Deep Hallucination Classification

Project Documentation

Stan Ana-Maria

Gr. 241

01/04/2022

I. Înțelegerea, citirea si procesarea datelor

Înțelegerea datelor

Folderul train+validation conține 9173 de imagini in format .png, din acestea 8000 aparțin setului Training data iar celelalte 1173 setului Validation data. Astfel, înainte de a încărca un prediction pe Kaggle mai întâi o sa fie antrenat cu Train Data(imaginile care aparțin de train) si comparat cu rezultatele date in validation data(imaginile care aparțin de validation). După ce găsim un procent de acuratețe satisfăcător combinam Train data si Validation data si antrenam modelul pe acestea pentru a avea mai multe date de antrenament ca in final sa dam predict la clasele imaginilor din test.

Folderul test conține 2819 de imagini in format .png si se vor folosi pentru generarea predict-ului care o sa fie încărcat pe Kaggle. Modelul o sa fie antrenat pe setul de date Train si Validation, iar cu ajutorul datelor prelucrate in urma citirii imaginilor din folderul test o sa dam predict la clasele imaginilor din test.

Fișierele train.txt si validation.txt conțin pe prima linie id, labels, pe care o sa le ignoram, iar pe restul de linii se afla numele fișierului si clasa de care aparțin.

Fișierul test.txt conține numele pozelor din folderul test.

Citirea si procesarea datelor

Exemplu de citire al numelor:

```
train_names = np.loadtxt("./data/train.txt",dtype=str,delimiter=',')
```

Ne folosim de librăria numpy(np) si funcția loadtxt pentru a prelua datele din fișierele .txt. Primul parametru reprezintă path-ul către fișier, al doilea modul in care salvam linia(string) si cu ajutorul ultimului separam id-ul si label-ul, care sunt separate de virgula, in doua elemente separate ale aceiași liste. Astfel, vom salva o lista de liste de tipul: [['id''label']['uwOt9wnw5cOryBN.png' '2']['tFoGtQbI1M2cqAQ.png' '1']....].

Facem același lucru si la validation.txt dar la test.txt eliminam ultimul parametru, deoarece aceasta conține doar numele pozelor.

Următorul pas este de a salva corect datele din train_names si validation_names, adică de a nu salva prima linie care conține id si label si de a transforma clasele din stringuri in întregi.

```
v=[]
for i in train_names[1:]:
    j=[]
    j.append(i[0])
    j.append(int(i[1]))
    v.append(j)
train_names=v
```

După citirea corecta a datelor urmează citirea imaginilor. In primul rând, o sa sortam alfabetic train_names si validation_names pentru a avea o ordine bine definita in care le citim si salvam.

```
train_names.sort(key=lambda y: y[0])
validation_names.sort(key=lambda y: y[0]

tl=[]
vl=[]
train_labels=[]
for i in train_names:
    tl.append(i[0])
validation_labels = []
for i in validation_names:
    vl.append(i[0])
for i in glob.glob('./data/train+validation/*'):
    if i[24:] in tl:
        images_train.append(i)
    else:
        images_validation.append(i)
images_train.sort()
images_validation.sort()
```

In tl si vl salvam numele fișierelor corespunzătoare pentru a ne asigura ca citim corect imaginile cu glob.glob. La final le dam din nou sort pentru a le avea ordonate alfabetic. Citim cu glob.glob imaginile din ./data/test/* si concatenam imaginile si data din train cu cele din validation pentru a le folosi împreuna la antrenarea modelului.

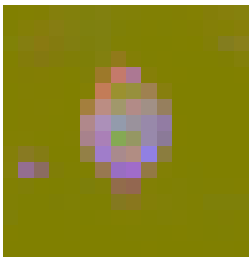
```
for i in glob.glob('./data/test/*'):
    images_test.append(i)

for i in images_validation:
    images_train.append(i)
for i in validation_names:
    train_names.append(i)
```

In final, citim imaginile cu cv2.imread(similar cu ce am făcut la laborator) si încerc sa le prelucrez convertind culoarea im = cv2.cvtColor(im, cv2.COLOR_BGR2YUV).

