Splątane sieci neuronowe CNN - architektura ResNet50

mgr inż. Grzegorz Kossakowski

6.10.2024

1. Opis architektury

ResNet50 [1] została opracowana w 2015 roku przez firmę Microsoft Research. Pełna nazwa to "Residual Network" a 50 w nazwie odnosi się do ilości warstw w sieci, która ma 50 warstw głębokich.

Jest bardzo potężną architekturą, którą można trenować na bardzo dużej ilości danych.

2. Pobranie potrzebnych bibliotek

Kolejnym krokiem jest wczytanie wszystkich potrzebnych bibliotek, dzięki którym będzie możliwe wykorzystanie ich w procesie klasyfikacji.

[2]:
```
TF_ENABLE_ONEDNN_OPTS=0
from astropy.io import fits
# from sklearn.model_selection import train_test_split
from keras.callbacks import ReduceLROnPlateau
from keras.optimizers import Adam
from keras import Sequential
from tests.test_layers import Dense, Flatten
from keras.applications import ResNet50
import pandas as pd
```

3. Pobranie danych z pliku fits

Dlatego że wcześniej podzieliliśmy dane na odpowiednie części, teraz pobieramy dwa zbiory. Pierwszy z nich to zbiór, na którym będziemy uczyć nasz model. Drugi to zbiór walidacyjny.

[3]:
```
hdu_train = fits.open('Data/train.fits')
hdu_valid = fits.open('Data/valid.fits')
x_train = hdu_train[0].data
y_train = hdu_train[1].data
x_valid = hdu_valid[0].data
y_valid = hdu_valid[1].data
```

[4]:
```
x_train.shape, x_valid.shape, type(x_train)
```

[4]:
```
((11350, 256, 256, 3), (2838, 256, 256, 3), numpy.ndarray)
```

4. Pobranie danych

W tym kroku pobieramy dane, a następnie przygotowujemy je do klasyfikacji. Modele głębokiej sieci neuronowej [4] wymaga danych z zakresu 0..1, dlatego wszystkie wartości w danych są dzielone przez 255. Powodem takiego zachowania jest fakt, że dane obrazów są przechowywane w zakresie liczb 0..255. Dzielenie przez 255 powoduje, że dane zostaną zapisane w zakresie od 0..1, zgodnie z wymaganiami modelu.

[5]:
```
reduceLR = ReduceLROnPlateau(monitor='accuracy', factor=.001, patience=1,␣
↪min_delta=0.01, mode="auto")
x_train = x_train / 255.0
x_valid = x_valid / 255.0
```

5. Ustawienie sposobu nauki

Modele, które używany są już wstępnie wyuczone, dlatego chciałem sprawdzić, jak dany model będzie się zachowywał w dwóch przypadkach. Pierwszy przypadek gdy wartość fullStudy zostanie ustawiona na false wtedy model będzie wykorzystywał wcześniej nauczony model i na ostatnich warstwach będzie douczał tylko danymi astronomicznymi. Gdy ustawimy wartość na true, model od początku będzie, wykonał naukę architektury. Wcześniejsza nauka nie będzie brana pod uwagę. Pozwoli to ocenić, który sposób jest bardziej efektywny.

[6]:
```
fullStudy = False
```

6. Budowa modelu.

Model w tym przypadku to ResNet50. Jest to model składający się z 50 warstw. Kolejną warstwą po warstwach splątanych jest warstwa flatten. Zadaniem tej warstwy jest spłaszczenie obrazu z wymiarów otrzymanych z warstw splątanych na pojedynczy ciąg. Ostatnią warstwą jest gęsto połączona warstwa wyjściowa. W naszym modelu klasyfikacja odbywa się dla 10 kategorii, dlatego ta warstwa zawiera 10 neuronów.

