Splątane sieci neuronowe CNN - architektura MobileNet

mgr inż. Grzegorz Kossakowski

8.10.2024

1. Opis architektury

MobileNet [1] jest to architektura, stworzoną przez firmę Google do wykorzystania w urządzeniach mobilnych. W porównaniu do innych CNN celem było, ograniczone liczby parametrów do nauczenia, dzięki temu architektura nadaje się do wykorzystania na dużo słabszych urządzeniach. Osiągnięto to przez wykorzystanie splotu rozdzielonego wgłębnie. Taki splot składa się z dwóch operacji splotu: głęboki i punktowy. Pierwsza wersja została zaprezentowana w 2017.

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 MobileNetV3Large
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. 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.

[5]:
```
fullStudy = False
```

5. 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.

[6]:
```
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
```

6. Budowa modelu.

Model w tym przypadku to MobileNetV3Large. 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 = MobileNetV3Large(weights='imagenet', 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()
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not
224. Weights for input shape (224, 224) will be loaded as the default.
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
```

```
=================================================================
 MobilenetV3large (Function  (None, 1000)              5507432
 al)

 flatten_1 (Flatten)         (None, 1000)              0

 dense (Dense)               (None, 10)                10010

=================================================================
Total params: 5517442 (21.05 MB)
Trainable params: 10010 (39.10 KB)
Non-trainable params: 5507432 (21.01 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 [==============================] - 105s 285ms/step - loss: 2.2628 -
accuracy: 0.1431 - val_loss: 2.2392 - val_accuracy: 0.1508 - lr: 0.0010
Epoch 2/10
355/355 [==============================] - 86s 242ms/step - loss: 2.2322 -
accuracy: 0.1453 - val_loss: 2.2261 - val_accuracy: 0.1388 - lr: 0.0010
Epoch 3/10
355/355 [==============================] - 86s 243ms/step - loss: 2.2267 -
accuracy: 0.1491 - val_loss: 2.2261 - val_accuracy: 0.1388 - lr: 1.0000e-06
Epoch 4/10
355/355 [==============================] - 86s 243ms/step - loss: 2.2267 -
accuracy: 0.1491 - val_loss: 2.2261 - val_accuracy: 0.1388 - lr: 1.0000e-09
Epoch 5/10
355/355 [==============================] - 86s 243ms/step - loss: 2.2267 -
accuracy: 0.1491 - val_loss: 2.2261 - val_accuracy: 0.1388 - lr: 1.0000e-12
Epoch 6/10
355/355 [==============================] - 86s 243ms/step - loss: 2.2268 -
accuracy: 0.1491 - val_loss: 2.2261 - val_accuracy: 0.1388 - lr: 1.0000e-15
Epoch 7/10
355/355 [==============================] - 86s 243ms/step - loss: 2.2268 -
accuracy: 0.1491 - val_loss: 2.2261 - val_accuracy: 0.1388 - lr: 1.0000e-18
Epoch 8/10
355/355 [==============================] - 86s 243ms/step - loss: 2.2267 -
```

```
accuracy: 0.1491 - val_loss: 2.2261 - val_accuracy: 0.1388 - lr: 1.0000e-21
Epoch 9/10
355/355 [==============================] - 86s 244ms/step - loss: 2.2267 -
accuracy: 0.1491 - val_loss: 2.2261 - val_accuracy: 0.1388 - lr: 1.0000e-24
Epoch 10/10
355/355 [==============================] - 87s 244ms/step - loss: 2.2267 -
accuracy: 0.1491 - val_loss: 2.2261 - val_accuracy: 0.1388 - lr: 1.0000e-27
```

8. Zapis architektury

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

Literatura

1. https://towardsdatascience.com/everything-you-need-to-know-about-mobilenetv3-and-its-comparison-with-previous-versions-a5d5e5a6eeaa dostęp 2024-01-04