Mai jos am făcut un test pe un model simplu de tip SVC pentru a observa care conversie a culorii ne oferă cel mai bun rezultat:

cv.COLOR_BGR2YCrCb	0.5498721227621484
cv.COLOR_RGB2YCrCb	0.5498721227621484
cv.COLOR_BGR2HSV	0.5217391304347826
cv.COLOR_BGR2Lab	0.5396419437340153
cv.COLOR_BGR2Luv	0.5362318840579711
cv2.COLOR_BGR2HLS	0.5132139812446718
cv.COLOR_BGR2YUV	0.5541346973572038
COLOR_BGR2GRAY	0.4722932651321398

BGR YUV

Aceasta schimbare de culoare ne oferă cel mai bun rezultat după testele de culoare. Acum urmează sa prelucram datele imaginii pentru a le da fit in model. In acest moment fiecare imagine are o lista in care fiecare pixel are la rândul lui o lista de trei termeni care reprezintă BGR, pentru a trimite datele trebuie sa transformam vectorul de pixeli intr-un singur număr după ce aflam forma vectorului(9173-numarul de imagini,16 lungimea,16 înălțimea si 3 care reprezintă GBR) si reformam vectorul.

II. Aplicarea testelor pe modele

Căutarea celui mai bun model din sklearn

- Naive Bayes

GaussianNB	0.3631713554987212
MultinomialNB	0.31202046035805625
ComplementNB	0.28388746803069054
BernoulliNB	0.19437340153452684

- SVC

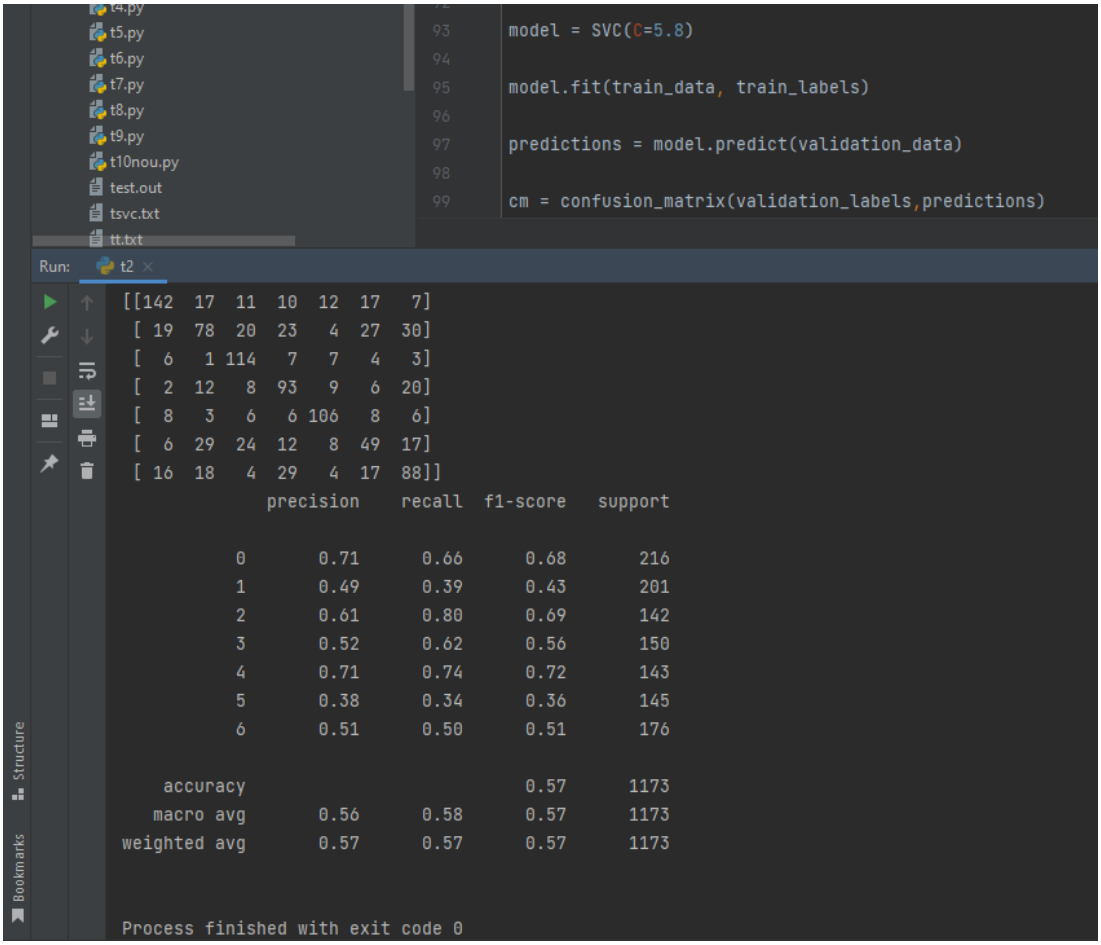
SVC	**0.5541346973572038**

Astfel, rezultatul la svc cu parametrii default are mult mai multa acuratețe decât Naive Bayes si urmează sa încerc mai mulți parametrii random pentru a găsi un rezultat mai bun.

