Baias Bogdan-Andrei
Grupa 244

Documentatie Image Classifier

1. Input

   Pentru crearea modelului am folosit un CNN. Pentru preluarea datelor am citit din fisierele train.txt si validation.txt linie cu linie, am preluat la fiecare linie numele fotografiilor (primele 10 caractere) si etichetele fiecarei fotografii, memorand aceste date in vectori corespunzatori (train_images, train_labels, validation_images, validation_labels). Fiecare imagine preluata a fost convertita intr-un vector numpy ale carui valori au fost impartite la 255. Astfel, vectorii rezultati memoreaza intensitatea unui pixel cu valori cuprinse intre 0 si 1 (0 pentru negru, 1 pentru alb pur).

2. Data augmentation

   Dupa citirea si memorarea imaginilor am aplicat un reshape pe vectorii in care erau memorate (`train_images = train_images.reshape(-1 , 32, 32, 1)`) pentru a seta tipul imaginii din vector ca fiind alb-negru si pentru a-si pastra forma. Dorind sa obtin o acuratete mai mare si sa remediez problema overfitting-ului am incercat sa augumentez setul de date de antrenament mai intai transformand cele 30001 de imagini de antrenament prin doua rotiri succesive la 90 de grade, obtinand astfel un set de date de antrenament format din de trei ori mai multe imagini, apoi am incercat o rotire de 90 de grade doar o singura data, rezultand un numar de 60002 imagini de antrenament, iar finalmente am incercat sa folosesc un pseudo-random cu care sa rotesc la intamplare imaginile inainte de a le salva in vectorul train_images (`if((17 * i + 5) %23 == 13):` `img = img.rotate(90)`). Am abandonat aceasta incercare de a augumenta setul de date de intrare intrucat aceste modificari au dus fie la underfitting, fie la o acuratete cu cateva procente mai mica si un impact mult prea mic asupra overfitting-ului.

3. Model generation

   Am folosit un model de tip sequential si, pentru fiecare model, un numar diferit de blocuri tip 2D convolutional layers + max pooling 2D. Pentru fiecare layer convolutional am folosit activation RELU (Rectified Linear Unit), care transforma toate valorile negative in 0, astfel neactivand neuronul asociat. Pentru max pooling 2D am ales un pooling de (2,2) pentru a discretiza inputul si pentru a face modelul sa se focuseze pe caracteristicile principale ale obiectului din poza (down-sampling). In acelasi timp, am pastrat acelasi padding pentru a nu “pierde” informatia de la marginea imaginii. Am constatat in timpul testarii ca lipsa pastrarii padding-ului ar fi dus si la eroare de rulare, insa relatia de cauzalitate nu e una clara intrucat pe acele teste incercam sa folosesc foarte multe layere convolutionale si eroarea nu aparea neaparat la prima adaugare de layere convolutionale, astfel ca restrangerea utilizata in privinta padding-ului a fost mai mult decat orice altceva o masura de safe practice. Numarul de blocuri, de layere si existenta unui pooling au diferit de la model la model si voi detalia ulterior pentru fiecare

model intrucat in perioada incipienta a dezvoltarii modelului final am testat folosind valori inutil de mari (e.g. 4096 de layere convolutionale adaugate cu o singura instructiune sau 16 adaugari a cate 64 de layere convolutionale) care au condus doar la modele extrem de overfitted (e.g. aproximativ 98% acuratete pe setul de date de antrenament si aproximativ 80% acuratete pe setul de date de validare), care se antrenau intr-un timp extrem de lung (70 de minute pe epoca in unele cazuri). Testarea cu aceste valori a avut ca scop doar sa imi aduca o mai buna intelegere a procesului de antrenare a modelului pentru a putea sa ajustez eficient pe viitor hiperparametrii, fiind foarte clar ca o testare de 70 de minute / epoca nu ar fi fost dezirabila, cu atat mai putin avand in vedere randamentul redus pe care il are asociat.

Dupa adaugarea acestor layere am adaugat un layer flattened cu care am “centralizat” neuronii. Mai apoi am folosit ca regularizare un dropout variabil de la model la model pentru a evita overfittingul (care de altfel a fost cea mai mare problema pe care am intampinat-o in cadrul acestui proiect), dropout care a avut valori incadrate intre 0.2 si 0.9 in faza in care am testat influenta acestei valori asupra eficientei alogritmului, teste in urma carora am constatat ca un dropout mai mic decat 0.5 nu are un impact notabil asupra overfitting-ului, pe cand un dropout mai mare decat 0.75-0.8 are un impact negativ significant asupra eficientei de recunoastere a imaginilor atat din setul de date de test, cat si din setul de date de validare.