[7]:
```
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(256,␣
↪256,3))
base_model.trainable = fullStudy
model_optimizer = Adam(learning_rate=0.001)

model = Sequential()
model.add(base_model)
model.add(Flatten())
model.add(Dense(10, activation="softmax"))

model.compile(optimizer=model_optimizer, loss='sparse_categorical_crossentropy',␣
↪metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
```

```
=================================================================
 resnet50 (Functional)       (None, 8, 8, 2048)        23587712

 flatten (Flatten)           (None, 131072)            0

 dense (Dense)               (None, 10)                1310730

=================================================================
Total params: 24898442 (94.98 MB)
Trainable params: 1310730 (5.00 MB)
Non-trainable params: 23587712 (89.98 MB)
_________________________________________________________________
```

7. Uczenie

W tym momencie model zaczyna proces uczenia. Czyli otrzymuje dwa zbiory danych i etykiet. Pierwszy z nich to dane, na podstawie których model się uczy. Drugi mniejszy zbiór jest zbiorem walidacyjnym, który pozwala na sprawdzenie postępów w nauce, na danych, których model jeszcze nie widział. Pozwala to ocenić postępy w nauce już w czasie uczenia. Kolejny zbiór danych zostanie wykorzystany na końcu celem ostatecznego sprawdzenia poprawności działania modelu.

[8]:
```
history = model.fit(x_train, y_train, epochs=10,␣
↪callbacks=[reduceLR],validation_data=(x_valid, y_valid))
```

```
Epoch 1/10
355/355 [==============================] - 457s 1s/step - loss: 6.2613 -
accuracy: 0.1348 - val_loss: 4.4371 - val_accuracy: 0.1691 - lr: 0.0010
Epoch 2/10
355/355 [==============================] - 441s 1s/step - loss: 4.0533 -
accuracy: 0.1825 - val_loss: 5.6995 - val_accuracy: 0.1498 - lr: 0.0010
Epoch 3/10
355/355 [==============================] - 440s 1s/step - loss: 3.9128 -
accuracy: 0.2051 - val_loss: 2.4049 - val_accuracy: 0.3034 - lr: 0.0010
Epoch 4/10
355/355 [==============================] - 440s 1s/step - loss: 4.1013 -
accuracy: 0.2213 - val_loss: 5.2775 - val_accuracy: 0.1526 - lr: 0.0010
Epoch 5/10
355/355 [==============================] - 441s 1s/step - loss: 4.0949 -
accuracy: 0.2237 - val_loss: 4.0820 - val_accuracy: 0.0923 - lr: 0.0010
Epoch 6/10
355/355 [==============================] - 439s 1s/step - loss: 2.4455 -
accuracy: 0.3519 - val_loss: 1.9172 - val_accuracy: 0.3890 - lr: 1.0000e-06
Epoch 7/10
355/355 [==============================] - 440s 1s/step - loss: 1.7008 -
accuracy: 0.4160 - val_loss: 1.6766 - val_accuracy: 0.4242 - lr: 1.0000e-06
Epoch 8/10
355/355 [==============================] - 441s 1s/step - loss: 1.6356 -
accuracy: 0.4278 - val_loss: 1.6754 - val_accuracy: 0.4218 - lr: 1.0000e-06
```

```
Epoch 9/10
355/355 [==============================] - 441s 1s/step - loss: 1.6348 -
accuracy: 0.4300 - val_loss: 1.6738 - val_accuracy: 0.4242 - lr: 1.0000e-06
Epoch 10/10
355/355 [==============================] - 444s 1s/step - loss: 1.6341 -
accuracy: 0.4293 - val_loss: 1.6738 - val_accuracy: 0.4242 - lr: 1.0000e-09
```

8. Zapis architektury

[9]:
```
if fullStudy == True:
    model.save('Models/ResNet50_full.keras')
else:
    model.save('Models/ResNet50.keras')
```

9. Zapis otrzymanych danych podczas nauki

Po zakończeniu uczenia zapisujemy dane, które otrzymaliśmy podczas uczenie do pliku CSV. Pozwoli nam to później przeanalizować dane w późniejszym czasie.

[10]:
```
historyModelLearning = pd.DataFrame()
historyModelLearning['loss'] = history.history['loss']
historyModelLearning['accuracy'] = history.history['accuracy']
historyModelLearning['val_loss'] = history.history['val_loss']
historyModelLearning['val_accuracy'] = history.history['val_accuracy']
if fullStudy == True:
    historyModelLearning.to_csv('ResultLearning/ResNet50_full.csv', index=True)
else:
    historyModelLearning.to_csv('ResultLearning/ResNet50.csv', index=True)
```

Literatura

1. https://medium.com/@nitishkundu1993/exploring-resnet50-an-in-depth-look-at-the-model-architecture-and-code-implementation-d8d8fa67e46f dostęp 4.10.2024