Rezultatul testului, C, Kernel, gamma
0.5541346973572038 1 rbf scale
0.44075021312872975 1 rbf auto
0.5524296675191815 1.2 rbf scale
0.4458653026427962 1.2 rbf auto
0.5524296675191815 1.4 rbf scale
0.4518329070758738 1.4 rbf auto
0.556692242114237 1.6 rbf scale
0.45694799658994034 1.6 rbf auto
0.5549872122762148 1.8 rbf scale
0.45865302642796246 1.8 rbf auto
0.5626598465473146 2 rbf scale
0.4620630861040068 2 rbf auto
0.5635123614663257 2.2 rbf scale
0.4629156010230179 2.2 rbf auto
0.5635123614663257 2.4 rbf scale
0.4680306905370844 2.4 rbf auto
0.5626598465473146 2.6 rbf scale
0.47144075021312876 2.6 rbf auto
0.5558397271952259 2.8 rbf scale
0.47655583972719523 2.8 rbf auto
0.5609548167092924 3 rbf scale
0.47911338448422847 3 rbf auto
0.5652173913043478 3.2 rbf scale
0.4774083546462063 3.2 rbf auto
0.566069906223359 3.4 rbf scale
0.48081841432225064 3.4 rbf auto
0.5643648763853367 3.6 rbf scale
0.4816709292412617 3.6 rbf auto
0.56692242114237 3.8 rbf scale
0.4833759590792839 3.8 rbf auto
0.566069906223359 4 rbf scale
0.4833759590792839 4 rbf auto
0.5635123614663257 4.2 rbf scale
0.484228473998295 4.2 rbf auto
0.5635123614663257 4.4 rbf scale
0.484228473998295 4.4 rbf auto
0.566069906223359 4.6 rbf scale

0.4816709292412617 4.6 rbf auto
0.5677749360613811 4.8 rbf scale
0.48678601875532823 4.8 rbf auto
0.5694799658994032 5 rbf scale
0.4884910485933504 5 rbf auto
0.5694799658994032 5.2 rbf scale
0.4884910485933504 5.2 rbf auto
0.5694799658994032 5.4 rbf scale
0.48934356351236147 5.4 rbf auto
0.5686274509803921 5.6 rbf scale
0.48934356351236147 5.6 rbf auto
0.5711849957374254 5.8 rbf scale
0.49019607843137253 5.8 rbf auto
0.5694799658994032 6 rbf scale
0.49104859335038364 6 rbf auto
0.5217391304347826 1 poly scale
0.37169650468883203 1 poly auto
0.5174765558397272 1.2 poly scale
0.3742540494458653 1.2 poly auto
0.5166240409207161 1.4 poly scale
0.38022165387894286 1.4 poly auto
0.5157715260017051 1.6 poly scale
0.38107416879795397 1.6 poly auto
0.5183290707587382 1.8 poly scale
0.3853367433930094 1.8 poly auto
0.5200341005967605 2 poly scale
0.3887468030690537 2 poly auto
0.5157715260017051 2.2 poly scale
0.39215686274509803 2.2 poly auto
0.5123614663256607 2.4 poly scale
0.40068201193520886 2.4 poly auto
0.5098039215686274 2.6 poly scale
0.4092071611253197 2.6 poly auto
0.5080988917306053 2.8 poly scale
0.41091219096334186 2.8 poly auto
0.5089514066496164 3 poly scale

In urma rezultatelor date am încercat valori de la 3 pana la 6 pentru a afla cele mai bune variante, iar in final cele mai apropiate au fost : 3.5, 3.55, 3.9, 5.8.



