DEEP HALLUCINATION CLASSIFICATION

Ionescu Alexandru,

grupa 241

Prelucrarea datelor de intrare

Folderul data contine urmatoarele fisiere si foldere:

1. train.txt – date de train, contine id-ul pozelor si label-ul acestora
2. validation.txt – date de validare, contine de asemenea id-ul pozelor si label-ul acestora
3. test.txt – datele de test, contine doar id-ul pozelor
4. sample_submission.txt, contine un exemplu pentru fisierul pe care il vom trimite pe kaggle
5. folderul test, care contine toate pozele pentru test, care au id-urile in fisierul test.txt
6. folderul train+validation, care contine toate pozele atat pentru test, cat si pentru validare, care au id-urile atat in train.txt, validation.txt

Prima incercare de prelucrare a datelor de intrare, gresita:

Pentru a citi label-ul, folosim comanda numpy.loadtxt, delimitam id-ul imaginilor de label si pastram dar label-ul. Procedam la fel pentru a citi label-ul datelor de train si de validare.

Pentru a citi numele imaginilor, folosim glob pentru a citi toate numele fisierelor din folderul train+validation, dar si din folderul test. Citim primele 8000 de filename de imagini si le punem intr-un array pentru train, iar pe restul in cel de validare. La fel procedam si cu datele de train, insa pentru acestea citim tot fisierul test.txt.

Parcurgem pe rand array-urile cu filename-uri, iar pentru fiecare nume de imagine, luam imaginea folosind libraria PIL si dam flatten la valorile pixelilor, pentru a fi pusi intr-un vector unidimensional. Procedam astfel pentru datele de train, validare si test.

Aceasta prelucrarea a datelor de intrare este gresita, intrucat nu corelam numele imaginilor cu label-ul, iar imaginile de train si validare nu sunt invatate cu label-ul corect, astfel ca predictiile vor fi gresite si de asemenea nu vor fi corelate imaginii de test corecte.

A doua incercare de prelucrare a datelor de intrare:

Citim id-urile si label-urile folosind numpy.loadtxt. Pentru a incarca numele fisierelor in array-uri, verificam daca sunt in setul de train sau cel de validare, le incarcam in array-ul care trebuie si apoi incarcam imaginile care au acele nume in array-ul imaginilor de train, validare sau test, folosind cv2. De asemenea, impartim valorile pixelilor la 255, pentru ca acestea sa fie cuprinse astfel intre 0 si 1, lucru care ajuta la precizia unor modele. Pe langa asta, inversam culorile, ele fiind incarcate in formatul BGR initial.

Dupa caz, in functie de model, dam flatten imaginii, trecand-o de pe 3 sau 4 dimensiuni pe una singura.

Apoi, le dam fit pe model si dam predict la datele de validare sau de test.

Preprocesarea imaginilor:

Pentru preprocesare, am incercat mai multe lucruri, precum crop, resize, flip, sharpen si alte modificari la culori, contrast si alte atribute ale imaginii, insa cele folosite cu cele mai bune

rezultate au fost inversarea culorilor, din BGR la citire in RGB, si folosirea imaginilor alb-negru, insa in final am folosit doar inversarea culorilor cu cele mai bune rezultate, cele de mai jos:

Am trimis 5 predictii pentru concurs:

(Acuratetea mentionata mai jos este cea rezultata aplicarii modelului pe datele de validare)

1. Una folosind MultinomialNB: out1;

Atunci cand am aplicat acest model, am folosit prima metoda de prelucrare a datelor de intrare, astfel incat rezultatele pe datele de validare au fost intre 9 si 13% acuratete.

Pe un set de date corect citit, acuratetea a fost intre 36% si 37% in toate testele.

2. O predictie in care am folosit CNN, cu tensorflow.keras.sequential: out4;

Definim modelul sequential cu urmatoarele layer properties:

- Conv2D(40, (2,2), padding = 'same', activation = 'relu', input_shape=(16,16,3))
- Conv2D(80, (2,2), padding = 'same', activation = 'relu')
- MaxPooling2D
- Flatten
- Dense (80, activation = "relu")
- Dense (7, activation = tf.nn.softmax), care specifica modelului in cate clase se impart imaginile

Aceste layer-uri transforma imaginile in straturi de neuroni care apoi sunt pusi intr-o retea neurala. Acest model este compilat cu loss 'sparse_categorical_crossentropy', optimizer 'adam', metrics 'accuracy'.

Atunci cand ii dam fit modelului, specificam si numarul de epoci (50 in cazul acestui test), batch_size, care a fost 22 in acest test.

Pentru aceste valori, insa cu 2 layere Conv2D, acuratetea modelului a variat intre 58% si 61%.

