Păunică Cosmin

Grupa 243

**Documentația abordărilor în competiția Monochrome Dreams Classification**

În fișierul basics.py am implementat funcții pentru citirea datelor și pentru scrierea predicțiilor în formatul cerut de competiție, precum și constante ce reprezintă numărul de imagini, dimensiunea unei imagini și numărul de clase.

Pentru măsurarea performanței modelelor (precizie, matrice de confuzie) am folosit funcțiile accuracy\_score și confusion\_matrix din sklearn.metrics.

Rețele neuronale convoluționale

Cel mai bun scor privat în competiție (0.89866 scor privat) l-am obținut folosind o rețea neuronală convoluțională (modelul Sequential din tensorflow.keras.models), un model foarte bun în recunoașterea modelelor din imagini, mulțumită convoluției. Am construit rețeaua după structura conv-conv-pool-conv-conv-pool și un număr din ce în ce mai mare de filtre pe măsură ce se avansează în adâncimea rețelei1. Mai precis:

* 2 straturi convoluționale (Conv2D) cu 64 de filtre de dimensiune 3x3, în care imaginea este parcursă cu niște filtre de la stânga la dreapta și de sus în jos, pentru accentuarea modelelor și contururilor
* 1 strat de pooling prin selecția maximului (MaxPool2D), cu dimensiunea pool-ului 2x2, care reduce dimensiunea imaginii, păstrând doar pixelul cu valoarea maximă dintr-un pătrat 2x2
* 1 strat de dropout (Dropout) cu rata 0.25
* 2 straturi convoluționale cu 128, respectiv 256 de filtre de dimensiune 3x3
* 1 strat de pooling prin selecția maximului, cu dimensiunea pool-ului 2x2
* 1 strat de dropout cu rata 0.25
* 1 strat de flatten (Flatten) pentru transformarea input-urilor 2D în array-uri 1D
* 1 strat dens (Dense) cu 1024 de neuroni, care arată ca un strat dintr-o rețea neuronală clasică
* 1 strat de dropout cu rata 0.3
* 1 strat dens cu 9 neuroni (strat de output)

Toate clasele pentru straturi sunt din tensorflow.keras.layers.

Pentru toate straturile convoluționale și pentru primul strat dense, am folosit funcția de activare ReLU (recomandată la curs). Pentru stratul de output, am folosit softmax. Straturile de dropout au fost adăugate pentru regularizare. Am început antrenarea modelului fără straturi de dropout, obținând o precizie foarte aproape de 1 pe datele de antrenare, dar slabă pe datele de validare. Am crescut treptat numărul și rata straturilor de dropout până când am obținut cel mai bun rezultat.

Funcția de pierdere folosită a fost Mean Squared Error, iar optimizatorul, Adam (Adaptive Moment Estimation, un algoritm care combină avantajele a două variații ale algoritmului stochastic gradient descent: Adaptive Gradient Algorithm și Root Mean Square Propagation2), cu rata de învățare 0.0005. Modelul a fost antrenat pe 20 de epoci, cu mărimea batch-ului 256.

Ca preprocesare, valorile pixelilor imaginilor au fost aduse în intervalul [0, 1] (prin împărțirea array-urilor de pixeli la 255), iar etichetele de antrenare și validare au fost codificate folosind one-hot encoding (funcția to\_categorical din tensorflow.keras.utils).

Cu această configurație, am obținut pe datele de validare precizia 0.8968 și matricea de confuzie:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Clasă\Predicție | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 493 | 7 | 5 | 3 | 24 | 3 | 11 | 14 | 10 |
| 1 | 9 | 487 | 3 | 5 | 7 | 2 | 2 | 12 | 0 |
| 2 | 5 | 8 | 461 | 15 | 20 | 19 | 2 | 3 | 0 |
| 3 | 6 | 2 | 8 | 531 | 5 | 8 | 5 | 8 | 5 |
| 4 | 16 | 4 | 8 | 18 | 478 | 6 | 3 | 14 | 7 |
| 5 | 2 | 1 | 16 | 11 | 7 | 511 | 2 | 10 | 1 |
| 6 | 9 | 2 | 1 | 2 | 2 | 5 | 547 | 1 | 11 |
| 7 | 9 | 8 | 12 | 18 | 3 | 15 | 8 | 444 | 3 |
| 8 | 14 | 0 | 0 | 4 | 2 | 4 | 17 | 4 | 532 |

Implementarea se găsește în fișierul cnn.py.

