Deep Hallucination Classification
Nedelcu Radu - Ioan
Grupa 252

Descrierea proiectului:

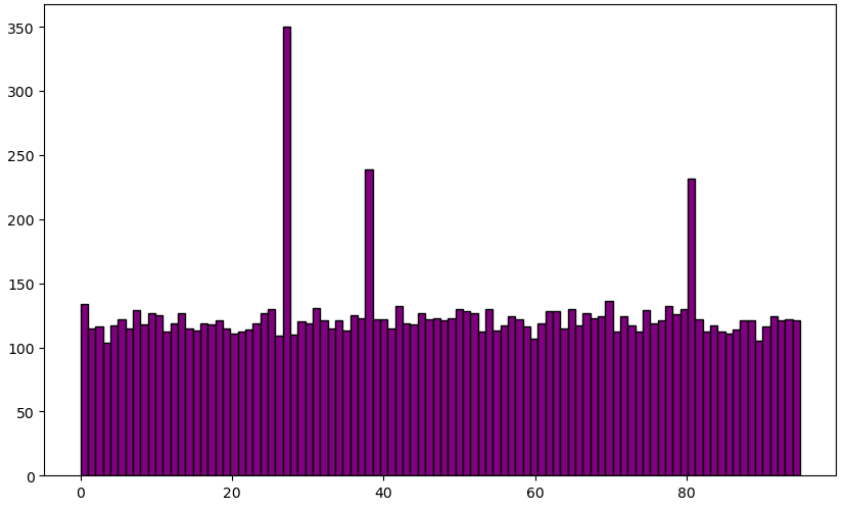
Proiectul Deep Hallucination Classification a constat într-o competiție de 3 săptămâni în care fiecare participant a trebuit să construiască un model de inteligentă artificială care să clasifice imagini intr-una dintre 96 de categorii. În acest scop au fost puse la dispoziție 12000 de perechi de forma (Imagine, Clasa) pentru antrenarea modelului, plus alte 1000 de perechi pentru validarea modelului în timpul dezvoltării.

Abordări:

I. Naive Bayes

Naive bayes este un simplu algoritm de clasificare, care pleacă de la presupunerea *naivă* că oricare dintre caracteristicile unui set de date sunt complet independente. Evident, în cazul cerinței clasificării imaginilor date, acestea nu sunt independente, deci ne putem aștepta la o acuratețe scăzută din partea acestui algoritm.

În primul rând, pentru a putea construi modelul am citit datele de citire, respectiv validare și am observat distribuția acestora în cele 96 de clase:

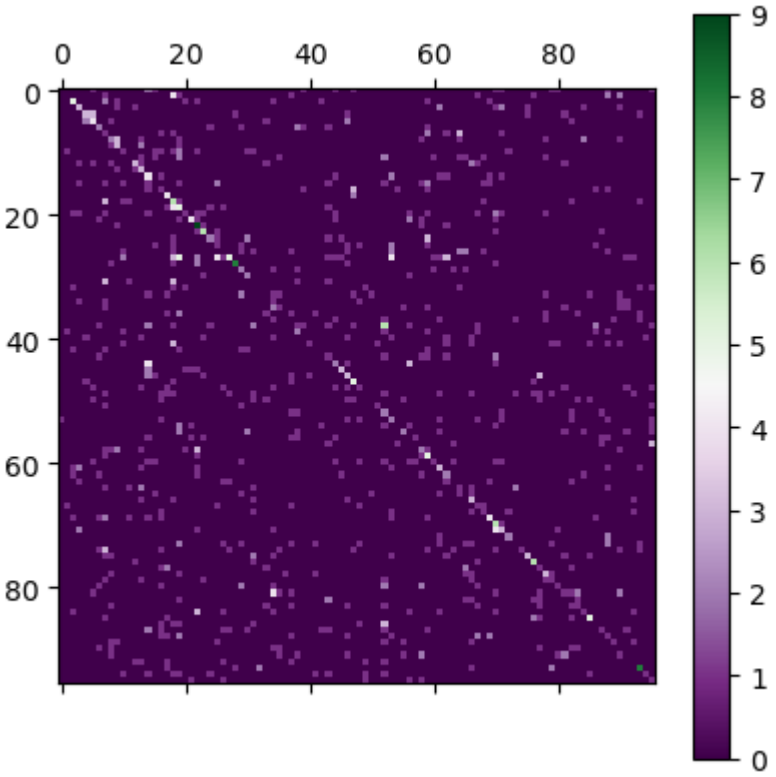


Cu excepția a 3 dintre clase, majoritatea au număr aproximativ egal de imagini în datele de train.

Pentru citirea imaginilor am folosit modulul PIL, după care am folosit funcția flatten() pentru a transforma fiecare imagine dintr-un array de shape 64 x 64 x 3 într-un array unidimensional de lungime 12288. Astfel, pentru algoritmul de clasificare, fiecare dintre cele 3 canale ale fiecărui pixel dintr-o imagine va reprezenta o trăsătură.

După acești pași, modelul Naive Bayes poate fi instanțiată și antrenat cu doar două linii de cod. Antrenarea durează doar câteva secunde pe datele de train, datorită simplității sale. Din nefericire, acuratețea obținută de acesta este de doar 0.18.

Matricea de confuzie generată de acest algoritm:



II. Convolutional Neural Networks

În cea de-a doua parte a proiectului, am folosit un CNN, care constă într-o succesiune de etape (layers) utilizate pentru antrenarea rețelei.

Citirea și prelucrarea inițială a datelor este similară cu cea descrisă pentru Naive Bayes, cu excepția faptului că nu mai este necesară apelarea funcției *flatten()*.