Pentru alte teste, precum:

- Dense(100, activation = "relu"), epoch=100, batch_size=30 – acuratete 56-57%
- Dense(200, activation = "relu"), epoch=100, batch_size=30 – acuratete 58-59%
- Dense(200, activation = "relu"), inca un layer Dense(100, activation = "relu"), epoch=100, batch_size=30 – acuratete 58-60%
- Dense(200, activation = "relu"), inca un layer Dense(100, activation = "relu"), epoch=100, batch_size=60 – acuratete 58-60%
- Dense(250, activation = "relu"), inca un layer Dense(125, activation = "relu"), Conv2D(60, (2,2)…), Conv2D(120, (2,2)…), epoch=100, batch_size=42 – acuratete 59-60%

Pentru modelul de pe kaggle, confusion matrix si classification report-ul sunt urmatoarele:

```
[[148  19   8   4  10  14  13]
 [ 22  89   7  13  10  39  21]
 [  7   4 107   6   8   8   2]
 [  3  11  12  80   9  12  23]
 [  3   3   6   8 112   1  10]
 [  8  34  11  13   8  58  13]
 [ 15  23   2  34   5  16  81]]
```

	precision	recall	f1-score	support
0.0	0.72	0.69	0.70	216
1.0	0.49	0.44	0.46	201
2.0	0.70	0.75	0.73	142
3.0	0.51	0.53	0.52	150
4.0	0.69	0.78	0.73	143
5.0	0.39	0.40	0.40	145
6.0	0.50	0.46	0.48	176
accuracy			0.58	1173
macro avg	0.57	0.58	0.57	1173
weighted avg	0.57	0.58	0.57	1173

3. Doua predictii folosind SVC, care a fost configurat diferit: out2, out3;

Pentru out2, am aplicat SVC, cu mai multe valori pentru gamma, insa majoritatea predictiilor au fost mai putin precise atunci cand am schimbat gamma. Astfel, am incercat mai multe valori pentru C, care reprezinta ponderea unei greseli. Pentru acest test, C a fost mai mic decat 1.

Pentru out3, am aplicat SVC, cu valorile C mai mari decat 1. Am observat ca cele mai bune rezultate sunt cu C intre 3 si 4, iar cele mai bune teste au fost:

C=3.0, 3.95 – acuratete 56.01%

C=3.1, 3.35, 3.4, 3.7, 3.75, 3.8 – acuratete 56.09%

C=3.2, 3.3, 3.6, – acuratete 56.18%

C=3.15, 3.25, 3.45, 3.5, 3.65 – acuratete 56.26%

C=3.55 – acuratete 56.35%

Alte valori intre 3 si 4 au avut acuratete mai mica de 56%, iar C mai mare dar si mai mic decat acest interval aplicat pe model da acuratete mai mica de 56%.

Pentru modelul out3 de pe kaggle, confusion matrix si classification report-ul sunt urmatoarele:

```
[[137  19   9   9  13  19  10]
 [ 19  70  19  23   7  30  33]
 [  3   2 116   6   9   3   3]
 [  3  14   7  94   9   4  19]
 [  3   2   5  12 109   8   4]
 [  6  30  20  14   6  51  18]
 [ 15  20   5  31   5  16  84]]
```

	precision	recall	f1-score	support
0.0	0.74	0.63	0.68	216
1.0	0.45	0.35	0.39	201
2.0	0.64	0.82	0.72	142
3.0	0.50	0.63	0.55	150
4.0	0.69	0.76	0.72	143
5.0	0.39	0.35	0.37	145
6.0	0.49	0.48	0.48	176
accuracy			0.56	1173
macro avg	0.56	0.57	0.56	1173
weighted avg	0.56	0.56	0.56	1173

4. O predictie cu MLPClassifier, out5.

Am configurat modelul pentru urmatorii parametrii: activation='relu', solver='adam', alpha=0.0001, learning_rate='constant', power_t=0.5, shuffle=True, random_state=None, tol=0.0001, momentum=0.9, early_stopping=True, validation_fraction=0.1, n_iter_no_change=10.

De asemenea, am incercat mai multe valori pentru urmatoarii parametrii:

hidden_layer_sizes=(100,), (100,100), (256,), (256,256), (1000,)

batch_size= 16, 32, 64, ‘auto’

learning_rate_init= 0.00001, 0.0001, 0.001, 0.01, 0.1, 1

max_iter= 100, 200, 500, 1000

Cea mai buna acuratete a fost de 55.41%, cu hidden_layer_sizes = (1000,), batch_size = 64, learning_rate_init = 0.0001, max_iter = 500.

Alte modele MLPClassifier au mai avut acuratetea:

54.56%, cu hidden_layer_sizes = (1000,), batch_size = 32, learning_rate_init = 0.0001, max_iter = 200.

54.04%, cu hidden_layer_sizes = (256,256), batch_size = 64, learning_rate_init = 0.0001, max_iter = 200.

54.21%, cu hidden_layer_sizes = (256,256), batch_size = 32, learning_rate_init = 0.0001, max_iter = 200.

54.47%, cu hidden_layer_sizes = (100,100), batch_size = 64, learning_rate_init = 0.001, max_iter = 1000.

54.13%, cu hidden_layer_sizes = (100,100), batch_size = 32, learning_rate_init = 0.0001, max_iter = 200.