizontal_flip
- vertical_flip
- featurewise_center

2. Definirea Modelului

2.1. Layere folosite:

- Conv2D - layer care creeaza un kernel de convoluție care aplica filtre asupra inputului pentru a genera o matrice de feature-uri.
- BatchNormalization - layer care permite transmiterea inputului intr-o forma standardizata - fără acest layer rețeaua mea are o învățare lentă - stratul normalizeaza datele primite ca input și le face astfel incat media este 0, iar dispersia 1
- MaxPool2D - layer care reduce dimensiunile inputului prin “glisarea” unei ferestre de dimensiunea oferită ca parametru și luand maximul din aceasta

- Dropout - layer care are rolul de a reduce overfitting-ul prin setarea random a unor unităţi(neuroni) cu 0(adică nu mai sunt activati) cu o frecvență data ca parametru
- Flatten - layer care aplatizeaza input-ul
- Dense - layer care reprezintă un strat dens conectat de NN.

2.2. Hiperparametrii:

- pentru layerele convoluţionale:
  - am folosit un număr de filtre din mulţimea {16, 32, 64, 128}, crescând din două în două layere numărul de filtre aplicate
  - am folosit un kernel de (3,3) - adică o ”fereastră” de 3 x 3 pe care am glisat-o pe suprafața pozelor
  - ca activator, am folosit relu - max(input, 0) - am ales relu deoarece este un optimizator simplu și rapid acre obține rezultate mai bune pe antrenarea NN-urilor decât sigmoid
  - padding=”same” - inputul va avea același shape ca outputul pentru layerele convolutionale
- pentru layerele doupout: las rata de drouput la 0,25 - am încercat și cu valori mai mari (spre exemplu cu 0.4 sau 0.5), dar modelul îmi făcea underfitting
- pentru layerele dense:
  - folosesc pentru toate layerele cu excepția ultimului funcția de activare relu și un număr de 256 de unități
  - pentru ultimul layer, folosesc 7 unități pentru a clasifica în 7 clase, cât și activatorul softmax care convertește pentru fiecare valoare de label probabilitatea ca rezultatul să fie acel label
- pentru celelalte layere le folosesc cu parametrii default

2.3 Compilarea modelului:

- folosesc funcția de loss categorical_crossentropy care calculează loss-ul dintre label-uri și predicții și este folosită pentru clasificarea în clase multiple - această funcție aplică succesiv softmax și apoi cross-entropy loss
- ca optimizator, mă folosesc de adam - care e o extindere de la stochastic gradient descent (care nu foloseste tot setul de date de train pentu direcție, ci o submulțime de sample-uri)
- iar metrica modelului este acuratețea

2.4 Checkpoint-uri:

- pentro o acuratețe mai bună, am ales să-mi păstrez un checkpoint la epoca în care acuratețea pe datele de validare este cea mai mare, salvând weight-urile acesteia

2.5 Variante de CNN-uri încercate:

Într-o primă varianta, am mers pe o rețea neurală mai mică:

Layer	Hiperparametrii
Conv2D	filters = 64, kernel = (3,3), activation= relu
BatchNormalization	default
Conv2D	filters = 64, kernel = (3,3), activation= relu
Dropout	rate = 0.25
Conv2D	filters = 64, kernel = (3,3), activation= relu
BatchNormalization	default
MaxPool2D	pool_size = (2,2)
Dropout	0.25
Flatten	default
Dense	units = 128, activation = ”relu”
Dropout	0.25
Dense	units = 7, activation = ”softmax”

Cu această rețea am reușit să obțin inițial o acuratețe de 0.55255 pe care am reușit să o îmbunătățesc până la 0.62642 prin augumentarea setului de date. Acest model de CNN însă nu m-a ajutat să depășesc o acuratețe de peste ultima valoare menționată, ceea ce a dus la un nou model, având câteva layerele în plus și modificări asupra hiperparametrilor.

