# Splątane sieci neuronowe CNN - architektura Xception

mgr inż. Grzegorz Kossakowski

17.10.2024

## 1. Opis architektury

Xception [1][2][3] jest architekturą splątanych sieci neuronowych CNN. Pełna nazwa architektury to Extreme Inception. Powstał w 2017 roku jako ewolucja architektury Inception i został stworzony przez firmę Google. Architektura Xception okazała się bardziej wydajna od VGG-16, ResNet i Inception v3. Głęboko rozdzielny sploty są uznawane za znacznie bardziej wydajne pod względem czasu obliczeń. Znakiem firmowym Xception jest wykorzystanie głęboko rozdzielnych splotów. Jest to bardzo wydajna architektura oparta na dwóch krokach: - Głęboko rozdzielny splot - Splot punktowy

Ogólna architektura składa się z trzech przepływów: - Przepływ wejściowy - Przepływ środkową, w której proces jest powtarzany osiem razy - Przepływ wyjściowy.

## 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 Xception
from keras.applications.xception 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
```

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

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

### 6. Budowa modelu.

Model został stworzony w 2017 roku. Jest to model, który powstał na podstawie modelu Inception. Po wykonaniu warstw splątanych następuje przejście przez warstwę flatten. Zadaniem tej warstwy jest spłaszczenie obrazu z wymiarów, które zostały po przejściu przez warstwy splątane na pojedynczy ciąg. Ostatnią warstwą jest gęsto połączona warstwa wyjściowa. W naszym modelu klasyfikacja odbywa się dla 10 kategorii dlatego właśnie taka.

[7]:
```
base_model = Xception(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:  132
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 xception (Functional)       (None, 8, 8, 2048)        20861480

 flatten (Flatten)           (None, 131072)            0

 dense (Dense)               (None, 10)                1310730

=================================================================
Total params: 22172210 (84.58 MB)
Trainable params: 16168994 (61.68 MB)
Non-trainable params: 6003216 (22.90 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.

[9]:
```
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 [==============================] - 812s 2s/step - loss: 1.2753 -
accuracy: 0.5646 - val_loss: 1.0143 - val_accuracy: 0.6920 - lr: 0.0010
Epoch 2/10
355/355 [==============================] - 803s 2s/step - loss: 0.8074 -
accuracy: 0.7297 - val_loss: 0.8686 - val_accuracy: 0.7230 - lr: 0.0010
Epoch 3/10
355/355 [==============================] - 802s 2s/step - loss: 0.6164 -
accuracy: 0.7944 - val_loss: 1.0672 - val_accuracy: 0.7016 - lr: 0.0010
Epoch 4/10
355/355 [==============================] - 802s 2s/step - loss: 0.4967 -
accuracy: 0.8369 - val_loss: 0.8631 - val_accuracy: 0.7209 - lr: 0.0010
Epoch 5/10
355/355 [==============================] - 802s 2s/step - loss: 0.3741 -
accuracy: 0.8741 - val_loss: 1.1352 - val_accuracy: 0.7516 - lr: 0.0010
Epoch 6/10
355/355 [==============================] - 811s 2s/step - loss: 0.2700 -
accuracy: 0.9101 - val_loss: 0.9435 - val_accuracy: 0.7593 - lr: 0.0010
Epoch 7/10
355/355 [==============================] - 802s 2s/step - loss: 0.1994 -
accuracy: 0.9339 - val_loss: 1.4125 - val_accuracy: 0.7442 - lr: 0.0010
Epoch 8/10
355/355 [==============================] - 802s 2s/step - loss: 0.1436 -
accuracy: 0.9519 - val_loss: 1.9569 - val_accuracy: 0.6600 - lr: 0.0010
Epoch 9/10
355/355 [==============================] - 803s 2s/step - loss: 0.1807 -
accuracy: 0.9441 - val_loss: 1.4221 - val_accuracy: 0.7572 - lr: 0.0010
Epoch 10/10
355/355 [==============================] - 807s 2s/step - loss: 0.1047 -
accuracy: 0.9681 - val_loss: 1.0122 - val_accuracy: 0.7706 - lr: 1.0000e-06
Potrzebny czas do wykonania operacji to:  134  minut
```

### 8. Zapis architektury

Jednak my nie będziemy testować od razu naszego modelu. Do tego celu przygotujemy oddzielny notebook. Dlatego, aby nie utracić naszej pracy, zapiszemy nas wyuczony model do pliku.

[10]:
```
if fullStudy == True:
    model.save('Models/Xception_full.keras')
else:
    model.save('Models/Xception.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.

[11]:
```
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/Xception_full.csv', index=True)
else:
    historyModelLearning.to_csv('ResultLearning/Xception.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.

[12]:
```
predict = model.predict(x_test).argmax(axis=1)
print("Otrzymany wynik to: ",(accuracy_score(y_test, predict)*100)," %")
```

```
111/111 [==============================] - 104s 937ms/step
Otrzymany wynik to:  76.52198421645998  %
```

### Literatura


1. https://keras.io/2.17/api/applications/xception/
2. https://maelfabien.github.io/deeplearning/xception/#
3. https://towardsdatascience.com/review-xception-with-depthwise-separable-convolution-better-than-inception-v3-image-dc967dd42568
4. https://medium.com/@saba99/xception-cd1adc84290f