Documentație proiect Inteligență artificială

Enunțul cerinței se găsește la adresa: https://www.kaggle.com/c/ai-unibuc-23-31-2021/overview.

În rezolvarea task-ului, am folosit mai multe modele specifice subiectului, de tip CNN (convolutional neural network).

O rețea neuronală convoluțională (CNN) reprezintă o clasă de rețele neuronale, adesea folosită în analiza imaginilor. Aceasta folosește o tehnică specială, numită convoluție. În matematică, această operație este aplicată asupra a 2 funcții, producând ca rezultat o a treia, ce evidențiază cum forma uneia este influențată/modificată de cealaltă.

Biblioteci folosite: TensorFlow, Keras, numpy, pandas, sklearn

Pașii algoritmului:

I. Etapa de preprocesare
II. Definirea modelului
III. Compilarea și antrenarea modelului
IV. Predicțiile
V. Construirea și afișarea matricei de confuzie pentru setul de validare
VI. Scrierea predicțiilor obținute în fișierul CSV

În **etapa de preprocesare**, am citit datele de antrenare, validare, respectiv datele de testare din fișierele CSV aferente, folosind biblioteca pandas, mai precis metoda read_csv.

Imaginile au fost stocate în structuri de tip numpy array, după ce au fost normalizate. Fenomenul de normalizare este necesar întrucât într-o rețea neuronală, prelucrarea valorilor numerice mari poate deveni complexă. Astfel, tranformăm fiecare pixel (care poate avea valori cuprinse între 0 și 255, fiecare reprezentând câte o culoare) în intervalul [0,1]. În acest fel, computația devinde mai rapidă.

Etichetele imaginilor au fost stocate sub forma unor matrici de tip număr_etichete X număr_clase, folosind metoda to_categorical (keras.utils.np_utils).

Etapa a doua a algoritmului constă în **construirea modelului,** căruia îi vom adăuga diferite straturi (layere): Convolution layers, Pooling layers, funcții de activare, Flatten, Dense layers etc.

Structura modelului:

Model = Sequential()

Inițializarea modelului se face prin **Sequential()**, ce pune baza „unei stive liniare de straturi” . În continuare, vom adăuga straturile:

- **CONV2D** (output_shape, convolution_window_size, input_shape) reprezintă primul layer al rețelei. Este folosit pentru extragerea caracteristicilor unei imagini oferite ca input. Convoluția păstrează legăturile dintre pixelii imaginii, pe care o împarte în mici ferestre pentru a „învăța”: *model.add(Conv2D(16,(3, 3), input_shape=(50,50,1))*
- Layer-ul de convoluție este **activat** prin funcția „ReLU” (Rectified Linear Unit). Aceasta preia valorile ce reprezintă imaginea, într-o formă liniară (listă de numere), datorită stratului de convoluție anterior și returnează max(0,x) - deviația standard ReLU. Avantajul acestei funcții de activare, față de altele, este că nu activează toți neuronii în același timp: *model.add(Activation('relu'))*
- **MaxPooling2D**(pool_size): acest layer preia informația („activată”) ce reprezintă imaginea (feature map), o „micșorează” (pentru a preveni overfitting-ul), apoi abstractizează părțile pe care le consideră irelevante în procesul de învățare. MaxPooling obține valorile maxime ale pixelilor, cu scopul de a depista posibile distorsionări ale imaginilor: *model.add(MaxPooling2D(pool_size=(2, 2)))*
- Următoarele **3 straturi** ale rețelei sunt aceleași ca mai sus, unul de **convoluție**, în care modificăm output shape (32 – putere a lui 2), o **funcție de activare de tip ReLU** și un layer de Pooling – **MaxPooling2D** cu aceiași parametri folosiți anterior (de obicei, în practică, pool_size este folosit cu valorile (2,2)):

  model.add(Conv2D(32, (3,3)))

  model.add(Activation('relu'))

  model.add(MaxPooling2D(pool_size=(2, 2)))

- Mai adăugăm un strat de convoluție și o funcție de activare de tip „ReLU”

  model.add(Conv2D(32, (3,3)))

  model.add(Activation('relu'))

- Folosim cel de-al 6-lea strat, **Flatten()**, pentru a „compresa” informația obținută într-un vector, înainte de a o transmite layerelor finale (densely connected layers):

  model.add(Flatten())