Layer	Hiperparametrii
Conv2D	filters = 64, kernel = (3,3), activation= relu
BatchNormalization	default
Conv2D	filters = 64, kernel = (3,3), activation= relu

Dropout	rate = 0.25
Conv2D	filters = 64, kernel = (3,3), activation= relu
BatchNormalization	default
Conv2D	filters = 64, kernel = (3,3), activation= relu
BatchNormalization	default
MaxPool2D	pool_size = (2,2)
Dropout	0.25
Flatten	default
Dense	units = 128, activation = ”relu”
Dropout	0.25
Dense	units = 128, activation = ”relu”
Dropout	0.25
Dense	units = 7, activation = ”softmax”

Cu această rețea am reușit să obțin o acuratețe maximă de 0.63068 cu tot cu augumentările asupra datelor. Pe această rețea am testat numeroase valori ale hiperparametilor care însă nu au reușit să mă ajute să depășesc acea valoare a acurateții:

- am crescut kernel_size =(5,5) și filters = 128,, însă am reușit să obțin maxim 0.62926
- am crescut rata de dropout la 0.4, insă acest lucru a dus la underfitting și la o acuratețe maxima de 0.60227

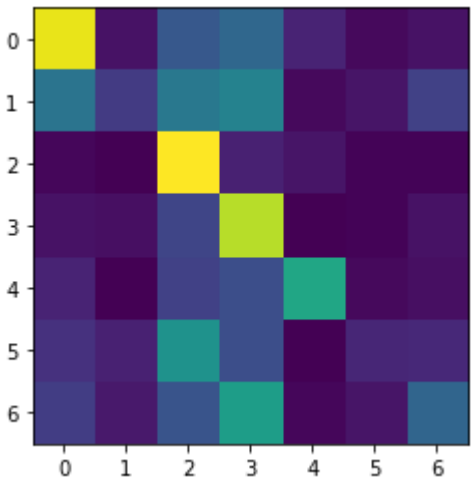
Aceste valori au dus la modificarea modelului și la apariția versiunii finale de rețea:

Layer	Hiperparametrii	Output Shape
Conv2D	filters = 16, kernel = (3,3), activation= relu, padding=same	(None, 16, 16, 16)
BatchNormalization	default	(None, 16, 16, 16)
Conv2D	filters = 16, kernel = (3,3), activation= relu, padding=same	(None, 16, 16, 16)

BatchNormalization	default	(None, 16, 16, 16)
MaxPool2D	default	(None, 8, 8, 16)
Dropout	rate = 0.25	(None, 8, 8, 16)
Conv2D	filters = 32, kernel = (3,3), activation= relu, padding=same	(None, 8, 8, 32)
BatchNormalization	default	(None, 8, 8, 32)
Conv2D	filters = 32, kernel = (3,3), activation= relu, padding=same	(None, 8, 8, 32)
BatchNormalization	default	(None, 8, 8, 32)
MaxPool2D	default	(None, 4, 4, 32)
Dropout	rate = 0.25	(None, 4, 4, 32)
Conv2D	filters = 64, kernel = (3,3), activation= relu, padding=same	(None, 4, 4, 64)
BatchNormalization	default	(None, 4, 4, 64)
Conv2D	filters = 64, kernel = (3,3), activation= relu, padding=same	(None, 4, 4, 64)
BatchNormalization	default	(None, 4, 4, 64)
MaxPool2D	default	(None, 2, 2, 64)
Dropout	rate = 0.25	(None, 2, 2, 64)
Conv2D	filters = 128, kernel = (3,3), activation= relu, padding=same	(None, 2, 2, 128)
BatchNormalization	default	(None, 2, 2, 128)
Conv2D	filters = 128, kernel = (3,3), activation= relu, padding=same	(None, 2, 2, 128)
BatchNorma	default	(None, 2, 2, 128)

