Rogoza Raluca-Ioana
234

Documentație proiect

Deep Hallucination Classification

Descrierea proiectului

Acest proiect a constat în clasificarea unor imagini în 96 de clase distincte, numerotate de la 0 la 95.

Setul de date

- **train**: 12000 de imagini cu etichetele corespunzătoare
- **validation**: 1000 de imagini cu etichetele corespunzătoare
- **test**: 5000 de imagini

Modele create

- KNN
- CNN

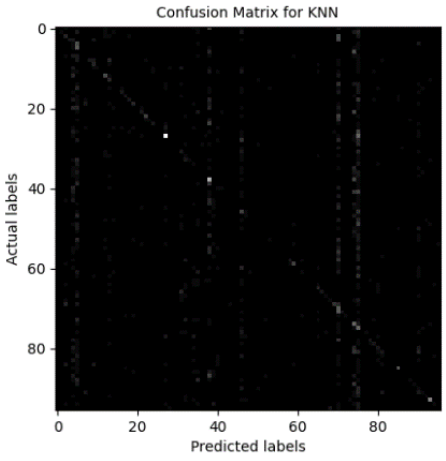
Pentru ambele modele prezentate mai jos am citit datele folosind un reader pentru fișiere CSV, iar pentru încarcarea imaginilor din folderele corespunzătoare am folosit funcția „open” din librăria PIL(Python Imaging Library).

KNN

Acest algoritm are rolul de a clasifica puncte în funcție de cei mai apropiați k vecini ai acestora. Se calculează distanța dintre pixelii imaginilor cu distanta euclidiana sau distanta Manhattan. La final este returnată eticheta majoritară a acestor vecini.

Am rulat algoritmul cu număr diferiți de vecini, dar cel mai bun scor l-am obținut pentru k = 110, știind că cel mai bun număr de vecini este aproximativ rădăcina pătrată din numărul de date din setul de train.

Matricea de confuzie pentru k = 110



Rogoza Raluca-Ioana
234

Am observat că cel mai bine clasifică atunci când utilizez metrica L2 și am obținut următoarele valori pentru 110 vecini:

- Accuracy score: 0.171
- Precision score: 0.2566615363979803
- Recall score: 0.15429691835941836

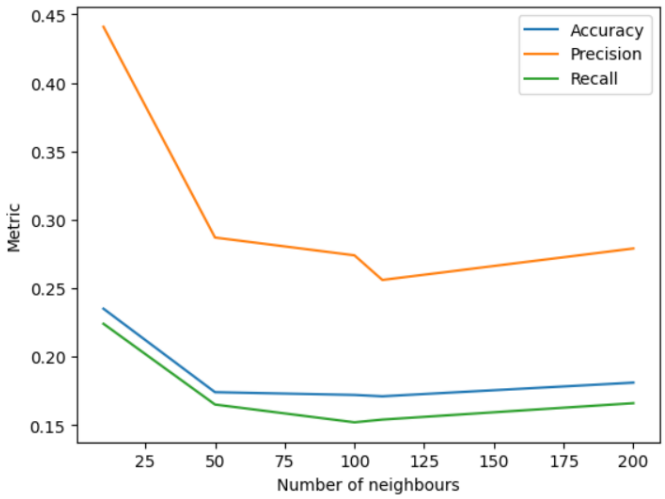
În concluzie, acuratețea maximă pe care am putut să o ating cu acest model este de 1.171.

Grafic accuracy, precision, recall în funcție de numărul de vecini

Accuracy = numărul de imagini etichetate corect / numărul total de imagini

Precision = numărul de imagini true positive / (numărul de imagini true positive + numărul de imagini false positive)

Recall = numărul de imagini true positive / (numărul de imagini true positive + numărul de imagini false negative)



CNN

CNN-urile (rețelele neurale convoluționale) sunt folosite pentru a procesa pixelii imaginilor, ajutând la clasificarea acestora. Ele sunt formate din mai multe layere care realizează operații de convoluție.

Pentru a se potrivi modelului folosit, bazat pe arhitectura modelului AlexNet, am redimensionat imaginile pentru a avea forma (227, 227, 3).