Variante încercate ale algoritmului:

Layer	Hiperparametrii
Conv2D	filters = 32, kernel = (3,3), activation= relu, input_shape=(64, 64, 3)
MaxPooling2D	pool_size = (2,2)
Conv2D	filters = 64, kernel = (3,3), activation= relu
MaxPooling2D	pool_size = (2,2)
Flatten	-
Dense	128, activation='relu'
Dense	96, activation='softmax'

Cu această abordare am obținut o acuratețe de 0.60 pe datele de test. Urmând exemplele și discuțiile din laborator, am adaugat ulterior încă un Convolution Layer (filters=128, kernel = (3,3)), cu care am reușit să îmbunătățesc acuratețea până la 0.629

Ulterior am încercat să optimizez funcțiile de activare, încercând alternative pentru ReLU:

Conv I	Conv II	Conv III	Dense	Acuratețe
ReLU	ReLU	ReLU	ReLU	0.629
GELU	GELU	GELU	GELU	0.638
GELU	GELU	GELU	ReLU	0.609
ReLU	ReLU	ReLU	GELU	0.624
GELU	ReLU	GELU	ReLU	0.698
GELU	ReLU	GELU	Swish	0.65

Toate aceste teste au fost realizate folosind 40 de epoci, pentru a economisi timp. Am după aceste încercări am concluzionat că este optimă alternarea între funcțiile de activare GELU si ReLU.

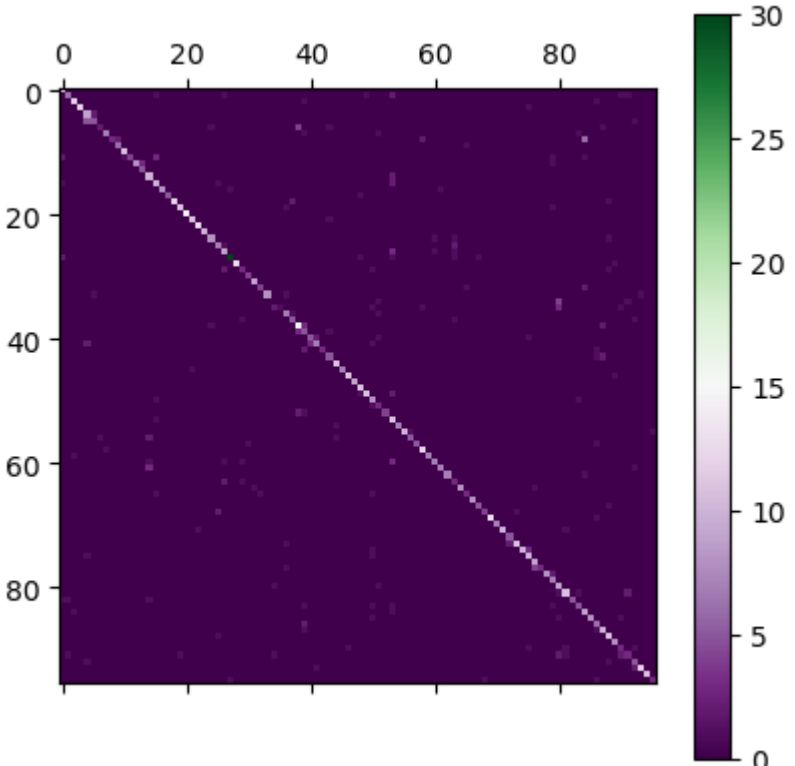
Următorul pas pe care l-am urmat pentru a îmbunătăți algoritmul a fost introducerea unui layer de Dropout. Un astfel de layer funcționează prin *aruncarea* aleatorie a unor neuroni în timpul antrenării, cu scopul de a combate overfittingul. Parametrul dat acestui tip de layer reprezintă rate-ul de aruncare a unităților. Inițial l-am plasat pe acesta după ultimul Convolution Layer și apoi un al doilea înaintea ultimului layer Dense:

Dropout I	Dropout II	Acuratețe
0.25	-	0.675
0.69	-	0.736
0.75	-	0.75
0.72	-	0.765
0.65	-	0.73
31	-	0.69
-	0.72	0.748
0.50	0.25	0.746
0.72	0.25	0.778

În ultima parte, am adaugat nivele de Batch Normalization, pentru a normaliza valorile înainte de a fi introduse în funcția de activare. În final, modelul arată astfel:

Layer	Hiperparametrii
Conv2D	filters = 32, kernel = (3,3), activation=gelu, input_shape=(64, 64, 3)
BatchNormalization	-
MaxPooling2D	pool_size = (2,2)
Conv2D	filters = 64, kernel = (3,3), activation= relu
Batch Normalization	-
MaxPooling2D	pool_size = (2,2)
Conv2D	filters = 128, kernel = (3,3), activation= gelu
Batch Normalization	-
MaxPooling2D	pool_size = (2,2)
Dropout	0.72
Flatten	-
Dropout	0.25
Dense	128, activation='relu'
Dense	96, activation='softmax'

Matricea de confuzie generată de acest algoritm:



Generarea submisiei:

Înaintea de a genera o submisie, am antrenat modelul pe un set mai mare de date, format din datele de train și cele de validare. Deoarece numărul de date a crescut, pentru această antrenare am crescut și numărul de epoci.

Algoritmul CNN va întoarce prin funcția *predict()* pentru fiecare imagine, un vector de lungime 96, reprezentând probabilitățile de apartenență la fiecare clasă. Astfel, pentru a genera submisia am folosit funcția *argmax()* pentru a găsi clasa cu cea mai mare probabilitate pentru fiecare imagine.

Referințe:

1. https://fmi-unibuc-ia.github.io/ia/
2. https://towardsdatascience.com/swish-booting-relu-from-the-activation-function-throne-78f87e5ab6eb
3. https://keras.io/examples/