Inteligență Artificială

Documentație – Monochrome Dreams Classification

Student*:* *David Albert-Constantin*

Grupa: *241*

Model: *SVM.SVC Classifier*

Una dintre metodele folosite pentru concursul ***Monochrome Dreams Classification*** este reprezentată de clasificatorul SVM.SVC, implementat cu ajutorul bibliotecii sklearn.

Pentru citirea datelor am parcurs fiecare fișier in format txt si am adaugat in liste, tupluri cu id-ul fiecarei imagini si eticheta. Dupa care, pentru fiecare element din aceste liste am luat imaginea corespunzătoare cu ajutor funcției Image.open() din biblioteca PIL și am convertit-o într-un np.array folosindu-mă de funcția Image.getdata() care întoarce conținutul imaginii sub forma unei secvențe de pixeli.

Știind că valoare minimă a pixelilor este 0, iar cea maximă este 255 am ales sa normalizez datele împărțind valorile pixelilor la 255 pentru a le aduce in intervalul [0,1].

Mașinile cu vectori suport (SVM) reprezintă un algoritm de învățare supervizată pentru clasificare și regresie, prin care se alege hiperplanul cu o margine separatoare maximă.

În cadrul implementării, am utilizat modeul SVC cu metodele SVC.fit() și SVC.predict() și la care am modificat parametrii C și kernel. Funcțiile kernel sunt utilizate când datele nu sunt liniar separabile. Astfel, ele scufundă datele într-un spațiu Hilbert cu mai multe dimensiuni în care se caută relațiile liniare. Funcțiile kernel pe care le-am testat sunt RBF și poly. Parametrul C este un parametru de penalitate pentru eroare. Pentru un C mare, hiperplanul ales are o margine mai mică ceea ce poate duce la overfitting, iar pentru un C mic, hiperplanul ales are o margine mai mare ceea ce poate duce la underfitting.

Pentru a obține informații despre performanța modelului am utilizat metrics.confussion_matrix din biblioteca sklearn pentru a determina matricea de confuzie. Elementele de pe diagonala principala a matricei de confuzie reprezintă evaluările corecte pentru etichete. Pe baza matricei de confuzie se pot determina anumite proprietăți precum precizia, recall-ul, acuratețea . O parte din combinațiile de parametrii pe care am testat modelul sunt:

- Kernel: RBF, C = 1, 10, 15
- Kernel: poly, C = 1, 10, 15

Rezultate pe datele de validare:

Kernel: RBF, C = 1

Acuratețe: 0.735

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

Kernel: RBF, C = 10

Acuratețe: 0.7554

```
Confussion matrix:
[[365  22  19  12  43   4  34  41  30]
 [ 19 439  12  10   6   6   8  23   4]
 [ 16  30 389  14  34  27   4  16   3]
 [ 25  16  19 423  22  20  11  21  21]
 [ 33  23  22  25 400   9   4  27  11]
 [  7   7  20  25  17 455   7  20   3]
 [ 26  14  11  11   9   6 465   8  30]
 [ 41  15  20  25  23  16   7 365   8]
 [ 27   5   4   9   5   6  38   7 476]]
```

Kernel: RBF, C = 15

Acuratețe: 0.756 (Best)

```
Confussion matrix:
[[367  22  19  12  44   4  34  37  31]
 [ 18 441  12  10   6   6   8  21   5]
 [ 18  32 386  14  33  26   4  17   3]
 [ 24  15  19 430  22  19  11  19  19]
 [ 38  22  21  23 400   9   4  27  10]
 [  7   7  21  25  17 455   7  19   3]
 [ 25  13  11  11   9   6 467   8  30]
 [ 42  14  19  28  23  17   7 362   8]
 [ 31   5   3  10   4   8  38   6 472]]
```

Kernel: Poly, C = 1

Acuratețe: 0.71

```
Confussion matrix:
[[391  24  13   8  42   5  24  31  32]
 [ 36 404  12  12   9   6  13  26   9]
 [ 25  25 374  19  42  22   7  15   4]
 [ 43  12  19 404  20  24  17  21  18]
 [ 57  24  29  29 368  11   4  21  11]
 [ 10  11  32  35  21 425   5  18   4]
 [ 51  15  18  11   6   4 445   9  21]
 [ 64  20  19  29  29  14  11 329   5]
 [ 62   8   2  13   4  11  39  11 427]]
```

Kernel: Poly, C = 10

Acuratețe: 0.70

```
Confussion matrix:
[[366  24  25  13  42   5  32  34  29]
 [ 33 395  18  12  12   6  14  29   8]
 [ 17  33 386  13  38  24   6  13   3]
 [ 29  18  22 401  20  28  16  26  18]
 [ 50  32  39  24 358  11   2  27  11]
 [  6   9  47  22  13 429   5  26   4]
 [ 46  19  15  10   8  10 440   9  23]
 [ 45  20  20  35  24  20  11 338   7]
 [ 52  13   4  11   3  10  46  11 427]]
```