Am obținut și alte scoruri de peste 0.82 pe datele de validare și pe cele de test, folosind diferite configurații de straturi și diferiți hiperparametri pe modelul Sequential (de exemplu: conv-pool-conv-pool1, cu 32 de filtre pe fiecare strat convoluțional, un strat dens cu 64 de neuroni, fără straturi de dropout, antrenarea pe 15 de epoci în loc de 20, learning rate-ul pentru optimizatorul Adam 0.001 în loc de 0.0005 și batch size-ul de 32, care a dus la scorul de 0.8450 pe imaginile de validare).

Clasificatorul SVM

Un alt model încercat este clasificatorul liniar SVM (clasa SVC din sklearn.svm).

Cea mai bună precizie pe datele de validare (0.7628) și de testare (0.75360 scor privat) pe acest model am obținut-o punând hiperparametrul C=100 (am încercat și modele mai lejere – C=10, C=75 – dar și mai restrictive – C=1000 – toate oferind rezultate cel mult la fel de bune).

Imaginile au fost preprocesate folosind StandardScaler, din sklearn.preprocessing. Acesta scade din imagini media valorilor pixelilor și le împarte, apoi, la deviația standard a acestora. Media și deviația standard sunt calculate pe fiecare trăsătură a input-ului în parte (în cazul nostru, pe pixeli de pe aceeași poziție a fiecărei imagini).

Am obținut următoarea matrice de confuzie pe datele de validare:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Clasă\Predicție | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 368 | 22 | 18 | 11 | 42 | 4 | 33 | 41 | 31 |
| 1 | 17 | 447 | 12 | 8 | 6 | 5 | 5 | 22 | 5 |
| 2 | 16 | 29 | 389 | 13 | 34 | 27 | 4 | 18 | 3 |
| 3 | 23 | 16 | 18 | 434 | 20 | 17 | 12 | 19 | 19 |
| 4 | 37 | 21 | 22 | 23 | 404 | 9 | 3 | 26 | 9 |
| 5 | 7 | 6 | 20 | 25 | 16 | 458 | 7 | 19 | 3 |
| 6 | 27 | 13 | 13 | 8 | 8 | 6 | 474 | 6 | 25 |
| 7 | 37 | 16 | 19 | 26 | 24 | 18 | 7 | 365 | 8 |
| 8 | 28 | 5 | 3 | 9 | 5 | 6 | 40 | 6 | 475 |

Implementarea se află în fișierul svm.py.

Modelul K-Nearest Neighbors

Cea mai slabă precizie am avut-o la prima submisie: 0.48400 scor privat pe datele de testare și 0.4916 pe datele de validare. Acesta a fost obținut cu ajutorul modelului k-NN (clasa KNeighborsClassifier din sklearn.neighbors (un clasificator “leneș”: învățarea nu se face la început, pe datele de antrenare, ci la fiecare imagine căruia îi trebuie precizată eticheta), cu K=9 (numărul de vecini) și fără preprocesare. Am obținut următoarea matrice de confuzie pe datele de validare:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Clasă\Predicție | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 248 | 22 | 2 | 10 | 21 | 4 | 108 | 43 | 112 |
| 1 | 89 | 200 | 12 | 17 | 11 | 4 | 83 | 46 | 65 |
| 2 | 100 | 12 | 108 | 25 | 26 | 38 | 131 | 55 | 38 |
| 3 | 58 | 13 | 12 | 294 | 19 | 32 | 76 | 23 | 54 |
| 4 | 98 | 25 | 16 | 47 | 208 | 20 | 43 | 46 | 51 |
| 5 | 31 | 2 | 10 | 45 | 12 | 329 | 73 | 25 | 34 |
| 6 | 42 | 8 | 1 | 11 | 7 | 10 | 425 | 6 | 70 |
| 7 | 62 | 22 | 6 | 22 | 25 | 17 | 91 | 237 | 38 |
| 8 | 40 | 3 | 1 | 13 | 2 | 6 | 95 | 8 | 409 |

Implementarea se află în fișierul knn.py.

Bibliografie

1. Inspirat de la <https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-parameter-42efca01e5d7>
2. <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/#:~:text=Adam%20is%20a%20replacement%20optimization,sparse%20gradients%20on%20noisy%20problems>.