Layerele folosite în modelele mele care care se bazează pe arhitectura modelului AlexNet:

- Conv2D: la acest layer se face un produs scalar dintre 2 matrice, prima este cea care conține datele imaginilor și a doua este reprezentată de un filtru și rezultatul este o matrice de trăsături;

- MaxPool2D: alege cel mai mare număr din regiunea vizată și este folosit pentru a reduce dimensiunile input-ului;
- BatchNormalization: este normalizat input-ul din fiecare layer astfel încât să media să ajungă la valoarea 0, iar dispersia la valoarea 1; cu ajutorul acestui layer rețeaua se va antrena mai rapid;
- Flatten: acest layer este utilizat pentru a transforma input-ul multidimentional într-un input 1-dimensional; trebuie utilizat înainte de layerul dense;
- Dense: în acest layer fiecare neuron primește input de la toți neuronii din stratul de anterior;
- Dropout: în acest layer se renunță la unii neuroni în funcție de o probabilitate, adică neuronii vor fi setați la 0 pentru a reduce overfitting-ul.

Hiperparametrii folosiți:

- Pentru layerele de convoluție:
  - filtre: 96, 128, 256, 256, 512;
  - kernel: (11, 11), (5, 5), (3, 3), (3, 3), (3, 3);
  - stride: reprezintă pasul pe care-l folosim pentru a trece kernelul peste input; pentru primul layer am folosit (4, 4), iar pentru celelalte (1, 1);
  - funcții de activare: relu – ia maximul dintre 0 și valoarea de input, adică valorile negative le va converti în valori de 0;
  - bordura: reprezintă câți pixeli vor fi adaugați pentru o imagine atunci când este procesată de un kernel; pentru primul layer am ales „valid” (fără bordură), iar pentru celelalte „same” (bordură de 1);
  - pentru primul layer am selectat forma input-ului (227, 227, 3) pentru a se potrivi modelului;
- Pentru layerele de pooling:
  - dimensiune: (3, 3);
  - stride: (2, 2);
- Pentru layerele dense:
  - folosesc 2048 de unități pentru primele 2, iar pentru al treilea 96, reprezentând numărul de clase folosite pentru etichetarea imaginilor;
  - funcții de activare: relu pentru primele 2, iar pentru al treilea softmax care convertește pentru fiecare etichetă probabilitatea ca rezultatul să aibă acea etichetă;
- Pentru layerele de dropout:
  - am lăsat probabilitatea la 0.5.

Compilarea modelului:

- funcția de loss: sparse_categorical_crossentropy – folosită pentru clasificarea în mai multe clase; aici label-urile sunt reprezentate prin numere întregi;
- optimizator: keras.optimizers.SGD(learning_rate=0.01);
- metrica: acuratețe.

Versiunea finală a modelului

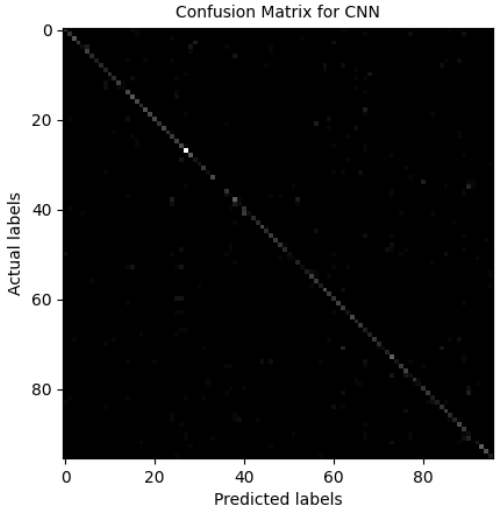
Layer	Hiperparametri
Conv2D	filters=96, kernel_size=(11, 11), strides=(4, 4), activation='relu', padding='valid', input_shape=(227, 227, 3)
MaxPool2D	pool_size=(3, 3), strides=(2, 2)
BatchNormalization	Default
Conv2D	filters=128, kernel_size=(5, 5), strides=(1, 1), activation='relu', padding='same'
MaxPool2D	pool_size=(3, 3), strides=(2, 2)
BatchNormalization	Default
Conv2D	filters=256, kernel_size=(5, 5), strides=(1, 1), activation='relu', padding='same'
Conv2D	filters=256, kernel_size=(5, 5), strides=(1, 1), activation='relu', padding='same'
Conv2D	filters=512, kernel_size=(5, 5), strides=(1, 1), activation='relu', padding='same'
BatchNormalization	Default
MaxPool2D	pool_size=(3, 3), strides=(2, 2)
Flatten	Default
Dense	2048, activation=’relu’
Dropout	0.5
Dense	2048, activation=’relu’
Dropout	0.5