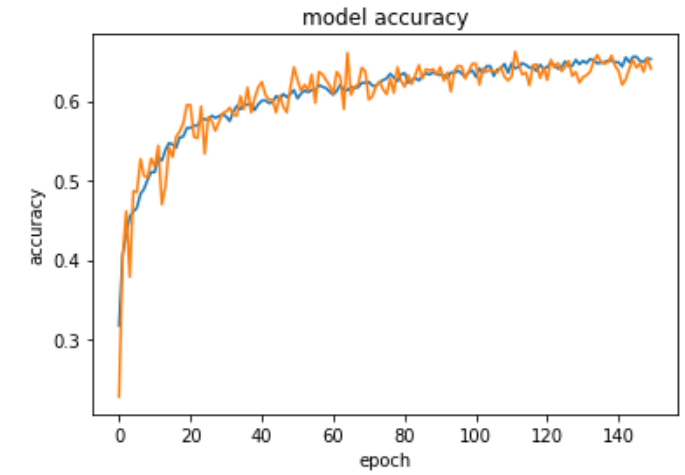
lization		
MaxPool2D	default	(None, 1, 1, 128)
Dropout	rate = 0.25	(None, 1, 1, 128)
Flatten	default	(None, 128)
Dense	256, activation="relu"	(None, 256)
Dropout	rate = 0.25	(None, 256)
Dense	256, activation="relu"	(None, 256)
Dropout	rate = 0.25	(None, 256)
Dense	256, activation="relu"	(None, 256)
Dropout	rate = 0.25	(None, 256)
Dense	7, activation='softmax'	(None, 7)

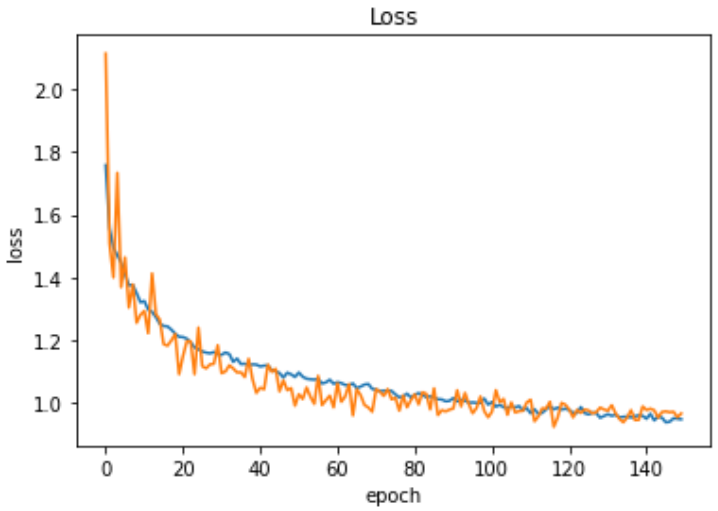
Pe acest model am reuşit să obţin o acurateţe de 0.6624 pe datele de validare, avand loss-ul egal cu 0.9421 după aproximativ 110 epoci, iar în competiţie am obţinut o acuratețe maximă de 0.65198.



Matricea de confuzie pentru una din ultimele rulări - care obține pe datele de test pe Kaggle ~0.62357

Broscoțeanu Daria-Mihaela
Grupa 243





2.6. Hiperparametrii

Model	Hiperparametrii	Acuratete
KNN	nr vecini = 400	0.41
CNN v1	din tabel	0.62642
CNN v2	kernel_size =(5,5) și filters = 128	0.62926
CNN v2	rata de dropout la 0.4	0.60227
CNN v3 (final)	din tabel	0.65198

Broscoțeanu Daria-Mihaela
Grupa 243

IV. Referințe

1. Cod laborator si https://fmi-unibuc-ia.github.io/ia/
2. https://keras.io/examples/
3. https://keras.io/api/
4. https://www.pythonpool.com/
5. https://machinelearningmastery.com/
6. https://www.tensorflow.org/resources/learn-ml?gclid=CjOKCQjw6J-SBhCrARIsAHOyMZg3t_XiyWxlpOYrODeg5oai_4d_P-oHab66W7jlLRnMltOGlsZgQ7gaArATEALw_wcB
7. https://docs.opencv.org/4.x/