- Rețele cu MLP

```
for h in [100,255,256,500,1000]:
    for b in [100,64,80,30,1]:
        for lr in ['constant', 'invscaling', 'adaptive']:
            for lri in [0.01,0.001,0.0001,0.00001]:
                for mi in (200,400,700,1000,1500,2000,2500,3000,3250,3500,3800):
                    for i in [True,False]:
                        model = MLPClassifier(hidden_layer_sizes=(h,), activation='relu', solver='adam',
alpha=0.0001,
                                              batch_size=b, learning_rate=lr, learning_rate_init=lri,
                                              power_t=0.5, max_iter=mi, shuffle=True, random_state=None,
tol=0.0001,
                                              momentum=0.9, early_stopping=i, validation_fraction=0.1,
                                              n_iter_no_change=10)
                        model.fit(train_data, train_labels)
```

```
                test_predictions = model.predict(validation_data)

                a=np.mean(test_predictions == validation_labels)
                print(a,h,b,lr,lri,mi,i)
                if a > maxi:
                    maxi = a
            v=[h,b,lr,lri,mi,i]
```

După mai multe teste pe un MLPClassifier am ajuns la concluzia ca cel mai bun rezultat rămâne încă cel cu SVC. După ce am rulat programul de mai sus am găsit câteva rezultate mai bune dar niciunul nu a fost destul de bun. Exemplu de rezultate:

['0.5618073316283035', '100', '100', 'invscaling', '1e-05', '3000', 'False']
['0.5558397271952259', '100', '100', 'constant', '1e-05', '3250', 'False']
['0.5618073316283035', '100', '100', 'constant', '1e-05', '3500', 'False']
['0.5618073316283035', '100', '64', 'constant', '1e-05', '3000', 'False']

Rețele cu tensorflow si keras

După zile de rulat pe MLPClassifier nu am reușit sa găsesc un rezultat satisfăcător așa ca am început sa studiez despre niște rețele mai avansate, acolo am aflat despre tensorflow si ce poate acesta sa facă acesta(deep learning).

Astfel, pentru a folosi un model de tip Sequential a trebuit mai întâi sa schimbam modul in care trimitem datele, nu o sa le mai dam reshape si doar o sa le facem de tip float si o sa împărțim la 255, deoarece modelul funcționează mai bine când numerele sunt in intervalul [0,1].

Definirea modelului:

```
            model = tf.keras.Sequential([])
```

In parantezele pătrate urmează sa adăugam layers de mai multe tipuri si sa le combinam pana găsim o combinație potrivita.

Tipurile de layers pe care o sa le aplic o sa fie o combinație de următoarele:
tf.keras.layers.Conv2D, tf.keras.layers.MaxPooling2D, tf.keras.layers.MaxPool2D, tf.keras.layers.Dropout, tf.keras.layers.Flatten, tf.keras.layers.Dense.

După primul test in care am pus conv2D(50,(2,2),'relu'), MaxPooling2D,Flatten() si Dense(7,'softmax') rezultatul a fost deja mai bun de cel oferit de SVC, 0.57. Așa ca următoarele teste au fost sa adaug, sa combin si sa schimb parametrii din aceste funcții. Adăugând încă un layer de Conv2D(100) acuratețea a crescut la 0.59, iar după adăugarea unui Dense(100) si un MaxPooling2D am reușit sa ajung pana la 0.61.

Următorul loc care poate sa influențeze acuratețea este când compilam modelul.

model.compile(loss=, optimizer=)