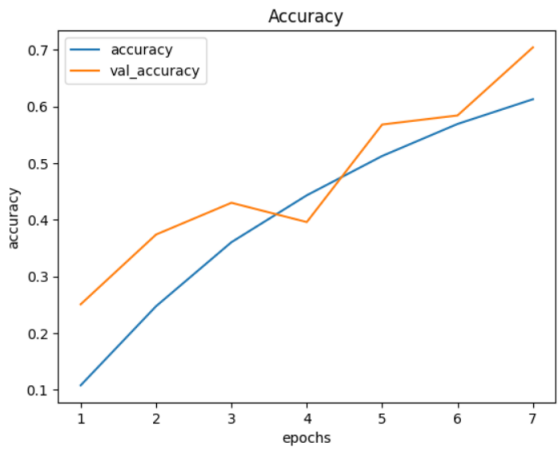
Am obținut următoarele valori pentru modelul meu:

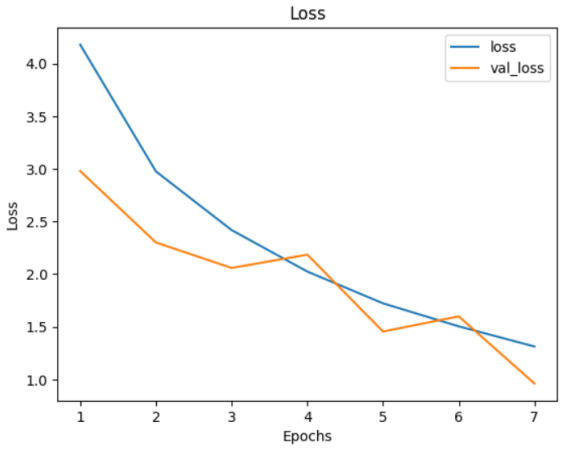
- Accuracy score: 0.70
- Precision score: 0.695
- Recall score: 0.658

Rogoza Raluca-Ioana
234

Matricea de confuzie pentru CNN







Am testat acest model cu mai multe rate de învățare pentru optimizatorul SGB.

- lr = 0.1 => accuracy = 0.013
- lr = 0.01 => accuracy = 0.70
- lr = 0.001 => accuracy = 0.54

Am testat și cu optimizatorul Adam, dar acuratețea ajungea abia la 0.4, deci am stabilit că optimizatorul SGB este potrivit pentru modelul meu.

Am testat modelul și cu diferite probabilități de dropout. Am observat că la 0.2 eroarea pe datele de validare era mai mare decât cea pe datele de test(overfitting), iar la 0.8 se realiza underfitting.

Rogoza Raluca-Ioana
234

Dropout 0.2 -> overfitting

Epoca	Loss	Accuracy	Val_loss	Val_accuracy
1	3.2934	0.2245	2.7559	0.2660
2	1.9733	0.4488	2.4814	0.3930
3	1.4140	0.5803	1.7146	0.5530
4	1.0481	0.6822	2.0553	0.5010
5	0.7804	0.7545	0.6937	0.7920
6	0.5859	0.8177	3.9004	0.2860
7	0.4207	0.8628	0.5242	0.8370

Dropout 0.8 -> underfitting

Epoca	Loss	Accuracy	Val_loss	Val_accuracy
1	5.5894	0.0211	4.5582	0.0370
2	4.5518	0.0279	4.5823	0.0300
3	4.5280	0.0300	4.4479	0.0360
4	4.5030	0.0313	4.4451	0.0450
5	4.4675	0.0336	4.5399	0.0370
6	4.4544	0.0361	4.4317	0.0510
7	4.4216	0.0418	4.4167	0.0520

În concluzie, acuratețea maximă pe care am putut să o ating cu acest model a fost de 0.7.

Model încercat

Layer	Hiperparametri
Conv2D	filters=96, kernel_size=(11, 11), strides=(4, 4), activation='relu', padding='valid', input_shape=(227, 227, 3)
MaxPool2D	pool_size=(3, 3), strides=(2, 2)
BatchNormalization	Default
Conv2D	filters=256, kernel_size=(5, 5), strides=(1, 1), activation='relu', padding='same'
MaxPool2D	pool_size=(3, 3), strides=(2, 2)
BatchNormalization	Default

Conv2D	filters=256, kernel_size=(5, 5), strides=(1, 1), activation='relu', padding='same'
Conv2D	filters=256, kernel_size=(5, 5), strides=(1, 1), activation='relu', padding='same'
Conv2D	filters=512, kernel_size=(5, 5), strides=(1, 1), activation='relu', padding='same'
BatchNormalization	Default
MaxPool2D	pool_size=(3, 3), strides=(2, 2)
Flatten	Default
Dense	2048, activation=’relu’
Dropout	0.5
Dense	2048, activation=’relu’
Dropout	0.5

Pe acest model am obținut acuratețea 0.53. Am încercat și cu dropout de 0.4 și am obținut acuratețea 0.518, mai mică decât cea obținută pe celălalt model, așa că am renunțat la el.