Splątane sieci neuronowe CNN - architektura VGG-16

mgr inż. Grzegorz Kossakowski

8.10.2024

1. Opis architektury

Architektura została zaprojektowana przez Visual Geomerty Group (VGG) na Uniwersytecie oksfordzkim. Charakteryzuje się prostotą i skutecznością, jednocześnie umożliwia naukę skomplikowanych hierarchicznych reprezentacji cech wizualnych. Dzięki temu mimo prostoty architektury można uzyskać solidne i dokładne przewidywania.

Zbudowana jest z szesnastu warstw. w tym 13 warstw splątanych i trzech warstw w pełni połączonych.

VGG-16 został zaprezentowany w 2014 roku przez Karen Simonyan i Andrew Zissermana na corocznym konkursie ImageNet Large Scale Visual Recognition Challenge (ILSVRC), gdzie odniósł bardzo wielki sukces.

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 VGG16
import pandas as pd
```

3. 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.

[3]:
```
fullStudy = False
```

4. 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.

[4]:
```
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
```

[5]:
```
x_train.shape, x_valid.shape, type(x_train)
```

[5]:
```
((11350, 256, 256, 3), (2838, 256, 256, 3), numpy.ndarray)
```

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 jest gotowy, dlatego nie musimy go budować od początku. Po dodaniu Jest to model warstwowy i jako pierwsza warstwa jest to warstwa flatten. Zadaniem tej warstwy jest spłaszczenie obrazu z wymiarów 256*256 na pojedynczy ciąg, jest to warstwa wejściowa. Kolejną warstwą jest warstwa ukryta z aktywatorem RELU. Aktywator ten powoduje, że każdy otrzymany wynik ujemy, zostaje zamieniony na zero [5][6]. Pozwala to na przełamanie liniowości procesu. Ostatnią warstwą jest gęsto połączona warstwa wyjściowa. W naszym modelu klasyfikacja odbywa się dla 10 kategorii dlatego właśnie taka.

[7]:
```
base_model = VGG16(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 #
=================================================================
 vgg16 (Functional)          (None, 8, 8, 512)         14714688

 flatten (Flatten)           (None, 32768)             0

 dense (Dense)               (None, 10)                327690

=================================================================
Total params: 15042378 (57.38 MB)
Trainable params: 327690 (1.25 MB)
Non-trainable params: 14714688 (56.13 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 [==============================] - 1062s 3s/step - loss: 1.7897 -
accuracy: 0.3915 - val_loss: 1.3430 - val_accuracy: 0.5134 - lr: 0.0010
Epoch 2/10
355/355 [==============================] - 1046s 3s/step - loss: 1.3396 -
accuracy: 0.5339 - val_loss: 1.4697 - val_accuracy: 0.5106 - lr: 0.0010
Epoch 3/10
355/355 [==============================] - 1045s 3s/step - loss: 1.1719 -
accuracy: 0.5924 - val_loss: 1.2396 - val_accuracy: 0.5715 - lr: 0.0010
Epoch 4/10
355/355 [==============================] - 1046s 3s/step - loss: 1.0244 -
accuracy: 0.6385 - val_loss: 1.3102 - val_accuracy: 0.5444 - lr: 0.0010
Epoch 5/10
355/355 [==============================] - 1047s 3s/step - loss: 0.9227 -
accuracy: 0.6795 - val_loss: 1.1758 - val_accuracy: 0.6025 - lr: 0.0010
Epoch 6/10
```

```
355/355 [==============================] - 1046s 3s/step - loss: 0.8468 -
accuracy: 0.6980 - val_loss: 1.3358 - val_accuracy: 0.5437 - lr: 0.0010
Epoch 7/10
355/355 [==============================] - 1046s 3s/step - loss: 0.7490 -
accuracy: 0.7407 - val_loss: 1.3680 - val_accuracy: 0.5599 - lr: 0.0010
Epoch 8/10
355/355 [==============================] - 1046s 3s/step - loss: 0.7180 -
accuracy: 0.7521 - val_loss: 1.3165 - val_accuracy: 0.5641 - lr: 0.0010
Epoch 9/10
355/355 [==============================] - 1048s 3s/step - loss: 0.6685 -
accuracy: 0.7694 - val_loss: 1.3951 - val_accuracy: 0.5511 - lr: 0.0010
Epoch 10/10
355/355 [==============================] - 1049s 3s/step - loss: 0.6609 -
accuracy: 0.7774 - val_loss: 1.2011 - val_accuracy: 0.6008 - lr: 0.0010
```

8. Zapis architektury

[9]:
```
if fullStudy == True:
    model.save('Models/VGG16_full.keras')
else:
    model.save('Models/VGG16.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/VGG16_full.csv', index=True)
else:
    historyModelLearning.to_csv('ResultLearning/VGG16.csv', index=True)
```

Literatura

1. https://www.geeksforgeeks.org/vgg-16-cnn-model/