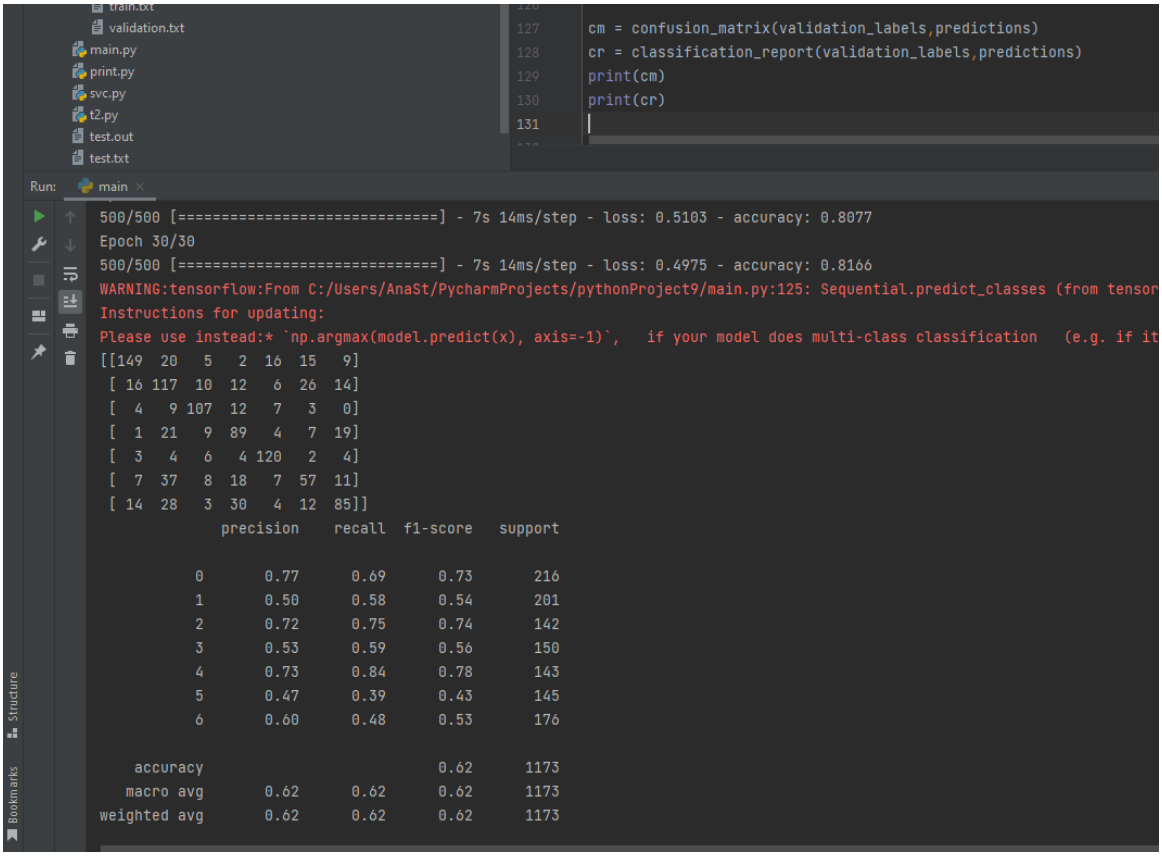
Loss	Acuratețea
mean_squared_error	0.12
mean_absolute_error	0.13
sparse_categorical_crossentropy	0.61
hinge	0.21
poisson	0.17

La optimizer cel care a dat rezultatele cele mai bune a fost Adam, căruia putem sa ii schimbam learning rate-ul.

Learning rate	Acuratețea
0.1	0.13
0.01	0.53
0.001	0.61
0.0001	0.55

In final, importante pentru acurateț̦e sunt numărul de epoci si batch size din funcția fit:

[[0.556692242114237, 10, 15], [0.5788576300085252, 10, 30], [0.5865302642796248, 10, 50], [0.5856777493606138, 10, 60], [0.6027280477408354, 10, 80], [0.6035805626598465, 20, 15], [0.6112531969309463, 20, 30], [0.6163682864450127, 20, 50], [0.6214833759590793, 20, 60], [0.6138107416879796, 20, 80], [0.6142028985507246, 30, 15], [0.6001705029838023, 30, 30], [0.5848252344416027, 30, 50], [0.6001705029838023, 30, 60], [0.5950554134697357, 30, 80], [0.5797101449275363, 40, 15], [0.5814151747655584, 40, 30], [0.5890878090366581, 40, 50], [0.5805626598465473, 40, 60], [0.5822676896845694, 40, 80], [0.5822676896845694, 50, 15], [0.5805626598465473, 50, 30], [0.5814151747655584, 50, 50], [0.5831202046035806, 50, 60], [0.5831202046035806, 50, 80], [0.5805626598465473, 60, 15], [0.5822676896845694, 60, 30], [0.5763000852514919, 60, 50], [0.5788576300085252, 60, 60], [0.5797101449275363, 60, 80]]



III. Concluzie

După toate testele făcute am ajuns sa găsesc un model făcut cu tenserflow, care sa îmi returneze o acuratețe de 0.63, iar pe site-ul competiției chiar de 0.64. Acesta se afla in CNN_principal.py, iar cel cu rezultat cel mai bun de tipul SVC se afla in SVC_principal.py.