Popa Iulia-Andreea – Grupa 241
Invatare automata – Inteligenta artificiala

Monochrome Dreams Classification – Documentatie proiect

In cadrul acestei competitii am abordat doua modele de clasificare:

1. Support Vector Machine (SVM)
2. Convolutional Neural Network (CNN)

In ambele cazuri, am citit seturile de date folosind modulele *os* si *cv2*. In cadrul unei functii, cu functia *listdir()* am accesat fiecare fisier dintr-un director dat ca parametru si am citit imaginile cu functia *imread()*. Pentru etichetele imaginilor, am citit din fisierele de tip text folosind functia *open()* si le-am salvat in *numpy array*.

Pentru afisarea datelor obtinute in urma clasificarii, am folosit un fisier de tip CSV in care am scris rezultatele pe rand cu ajutorul functiei *writerow()*, pe doua coloane: id si label.

1. Support Vector Machine (SVM)

Este un clasificator binar de invatare supervizata folosit fie pentru clasificare, fie pentru regresie. Acesta clasifica doua clase construind un hiperplan intr-un spatiu n-dimensional (n reprezinta numarul de proprietati ale obiectelor) ce maximizeaza separarea marginii dintre clase.

Dupa citirea imaginilor, pentru a extrage trasaturile (pixelii), am folosit functia *flatten()* pentru a liniariza datele schimbandu-le dimensiunea din (32,32,3) in (1024). Apoi, prin normalizare, impartind vectorul numpy la 255, valorile pixelilor vor trece din numere intregi din intervalul [0,255] la numerele reale cuprinse intre [0,1], astfel imbunatatind timpul de executie al antrenamentului si eficienta acestuia.

Am implementat acest clasificator utilizand modelul SCV din libraria sklearn.svm . Pentru configuratia implicita a modelului am obtinut o acuratete de 0.74720.

```
[[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]]
```

Putem observa in matricea de confuzie, pe diagonala, care elemente au fost prezise corect pentru fiecare eticheta.

Totusi, modificarea parametrilor poate eficientiza executia modelului, acesta ajungand la o acuratete mai buna. In acest sens, parametrul de regularizare C sugereaza cat de mult admite modelul sa evite clasificarea gresita a datelor din setul de antrenare (daca valoarea lui C este prea mare, hiperplanul va avea marginea prea mare si vom ajunge la underfitting; daca este prea mica, se produce overfitting), iar valoarea coeficientului pentru kernel non-liniar, gamma, isi mareste valoarea odata cu strictetea de clasificare a modelului.

Astfel, cu kernel 'rbf', C = 50, gamma = 0.0015 am reusit sa cresc acuratetea la 0.7526.

2. Convolutional Neural Network (CNN)

Retele neuronale convolutionale folosesc conectivitatea locala care implica legarea fiecarui neuron de o regiune locala a inputului, spre deosebire de MLP (Multilayer Perceptron) care, de exemplu, leaga neuronii din stratul urmator de fiecare trasatura din input. De aceea, CNN-urile sunt mai eficiente din punct de vedere al spatiului si timpului.

Acest clasificator foloseste straturi care stocheaza informatii despre trasaturile imaginilor, in cazul nostru despre pixeli.

Pentru implementarea algoritmului am folosit libraria *tensorflow* si API-ul keras cu modelul *Sequential()* si structurile de date *layers* si *models*.

Initial, am inceput cu un model cu 4 straturi convolutionale 2D cu numar diferit de filtre din care straturile invata, aceeasi dimensiune a kernelului, parametru (functia) de activare ReLU (rectified linear unit) definit ca y=max(0,x) (fiind usor de calculat, antrenarea si executia dureaza mai putin). Un strat convolutional creeaza un nucleu de convolutie pentru extragerea trasaturilor inputului ca, mai apoi, sa genereze un strat de iesire. Pentru fiecare strat convolutional de inceput am mentionat input_shape-ul obiectelor (32,32,3).

Dupa doua straturi convolutionale am adaugat si cate un strat de pooling pentru a face modelul mai robust (acesta calculeaza valoarea maxima pentru fiecare zona din features map si rezuma caracteristicile inputului in zone).

Dupa fiecare strat convolutional am adaugat un strat de dropout ce are ca scop evitarea overfittingului prin setarea random la 0 a unitatilor din input cu o frecventa data. Dupa cum se observa si in model, valoarea dropout-ului creste odata cu numarul de filtre ale stratului convolutional.

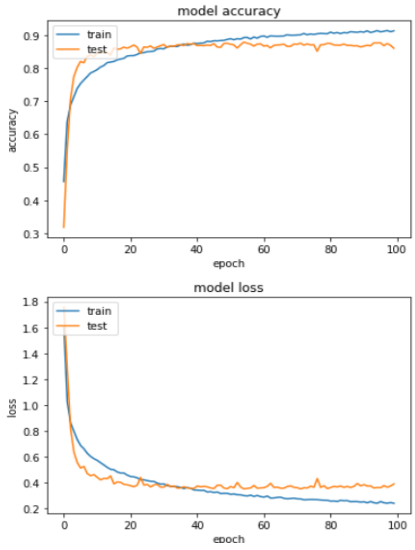
De asemenea, straturile de BatchNormalization contribuie la mentinerea mediei outputului aproape de 0 si a deviatiei standard aproape de 1, astfel facand retelele neuronale mai stabile prin recentrare si redimensionare.