- Următorul strat, **Dense(num_neurons)**: acest layer este conectat „în adâncime”, ceea ce înseamnă că fiecare neuron al acestuia primește informație de la toți neuronii layer-ului anterior.
- **Funcție de activare „ReLU”**, cu aceleași specificații ca mai sus

  model.add(Dense(128)) #param = nr de neuroni

  model.add(Activation('relu'))

- **Dropout(0.5)** este folosit pentru a evita overfitting-ul, renunță la 0.5 unități din tensor

  model.add(Dropout(0.5))

- **Dense(3)** este folosit ca și ultim layer, de data asta având ca parametru numărul de clase.
- **Funcția de activare** acum este de tip **„softmax”**. Adesea, ea este folosită ca și layer final în modelele de tip „multi-classification logistic regression”, alături de un loss de tip **categorical_crossentropy**.

Compilăm modelul astfel:

model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])

Parametri utilizați:

- Performance function/optimizer **ADAM (**Adaptive Moment Estimation)

Potrivit Kingma et al., 2014, metoda este „computațional eficientă, nu necesită multă memorie [...] ”. În cele mai multe cazuri, se dovedește a fi printre cele mai eficiente tehnici. Poate fi privit ca o combinație între optimizatorul RMSprop și descendența gradientă de tip stochastic. Adam este o metodă cu rata de învățare adaptivă - calculează rate de învățare individuale în funcție de parametri diferiți. Are ca parametru default pentru learning_rate valoarea 0.001.

- Loss function: **categorical_crossentropy**

Este folosită adesea pentru clasificarea cu mai mult de 2 clase (multi-class classification). În acest tip de clasificare , o imagine poate aparține doar uneia din multiple categorii. Funcția estimează diferența dintre 2 distribuții de probabilitate.

Antrenarea modelului:

model.fit(train_data,train_labels,epochs=50,validation_data=(validation_data, validation_labels))

Pentru predicții folosim: *test_labels = model.predict_classes(test_data)*

Metoda predict_classes returneaza clasa specifica fiecarei imagini.

Afișarea matricei de confuzie pe setul de validare:

validation_predictions = model.predict_classes(validation_data)

rounded_labels=np.argmax(validation_labels, axis=1)

cm = confusion_matrix(rounded_labels,validation_predictions)

print(cm)

Facem predicțiile pentru datele de validare, apoi le „rotunjim”. Este folosită funcția confusion_matrix a bibilotecii sklearn.metrics.

Output-ul:

[[1299 179   22]

[287 790 423]

[135 283 1082]]

Scrierea în fișierul CSV

Copiez conținutul fișierului submission.txt, adaug predicțiile coloanei cu numele ”label”, apoi convertesc rezultatul la tipul .csv.

```
outputFile = read_csv('sample_submission.txt')
 outputFile['label'] = test_labels
outputFile.to_csv('submission4.csv', header=True, index=False)
```

MODELUL 2

```
model = Sequential()
model.add(Conv2D(16,(3, 3), input_shape=(50,50,1)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, (3,3)))
model.add(Activation('relu'))
model.add(Dropout(0.3)) #to avoid overfitting, just in case
model.add(Flatten())
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(3))
 model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',optimizer='rmsprop',metrics=['accuracy'])
```

Funcția de activare este de tip **sigmoid.** Ea returnează pentru x>5, valori apropiate de 1, iar pentru x<-5, valori apropiate de 0. Intervalul valorilor returnate este [0,1].

Matricea de confuzie pentru datele de validare:

[[1172 243   85]

[264 929 307]

[158 361 981]]

Model 3

```
model = Sequential()
model.add(Conv2D(16,(3, 3), input_shape=(50,50,1)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, (3,3)))
model.add(Activation('relu'))
model.add(Flatten())
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(3))
 model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adadelta', metrics=['accuracy'])
```

Model 4

```
model = Sequential()
model.add(Conv2D(16,(3, 3), input_shape=(50,50,1)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3,3)))
model.add(Activation('relu'))
```

```
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, (3,3)))
model.add(Activation('relu'))
model.add(Dropout(0.3)) #to avoid overfitting, just in case
model.add(Flatten())
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(3))
 model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',optimizer='SGD',metrics=['accuracy'])
```