Am compilat modelul folosind optimizerul adam, acesta fiind cel mai bun pentru sparse data (multe valori de 0 care ar fi putut rezulta din utilizarea optimizatorului adam) si am reprezentat diferenta dintre labelurile reale si cele prezise folosind `loss='sparse_categorical_crossentropy'`. Numarul de epoci pe care s-a antrenat modelul a variat intre 10 si 30, acest numar crescand odata cu o mai buna gestionare a fenomenului de overfitting. Modelele 1 si 3 au rulat cu un numar de 10 epoci (am constatat totusi ca overfitting-ul intervine incepand cu generatia a 6-a), modelul 2 a rulat cu un numar de 30 de epoci(am observat ca overfitting-ul apare odata cu generatia 24), iar modelele 4 si 5 au fost compilate cu un numar de 25 de epoci(nu am observat un punct clar de incepere a overfitting-ului, totusi incepand cu intervalul de epoci 10-15 am putut observa o oarecare stabilizare a acuratetilor, acuratetea pe setul de date de test devenind apropiata de acuratetea pe setul de date de validare, cele doua crescand in mare parte concomitent).

4. Modele finale salvate

Am salvat 5 modele, cele care au dat in testele efectuate de mine cele mai bune acurateti pentru fiecare zi in care am realizat submission-uri pe kaggle. De asemenea, in cod am salvat sub forma de comentarii si valori intermediare care au dus la gasirea configuratiilor salvate in modele, insa modelele compilate din acele valori nu au fost salvate sau au fost suprascrise cu modelele salvate la final in acea zi. Toate aceste modele au condus, printre altele, si la concluzia ca daca adaug de mai mult de 5 ori layere convolutionale voi obtine atat overfitting accentuat cat si acuratete pe setul de date de validare scazuta comparativ cu 5 sau mai putine adaugari. Acest numar de 5 adaugari de layere convolutionale a fost testetat initial pornind de la ideea retelei neuronale convolutionale AlexNet, motiv pentru care am si experimentat cu anumite

layere convolutionale care nu sunt urmate de max pooling si cu adaugarea mai multor dense layers, teste care s-au dovedit a nu aduce un beneficiu notabil modelelor.

Pentru citirea imaginilor de test am folosit aceeasi tehnica folosita si la memorarea imaginilor de antrenare si validare.

Pentru scrierea in fisier csv am folosit biblioteca csv si writer-ul cu care aceasta vine, scriind pe fiecare rand numele imaginii analizate si apoi predictia pentru ea.

Matricea de confuzie am generat-o cu ajutorul sklearn.metrics.

De mentionat este ca toate testele efectuate de mine au fost influentate si de explicatiile oferite pe canalul oficial de YouTube al Tensorflow in videoclipurile in care explica retelele neuronale convolutionale si implementarea lor.

5. Implementare modele, acuratete, matrici de confuzie

Model 1
Acuratete pe setul de validare: 87.30%

Cod

```
model = models.Sequential()
model.add(layers.Conv2D(32, (5,5),  activation='relu',data_format='channels_last'))
model.add(layers.MaxPooling2D((2,2), data_format='channels_last', padding='same'))
model.add(layers.Conv2D(64 , (5,5) , activation='relu', data_format='channels_last'))
model.add(layers.MaxPooling2D((2,2), data_format='channels_last', padding='same'))
model.add(layers.Conv2D(64 , (5,5), activation='relu', data_format='channels_last'))
model.add(layers.MaxPooling2D((2,2), data_format='channels_last', padding='same'))
model.add(layers.Conv2D(32 , (5,5) , activation='relu', data_format='channels_last',
padding='same'))
model.add(layers.MaxPooling2D((2,2), data_format='channels_last', padding='same'))
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(32))
model.add(layers.Dense(9, activation='softmax'))
```

Matrice de confuzie

```
[[193 109  26  14  94  14  12  45  63]
 [  1 483  13   1  10  10   0   8   1]
 [  1  28 444   3  18  28   2   6   3]
 [  1  38  60 295  70  46   8  27  33]
 [  7  63  77  16 323  34   7  20   7]
 [  0   8  68   6   9 444   8   4  14]
 [  4  95  32  10  12  14 352  15  46]
 [  2  51  25   9  20  76   3 328   6]
 [ 17  45   8  12  14  20  27  13 421]]
```

Model 2
Acuratete pe setul de validare: 87.46%

Cod:

```
model = models.Sequential()
model.add(layers.Conv2D(32, (5,5),  activation='relu',data_format='channels_last'))
model.add(layers.MaxPooling2D((2,2), data_format='channels_last', padding='same'))
model.add(layers.Conv2D(64 , (5,5) , activation='relu', data_format='channels_last'))
model.add(layers.MaxPooling2D((2,2), data_format='channels_last', padding='same'))
model.add(layers.Conv2D(64 , (5,5), activation='relu', data_format='channels_last'))
model.add(layers.MaxPooling2D((2,2), data_format='channels_last', padding='same'))
model.add(layers.Conv2D(32 , (5,5) , activation='relu', data_format='channels_last',
padding='same'))
model.add(layers.MaxPooling2D((2,2), data_format='channels_last', padding='same'))
model.add(layers.Conv2D(32 , (5,5) , activation='relu', data_format='channels_last',
padding='same'))
model.add(layers.MaxPooling2D((2,2), data_format='channels_last', padding='same'))
model.add(layers.Flatten())
model.add(layers.Dropout(0.7))
model.add(layers.Dense(32))
model.add(layers.Dense(9, activation='softmax'))
```

