Splątane sieci neuronowe CNN - architektura Inception

mgr inż. Grzegorz Kossakowski

17.10.2024

1. Opis architektury

Inception [1][2] jest to architektura opracowana przez badaczy Google. Obecnie jest używana wersja trzecia, która wprowadza kilka dodatkowych ulepszeń. w porównaniu do wcześniejszych wersji.

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 keras.callbacks import ReduceLROnPlateau
from keras.optimizers import Adam
from keras import Sequential
from tests.test_layers import Dense, Flatten
from keras.applications import InceptionV3
from keras.applications.inception_v3 import preprocess_input
import pandas as pd
import datetime
from sklearn.metrics import accuracy_score
```

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')
hdu_test = fits.open('Data/test.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
x_test = hdu_test[0].data
y_test = hdu_test[1].data
```

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

[4]:
```
((11350, 256, 256, 3), (2838, 256, 256, 3), (3548, 256, 256, 3), numpy.ndarray)
```

4. Pobranie danych

W tym kroku pobieramy dane, a następnie przygotowujemy je do klasyfikacji. Modele głębokiej sieci neuronowej [1] wymaga danych z zakresu od -1 do 1, dlatego do prawidłowego działania wykorzystamy funkcję preprocess_input, umieszczonej w module inception_v3. Dzięki temu będziemy mieli pewność, że dane zostaną prawidłowo przygotowane.

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

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 Inception v3. Po warstwach splątanych jest warstwa flatten. Zadaniem tej warstwy jest spłaszczenie obrazu z wymiarów otrzymanych po przejściu warstw splątanych do pojedynczego ciągu. Ostatnią warstwą jest gęsto połączona warstwa wyjściowa. W naszym modelu klasyfikacja odbywa się dla 10 kategorii, dlatego ta warstwa ma 10 neuronów.

[7]:
```
base_model = InceptionV3(weights='imagenet', include_top=False,␣
 ↪input_shape=(256, 256,3))
numberLayers = len(base_model.layers)
numberClosedLayers = int(numberLayers/2)
print("Liczba warstw: ", numberLayers)
if fullStudy == True:
    base_model.trainable = True
else:
    for layer in base_model.layers[:numberClosedLayers]:
        layer.trainable = False

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()
```

```
Liczba warstw:  311
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 inception_v3 (Functional)   (None, 6, 6, 2048)        21802784

 flatten (Flatten)           (None, 73728)             0

 dense (Dense)               (None, 10)                737290

=================================================================
Total params: 22540074 (85.98 MB)
Trainable params: 17526730 (66.86 MB)
Non-trainable params: 5013344 (19.12 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]:
```
now = datetime.datetime.now()
history = model.fit(x_train, y_train, epochs=10,␣
 ↪callbacks=[reduceLR],validation_data=(x_valid, y_valid))
time = datetime.datetime.now()-now
print("Potrzebny czas do wykonania operacji to: ",int(time.seconds/60)," minut")
```

```
Epoch 1/10
355/355 [==============================] - 356s 991ms/step - loss: 1.3196 -
accuracy: 0.6087 - val_loss: 1.2774 - val_accuracy: 0.6744 - lr: 0.0010
Epoch 2/10
355/355 [==============================] - 349s 983ms/step - loss: 0.7899 -
accuracy: 0.7366 - val_loss: 0.8286 - val_accuracy: 0.7297 - lr: 0.0010
Epoch 3/10
355/355 [==============================] - 348s 981ms/step - loss: 0.6636 -
accuracy: 0.7772 - val_loss: 1.3467 - val_accuracy: 0.6949 - lr: 0.0010
Epoch 4/10
```

```
355/355 [==============================] - 348s 979ms/step - loss: 0.5611 -
accuracy: 0.8089 - val_loss: 0.6481 - val_accuracy: 0.7921 - lr: 0.0010
Epoch 5/10
355/355 [==============================] - 347s 979ms/step - loss: 0.4877 -
accuracy: 0.8347 - val_loss: 0.7778 - val_accuracy: 0.7625 - lr: 0.0010
Epoch 6/10
355/355 [==============================] - 348s 980ms/step - loss: 0.3917 -
accuracy: 0.8683 - val_loss: 0.9108 - val_accuracy: 0.7530 - lr: 0.0010
Epoch 7/10
355/355 [==============================] - 347s 979ms/step - loss: 0.3658 -
accuracy: 0.8751 - val_loss: 0.8010 - val_accuracy: 0.7875 - lr: 0.0010
Epoch 8/10
355/355 [==============================] - 347s 979ms/step - loss: 0.2208 -
accuracy: 0.9244 - val_loss: 0.7357 - val_accuracy: 0.8009 - lr: 1.0000e-06
Epoch 9/10
355/355 [==============================] - 348s 981ms/step - loss: 0.2065 -
accuracy: 0.9303 - val_loss: 0.7351 - val_accuracy: 0.8006 - lr: 1.0000e-06
Epoch 10/10
355/355 [==============================] - 348s 982ms/step - loss: 0.2110 -
accuracy: 0.9293 - val_loss: 0.7332 - val_accuracy: 0.8023 - lr: 1.0000e-09
Potrzebny czas do wykonania operacji to:  58  minut
```

8. Zapis architektury

[9]:
```
if fullStudy == True:
    model.save('Models/Inception_full.keras')
else:
    model.save('Models/Inception.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/Inception_full.csv', index=True)
else:
    historyModelLearning.to_csv('ResultLearning/Inception.csv', index=True)
```

10. Sprawdzenie uzyskanych wyników

Celem tego elementu jest wstępne sprawdzenie uzyskanych wyników. Pozwoli to na porównanie wyników z predykcją w zapisanym modelu. Dzięki temu uzyskamy informację czy otrzymane wyniku

różnią się od siebie.

[11]:
```
predict = model.predict(x_test).argmax(axis=1)
print("Otrzymany wynik to: ",(accuracy_score(y_test, predict)*100)," %")
```

```
111/111 [==============================] - 54s 483ms/step
Otrzymany wynik to:  78.72040586245772  %
```

Literatura

1. https://keras.io/2.17/api/applications/inceptionv3/ dostęp 11.10.2024
2. https://iq.opengenus.org/inception-v3-model-architecture/#google_vignette dostęp 11.10.2024
3. https://keras.io/guides/serialization_and_saving/ dostęp 15.10.2024