# ArchitectureInceptionResNet

November 17, 2024

## 0.1 # Splątane sieci neuronowe CNN – architektura InceptionResNet

autor: mgr inż. Grzegorz Kossakowski

## 0.2 1. Opis architektury

InceptionResNet [1][2] jest to połączenie dwóch sieci Inception oraz ResNet. Spowodowało to znaczne przyspieszenie szkolenia sieci i pozwoliło poprawić wydajność oraz otrzymywane wyniki.

## 0.3 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 InceptionResNetV2
from keras.applications.inception_resnet_v2 import preprocess_input
import pandas as pd
import datetime
from sklearn.metrics import accuracy_score
```

## 0.4 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)
```

### 0.5 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 = preprocess_input(x_train)
x_valid = preprocess_input(x_valid)
x_test = preprocess_input(x_test)
```

### 0.6 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
```

### 0.7 6. Budowa modelu.

Model w tym przypadku jest bardzo prosty. Jest to model warstwowy i jako pierwsza warstwa jest to warstwa flatten. Zadaniem tej warstwy jest spłaszczenie obrazu z wymiarów 69*69 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 = InceptionResNetV2(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:  780
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 inception_resnet_v2 (Funct  (None, 6, 6, 1536)        54336736
 ional)

 flatten (Flatten)           (None, 55296)             0

 dense (Dense)               (None, 10)                552970

=================================================================
Total params: 54889706 (209.39 MB)
Trainable params: 42473962 (162.03 MB)
Non-trainable params: 12415744 (47.36 MB)
_________________________________________________________________
```

### 0.8 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 [==============================] - 1044s 3s/step - loss: 1.2119 -
accuracy: 0.6656 - val_loss: 12.9242 - val_accuracy: 0.6290 - lr: 0.0010
Epoch 2/10
```

```
355/355 [==============================] - 1024s 3s/step - loss: 0.6778 -
accuracy: 0.7841 - val_loss: 0.5980 - val_accuracy: 0.8228 - lr: 0.0010
Epoch 3/10
355/355 [==============================] - 1018s 3s/step - loss: 0.5912 -
accuracy: 0.8062 - val_loss: 1177.5544 - val_accuracy: 0.2185 - lr: 0.0010
Epoch 4/10
355/355 [==============================] - 1018s 3s/step - loss: 0.4933 -
accuracy: 0.8382 - val_loss: 102.4331 - val_accuracy: 0.5990 - lr: 0.0010
Epoch 5/10
355/355 [==============================] - 1017s 3s/step - loss: 0.4271 -
accuracy: 0.8570 - val_loss: 7.3036 - val_accuracy: 0.2935 - lr: 0.0010
Epoch 6/10
355/355 [==============================] - 1021s 3s/step - loss: 0.4149 -
accuracy: 0.8596 - val_loss: 0.6027 - val_accuracy: 0.7988 - lr: 0.0010
Epoch 7/10
355/355 [==============================] - 1030s 3s/step - loss: 0.2043 -
accuracy: 0.9344 - val_loss: 0.6421 - val_accuracy: 0.8073 - lr: 1.0000e-06
Epoch 8/10
355/355 [==============================] - 1025s 3s/step - loss: 0.1904 -
accuracy: 0.9362 - val_loss: 0.6352 - val_accuracy: 0.8108 - lr: 1.0000e-06
Epoch 9/10
355/355 [==============================] - 1017s 3s/step - loss: 0.1860 -
accuracy: 0.9406 - val_loss: 0.6404 - val_accuracy: 0.8104 - lr: 1.0000e-09
Epoch 10/10
355/355 [==============================] - 1038s 3s/step - loss: 0.1846 -
accuracy: 0.9389 - val_loss: 0.6383 - val_accuracy: 0.8101 - lr: 1.0000e-12
Potrzebny czas do wykonania operacji to:  170  minut
```

### 0.9 8. Zapis architektury

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

### 0.10 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/InceptionResNet_full.csv',␣
 ↪index=True)
```

```
else:
    historyModelLearning.to_csv('ResultLearning/InceptionResNet.csv',␣
 ↪index=True)
```

## 0.11 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 [==============================] - 147s 1s/step
Otrzymany wynik to:  80.55242390078917  %
```

## 0.12 Literatura


1. https://arxiv.org/abs/1602.07261 dostęp 11.10.2024
2. https://keras.io/2.17/api/applications/inceptionresnetv2/ dostęp 11.10.2024


[ ]:
```
```