Matrice de confuzie:

```
[[262 154  22  10  34  19  24  18  27]
 [  0 503   9   2   2   9   0   1   1]
 [  1  87 396  11   7  26   4   1   0]
 [  5  87  15 314  22  98  16   9  12]
 [ 18 133  45  28 254  54   6  11   5]
 [  2  36  12   8  14 474   8   3   4]
 [  3  78  15   5   4   7 454   5   9]
 [  5  95  17   9  10  67   7 309   1]
 [ 39  58   3  21  15  21  55   7 358]]
```

Model 3

Acuratete pe setul de validare: 87.42%

Cod:

```
model = models.Sequential()
model.add(layers.Conv2D(32, (5,5),  activation='relu',data_format='channels_last'))
model.add(layers.MaxPooling2D((2,2), data_format='channels_last', padding='same'))
model.add(layers.Conv2D(64 , (5,5) , activation='relu', data_format='channels_last'))
model.add(layers.MaxPooling2D((2,2), data_format='channels_last', padding='same'))
model.add(layers.Conv2D(64 , (5,5), activation='relu', data_format='channels_last'))
model.add(layers.MaxPooling2D((2,2), data_format='channels_last', padding='same'))
model.add(layers.Conv2D(128 , (5,5) , activation='relu', data_format='channels_last',
padding='same'))
model.add(layers.MaxPooling2D((2,2), data_format='channels_last', padding='same'))
model.add(layers.Conv2D(128 , (5,5) , activation='relu', data_format='channels_last',
padding='same'))
model.add(layers.Flatten())
model.add(layers.Dense(128))
model.add(layers.Dense(9, activation='softmax'))
```

Matrice de confuzie:

[[165 166 34   8 72 35 15 36 39]
 [0 488 13   3   4 13   1   4   1]
 [0 47 413 14 13 37   6   1   2]
 [5 37 39 322 31 87 18 24 15]
 [4 108 36 27 314 35   2 20   8]
 [0 14 33 11 16 461   7 12   7]
 [2 99 25 19 13 32 361 11 18]
 [5 83 23 14   9 85   0 298   3]
 [20 40   6 27 14 26 55 11 378]]

Model 4
Acuratete pe setul de validare: 88.06% - 90.16% public kaggle

Cod:

```
model = models.Sequential()
model.add(layers.Conv2D(32, (5,5),  activation='relu',data_format='channels_last'))
model.add(layers.MaxPooling2D((2,2), data_format='channels_last', padding='same'))
model.add(layers.Conv2D(64 , (5,5) , activation='relu', data_format='channels_last'))
model.add(layers.MaxPooling2D((2,2), data_format='channels_last', padding='same'))
model.add(layers.Conv2D(64 , (5,5), activation='relu', data_format='channels_last'))
model.add(layers.Conv2D(32 , (5,5) , activation='relu', data_format='channels_last',
padding='same'))
model.add(layers.MaxPooling2D((2,2), data_format='channels_last', padding='same'))
model.add(layers.Conv2D(128 , (5,5) , activation='relu', data_format='channels_last',
padding='same'))
model.add(layers.Flatten())
model.add(layers.Dropout(0.75))
model.add(layers.Dense(128))
model.add(layers.Dense(64))
model.add(layers.Dense(32))
model.add(layers.Dense(9, activation='softmax'))
```

Matrice de confuzie:

[[220 155 20 18 56 13 12 35 41]
 [0 506   6   1   0   9   0   5   0]
 [2 88 372 20   8 31   2   9   1]
 [2 81 32 318 44 53 18 23   7]
 [8 150 33 22 294 18   2 21   6]
 [5 20 33 14 14 454   9 10   2]
 [18 117 24 21   4 18 335 28 15]
 [3 103   8 19   9 53   0 323   2]
 [29 68   6 37   9 18 35 16 359]]

Model final
Acuratete pe setul de validare: 89.94% -90.32% public kaggle

Cod:

```
model = models.Sequential()
model.add(layers.Conv2D(32, (5,5),  activation='relu',data_format='channels_last'))
model.add(layers.MaxPooling2D((2,2), data_format='channels_last', padding='same'))
model.add(layers.Conv2D(64 , (5,5), activation='relu', data_format='channels_last'))
model.add(layers.MaxPooling2D((2,2), data_format='channels_last', padding='same'))
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(64))
model.add(layers.Dense(32))
model.add(layers.Dense(9, activation='softmax'))
```

Matrice de confuzie:

```
[[212 134  12  30  40  13  25  50  54]
 [  4 447  11  13  12  15   4  14   7]
 [  6  54 302  44  36  59   7  21   4]
 [  4  32  12 396  19  46  20  26  23]
 [ 25 111  20  82 218  35   8  44  11]
 [  2   8  12  23  22 454  11  22   7]
 [  2  74  12  26   0  13 404  20  29]
 [  1  56  12  33  17  47   5 345   4]
 [ 17  40   0  29   4  12  51  20 404]]
```