Kernel: Poly, C = 15

Acuratețe: 0.7092

```
Confussion matrix:
[[366  25  22  15  42   4  28  36  32]
 [ 30 398  19  13  13   7  13  27   7]
 [ 17  34 387  11  37  23   6  14   4]
 [ 30  17  25 399  18  28  13  29  19]
 [ 49  29  37  25 364  10   1  26  13]
 [  6  12  45  22  12 428   6  26   4]
 [ 48  19  19  11   7  11 434  10  21]
 [ 44  21  20  33  25  22  11 339   5]
 [ 44  13   4  11   5  11  46  12 431]]
```

Model: *Convolutional neural network*

O altă metodă utilizată în cadrul concursului este reprezentată de rețelele neuronale convoluționale. Pentru implementarea rețelei am utilizat bibliotecile Tensorflow si Keras.

Citirea datelor am realizat-o in acelasi mod cu cea de la modelul anterior.

Rețelele neuronale sunt formate din perceptroni care pot fi dispuși pe mai multe straturi.

Straturile pot fi:

- de intrare – unde perceptronii primesc date de intrare din exterior
- ascunse – unde perceptronii primesc ca date de intrare rezultatul stratului anterior
- de iesire – care produc rezultatul pentru datele de intrare

Rețelele neuronale convoluționale sunt formate dintr-o secvență de straturi convoluționale la care se aplică anumite funcții de activare. Aceste straturi aplică filtre convoluționale pe un tensor de intrare, mai exact filtrele sunt trecute peste imagine și calculeaza produsul scalar pentru fiecare poziție. Filtrele, in general, au dimensiunea spațială impară, pentru a fi centrate pe o anumită poziție

Funcțiile de activare au drept scop introducerea de non-linearitate pentru datele de ieșire ale neuronilor.

Un alt tip de strat ce poate fi găsit în aceste rețele este stratul de pooling, care reduce dimensiunea spațială și operează pe fiecare matrice de activare returnată de stratul convoluțional.

Modelul implementat este unul secvențial, adăugat cu ajutorul funcției models.Sequential() din biblioteca Keras. Am utilizat straturi convoluționale (Conv2D din Keras) cu număr diferit de filtre (puteri ale lui 2, ex: 32,64,128). Dimensiunea aleasă pentru filtre este (3x3), iar pentru funcția de activare am ales RELU (Rectified Linear Unit) care preia răspunsul pozitiv al filtrelor.

Pentru reducerea dimensiunii spațiale am folosit straturi de MaxPooling2D care fac parte din biblioteca Keras și iau valoarea maximă pentru un anumit pool. Pentru parametrii am utilizat pool_size = (2, 2), iar parametrii strides si padding sunt default. Parametrul strides specifică pentru fiecare pas cât de mult se deplasează pool-ul, iar padding-ul daca este “same” atunci adaugă padding input-ului, alftel daca este “valid”, nu adaugă.

Un alt strat utilizat în rețea este Batch Normalization, care preia iesirile din stratul anterior și le normalizeaza astfel încât să aibă media 0 si dispersia 1. Am inserat acest tip de strat atât după straturile dense, cât și după cele convoluționale.

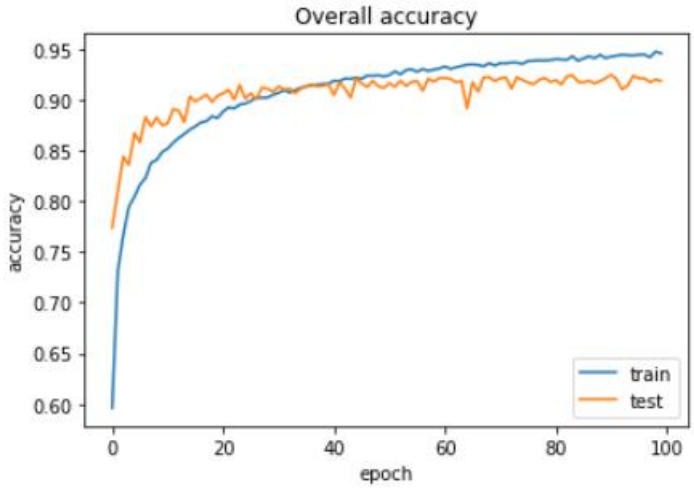
Între straturile convoluționale si cele dense (în care fiecare neuron primește date de intrare de la fiecare neuron din stratul anterior) am folosit stratul Flatten care aplatizează datele de intrare.