La final, dupa liniarizarea vectorului prin *Flatten()*, am adaugat un strat de *Dense* (cel mai de baza strat al unei retele neuronale) ce leaga neuronii de toate ieșirile din stratul anterior, fiecare neuron furnizând o ieșire către stratul următor. Dupa Dropout() si BatchNormalization() am adaugat stratul final de Dense de 9 neuroni (cate unul pentru fiecare label) cu functia de activare *softmax* care transforma vectorul real intr-un vector de probabilitati cu valori, evident, intre 0 si 1, astfel rezultatul putand fi interpetrat ca o distributie de probabilitati.

Am creat modelul cu functia compile, l-am antrenat cu functia fit pe vectorii normalizati, 100 de epoci si batch_size = 256, iar apoi am prezis pe baza acestuia etichetele imaginilor de validare, obtinand o acuratete de 0.8733.

```
cnn = models.Sequential([
    layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.2),
    layers.BatchNormalization(),

    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu'),
    layers.Dropout(0.2),
    layers.BatchNormalization(),

    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.25),
    layers.BatchNormalization(),

    layers.Conv2D(filters=128, kernel_size=(3, 3), activation='relu'),
    layers.Dropout(0.4),
    layers.BatchNormalization(),

    layers.Flatten(),
    layers.Dense(256, activation='relu'),
    layers.Dropout(0.2),
    layers.BatchNormalization(),
    layers.Dense(9, activation='softmax')
])
```

```
[0.3879695534706116, 0.8600000143051147]
```

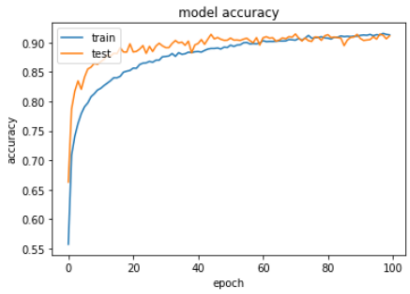


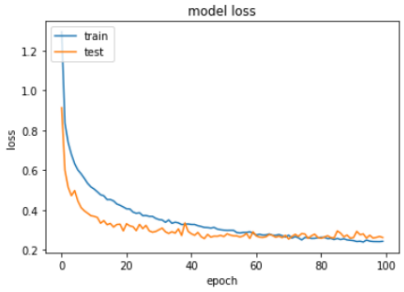
Totusi, am reusit sa optimizez acest model modificand straturile si parametrii.

Pe un model cu 3 straturi convolutionale, dar cu straturile de BatchNormalization si Dropout interschimbate si un strat in plus de Dense la final, construit cu 100 de epoci si batch_size=64, am reusit sa obtin o acuratete de 0.916. Desi aceste optimizari pot fi benefice, in cazul acestui model s-a produs o usoara instabilitate deoarece acuratetea a scazut la 0.905 pe intreg setul de date. Acest lucru se datoreaza valorii mari ale Dropout-ului care duc la underfitting, cum se poate observa si in plotul de mai jos (liniile test_loss si train_loss sunt foarte apropiate pe o zona extinsa).

```python
cnn = models.Sequential([
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.BatchNormalization(),
    layers.Dropout(0.25),

    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.35),

    layers.Conv2D(filters=128, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.BatchNormalization(),
    layers.Dropout(0.5),

    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.3),
    layers.Dense(64, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.25),
    layers.Dense(9, activation='softmax')
])
```

```
[0.26229149103164673, 0.9115999937057495]
```

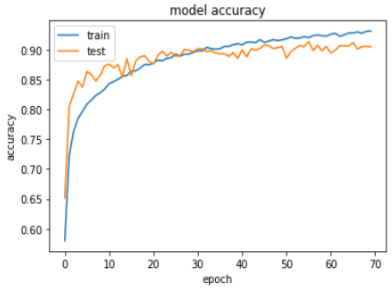


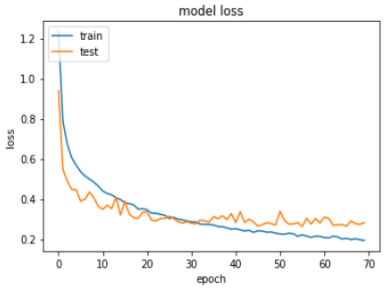


Pentru a stabiliza modelul anterior am modificat valorile straturilor de Dense si de Dropout si de la o acuratete initiala de 0.900 pe 25% din setul de date, aceasta a crescut la 0.909 pe setul complet, pe 70 de epoci, de unde ne putem da seama ca modelul final este semnificativ mai stabil. Dupa cum se observa si in plot, nu mai avem caz de underfitting deoarece liniile test_loss si train_loss se intersecteaza pe o singura portiune, iar apoi pastreaza aceeasi distanta (care nu este foarte mare, caz in care ar fi fost overfitting) pana la finalul executiei.

```python
cnn = models.Sequential([
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.BatchNormalization(),
    layers.Dropout(0.25),

    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.35),

    layers.Conv2D(filters=128, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.BatchNormalization(),
    layers.Dropout(0.5),

    layers.Flatten(),
    layers.Dense(256, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.3),
    layers.Dense(128, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.25),
    layers.Dense(9, activation='softmax')
])
```

```
[0.2826659083366394, 0.9047999978065491]
```





Bibliografie:

1. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
2. https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/
3. https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/
4. https://keras.io/guides/sequential_model/
5. https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/