Monochrome Dreams Classification – Documentatie

Strimbeanu Luana | Grupa 243

1. **Clasificator CNN** – acuratete 94,22%

Preprocesare + Feature Extraction

Pentru preprocesare si feature extraction am folosit functia imread din biblioteca matplotlib, care proceseaza imaginile astfel incat fiecarui pixel sa ii revina o valoare din [0,1]. Pentru ca numpy array-urile returnate de imread pentru imaginile grayscale au shape (32, 32), am facut un reshape la (32, 32, 1), marcand astfel canalul monocromatic.

Metoda de clasificare folosita

Pentru clasificare am folosit un Convolutional Neural Network, care este un clasificator neliniar. Am ales CNN in detrimentul unei retele neuronale standard cum ar fi MLP(invatata la curs) deoarece CNN-ul are recunoastere spatiala datorita filtrelor din convolutii, pe cand MLP poate primi doar flat data. Miscarea filtrelor in CNN permite impartasirea parametrilor si a greutatilor, fiind invariata de locatie, deci modelul poate gasi pattern-uri oriunde in imagine. CNN, nefiind fully connected, nu are la fel de multi parametri ca MLP, eliminandu-se astfel redundante is ineficienta.

Pentru procesarea cu CNN am instalat biblioteca TensorFlow si am folosit modulul Keras. Pentru procesare mai rapida, am instalat si CUDA/cuDNN apeland astfel la puterea de calculare a GPU-ului, avand la dispozitie mai multe threaduri.

Etape de testare

Initial am facut doua convolutii de cate 150 de filtre fiecare (media dintre 192 si 128) deoarece nu am stiut exact cate convolutii sa aleg initial, asa ca am testat o valoare de mijloc urmand eventual sa ma indrept catre 192 sau 128, in functie de rezultat. Pentru kernel size am ales valoarea 3. Intre straturile convolutionale am facut MaxPooling. Dupa

ce am terminat cu convolutiile am adus datele din 2D intr-o singura dimensiune cu Flatten. Pentru straturile dense am ales tot un numar de mijloc, 700 (intre 1000 si 500) tot cu acelasi scop, de a vedea o medie si a afla ulterior catre care capat al intervalului propus de mine initial ar trebui sa ma deplasez. Am creat trei astfel de straturi cu functiile de activare pentru straturile ascunse: tanh relu si sigmoid. Pentru stratul de output am folosit softmax cu 9 canale. Pentru optimizare am folosit Adamax. Modelul a obtinut acuratete 88%. Ulterior am scos stratul cu functia de activare tanh, am adaugat Dropout de 15% ca sa previn overfitting-ul si am atins o acuratete de 89-90%.

In acest punct am realizat ca „holy grail”-ul CNN-ului este filtrarea datelor. Am marit kernel size-ul la 5 si am inceput sa adaug Dropout de 30% dupa fiecare layer, ca sa previn overfit-ul. Functia sigmoida se comporta si ea ca un fel de filtru, asa ca am pastrat-o. Am mai normalizat odata datele cu BatchNormalization dupa ce calculez convolutiile, si modelul meu a atins acuratete 92%. Dupa ce am mai adaugat doua layere dense de cate 700 de neuroni fiecare, unul cu functie de activare relu si altul cu sigmoid si am marit Dropout-ul la 40% - am considerat ca daca mai adaug straturi, ar trebui sa filtrez mai mult datele ca sa previn overfitting-ul – modelul meu a ajuns la acuratete 94,22%. Cu Dropout 30%, modelul atinge doar 93,52% acuratete.

Matricea de confuzie si acuratetea pentru modelul final:

```
0.942
[[510.   1.   2.   4.  21.   0.   8.  12.  12.   0.]
 [  6. 507.   1.   3.   2.   2.   0.   4.   2.   0.]
 [  3.   4. 490.   6.  12.  12.   2.   3.   1.   0.]
 [  6.   0.   2. 551.   2.   4.   1.   3.   9.   0.]
 [ 13.   1.   2.   9. 514.   5.   1.   4.   5.   0.]
 [  2.   0.   7.   7.   3. 536.   1.   5.   0.   0.]
 [  4.   0.   2.   3.   0.   2. 560.   3.   6.   0.]
 [  4.   2.   6.  13.   5.   8.   3. 477.   2.   0.]
 [  1.   0.   0.   2.   1.   2.   6.   0. 565.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]]
```

2. **Clasificator SVM** – acuratete 73,5%

Preprocesare + Feature Extraction

Pentru preprocesare si feature extraction am folosit aceeasi metoda ca la SVM, anume imread, doar ca la citire am transformat fiecare imagine intr-un array de 32*32 pentru a acomoda modelul SVM.

Metoda de clasificare folosita

SVM-ul este un clasificator care alege un hiperplan in functie de care separa datele pe clase. Pentru a clasifica datele, modelul compara clasele candidat one-versus-one, stabilind un hiperplan intre fiecare clasa cu fiecare dintre celelalte clase ramase. Mai exista si tratarea one-versus-all, insa clasificatorul SVC din modulul SVM din biblioteca SKLearn pe care l-am folosit implementeaza o arhitectura one-versus-one. Hiperplanul ales poate fi controlat prin parametrul C prin care putem controla cat de permisiv este modelul. Prin parametrul kernel decidem cum va arata hiperplanul, clasificatorul putand astfel fi si liniar si neliniar.

Etape de testare

Initial am ales un SVC cu C = 1.0 si kernel liniar. Acest model a atins insa acuratete de doar 60%. Ulterior am lasat SVC-ul cu parametrii default: C = 1.0 si kernel neliniar de tip RBF (Radial Basis Function) si am obtinut o acuratete de 73,5%.

```
0.735
[[337.  27.  20.  18.  46.   3.  33.  44.  42.   0.]
 [ 23. 416.  10.  13.  12.   4.   8.  29.  12.   0.]
 [ 15.  27. 383.  16.  35.  26.  10.  17.   4.   0.]
 [ 25.  16.  13. 424.  22.  21.  16.  18.  23.   0.]
 [ 36.  18.  25.  29. 389.  12.   5.  25.  15.   0.]
 [  9.  10.  18.  25.  16. 442.   6.  31.   4.   0.]
 [ 31.  12.  12.  11.   6.   5. 462.   6.  35.   0.]
 [ 38.  15.  20.  29.  31.  18.  14. 344.  11.   0.]
 [ 23.   4.   6.   8.   3.   8.  42.   5. 478.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]]
```