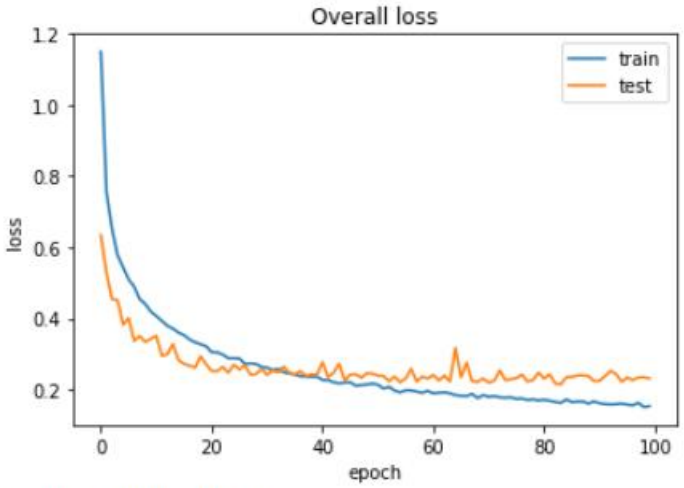
Pentru a evita supraînvățarea modelului, am folosit Dropout care elimină în mod aleator un anumit număr de neuroni din rețea.

Compilarea modelului am realizat-o cu optimizare “adam”, iar pentru loss am folosit funcția SparseCategoricalCrossentropy() deoarece avem un număr mare de etichete. Pentru antrenare am folosit functia fit(), pentru evaluare pe datele de validare, funcția evaluate(), iar pentru predicția datelor de test, funcția predict().

Pentru plotarea graficelor pentru loss si acuratețe am utilizat funcția plot() din biblioteca matplotlib.

Rezultate pe modelul final (cel din fisierul .py):





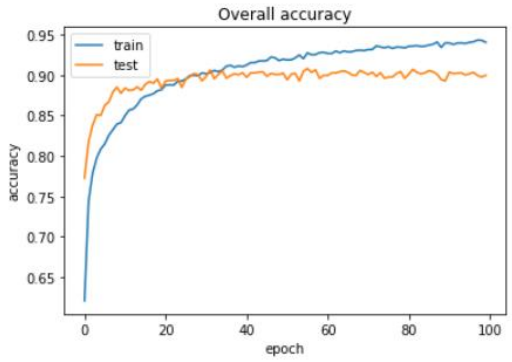

```
Matricea de confuzie:
[[483   1   3   4  21   1  14  19  24]
 [  6 491   4   5   4   2   2   8   5]
 [  7   2 474  12   9  16   2  11   0]
 [  4   2   2 526   7   8   2  11  16]
 [ 14   2   5  11 496   6   3  10   7]
 [  1   0   7   6   4 529   4   8   2]
 [  6   2   0   1   2   3 561   2   3]
 [  3   3   5  13   0  11   7 475   3]
 [  3   0   0   2   2   2  10   1 557]]
```

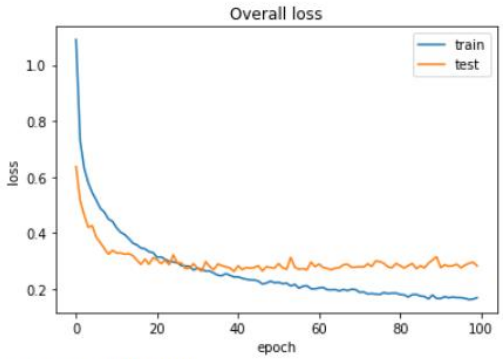
```
Acc 0.91839998960495
```

Un alt model pe care am testat este urmatorul:

```
cnnModel = models.Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 1)),
    BatchNormalization(),   #  layer de normalizare
    Dropout(0.2),   # layer de regularizare

    Conv2D(64, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D(pool_size=2),  # layer de pooling pentru a reduce
    Dropout(0.3),

    Conv2D(128, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D(pool_size=2),
    Dropout(0.4),

    Flatten(),
    Dense(128, activation='relu'),      # layer complet
    BatchNormalization(),
    Dropout(0.4),
    Dense(64, activation='relu'),
    BatchNormalization(),
    Dense(9, activation="softmax")  # layer de output cu activare
])
```

Rezultate:





Acc 0.8996000289916992

```
Matricea de confuzie:
[[472   4   9   8  33   0  10  15  19]
 [  7 484   7   4   5   6   1  11   2]
 [  3   6 469  10  17  18   2   8   0]
 [  2   1  13 505  17  11   2  11  16]
 [ 12   4  15  10 484   9   4   8   8]
 [  1   0  13  10   6 516   3  11   1]
 [  9   2   3   1   1   1 555   2   6]
 [  7   3   6  14   5  15   7 459   4]
 [  8   1   0   3   1   2   7   1 554]]
```

Pentru modelele prezentate am modificat hiperparameterii precum filtrele, rate-ul dropout-ului, funcțiile de activare astfel încât să obțin rezultate cat mai bune.