

# Splątane sieci neuronowe CNN - architektura AlexNet

mgr inż. Grzegorz Kossakowski

5.10.2024

## 1. Opis architektury

AlexNet [1][2] jest architekturą splątanych sieci neuronowych CNN. Została stworzona przez Alex Kizhevsky i Ilya Sutskever. W pracach uczestniczył również Geoffrey Hinton, który był promotorem doktoratu as Krizhevsky. Jest bardziej rozbudowany niż LeNet5. Głównym celem było uczestnictwo w zawodach ImageNet Large Scale Visual Recognition Challenge (ILSVRC), które wygrał w 2012 roku.

## 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.layers import Conv2D, Dropout, Flatten, Dense, MaxPool2D,
↳BatchNormalization
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 uczący, 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. 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 = x_train / 255.0
x_valid = x_valid / 255.0
```

## 5. Budowa modelu.

Model można podzielić na dwie podstawowe części. Pierwsza część to warstwy spletane. Naprzemiennie są układane warstwy Conv2D, BatchNormalization oraz MaxPool2D. Kolejność ułożenia warstw jest zgodna z AlexNet. Kolejne warstwy zaczynające się od warstwy flatten są zwykłym modelem głębokiego uczenia. Zadaniem warstwy flatten jest spłaszczenie obrazu z wymiarów 256\*256 na pojedynczy ciąg. Kolejną warstwą jest warstwa ukryta z aktywatorem RELU. Aktywator ten powoduje, że każdy otrzymany wynik ujemny, zostaje zamieniony na zero [3][4]. 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 wartość jest wybrana.

```
[6]: model = Sequential()
model.add(Conv2D(filters=96, kernel_size=(11, 11), strides=(4, 4),
    ↪activation="relu", input_shape=(256, 256, 3)))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(3, 3), strides=(2, 2)))
model.add(Conv2D(filters=256, kernel_size=(5, 5), strides=(1, 1),
    ↪activation="relu", padding="same"))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(3, 3), strides=(2, 2)))
model.add(Conv2D(filters=384, kernel_size=(3, 3), strides=(1, 1),
    ↪activation="relu", padding="same"))
model.add(BatchNormalization())
model.add(Conv2D(filters=384, kernel_size=(3, 3), strides=(1, 1),
    ↪activation="relu", padding="same"))
model.add(BatchNormalization())
model.add(Conv2D(filters=256, kernel_size=(3, 3), strides=(1, 1),
    ↪activation="relu", padding="same"))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(3, 3), strides=(2, 2)))
model.add(Flatten())
model.add(Dense(4096, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(10, activation="softmax"))
model_optimizer = Adam(learning_rate=0.001)
```

```
model.compile(optimizer=model_optimizer, loss='sparse_categorical_crossentropy',
↳metrics=['accuracy'])
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 62, 62, 96)	34944
batch_normalization (Batch Normalization)	(None, 62, 62, 96)	384
max_pooling2d (MaxPooling2D)	(None, 30, 30, 96)	0
conv2d_1 (Conv2D)	(None, 30, 30, 256)	614656
batch_normalization_1 (Batch Normalization)	(None, 30, 30, 256)	1024
max_pooling2d_1 (MaxPooling2D)	(None, 14, 14, 256)	0
conv2d_2 (Conv2D)	(None, 14, 14, 384)	885120
batch_normalization_2 (Batch Normalization)	(None, 14, 14, 384)	1536
conv2d_3 (Conv2D)	(None, 14, 14, 384)	1327488
batch_normalization_3 (Batch Normalization)	(None, 14, 14, 384)	1536
conv2d_4 (Conv2D)	(None, 14, 14, 256)	884992
batch_normalization_4 (Batch Normalization)	(None, 14, 14, 256)	1024
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 256)	0
flatten (Flatten)	(None, 9216)	0
dense (Dense)	(None, 4096)	37752832
dropout (Dropout)	(None, 4096)	0

dense\_1 (Dense) (None, 10) 40970

```
=====
Total params: 41546506 (158.49 MB)
Trainable params: 41543754 (158.48 MB)
Non-trainable params: 2752 (10.75 KB)
-----
```

## 6. 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.

```
[7]: history = model.fit(x_train, y_train, epochs=10,
    ↪callbacks=[reduceLR], validation_data=(x_valid, y_valid))
```

```
Epoch 1/10
355/355 [=====] - 251s 700ms/step - loss: 6.3898 -
accuracy: 0.2207 - val_loss: 3.6178 - val_accuracy: 0.1515 - lr: 0.0010
Epoch 2/10
355/355 [=====] - 242s 681ms/step - loss: 1.8181 -
accuracy: 0.3204 - val_loss: 1.7775 - val_accuracy: 0.3277 - lr: 0.0010
Epoch 3/10
355/355 [=====] - 241s 678ms/step - loss: 1.6687 -
accuracy: 0.3731 - val_loss: 1.12617 - val_accuracy: 0.1589 - lr: 0.0010
Epoch 4/10
355/355 [=====] - 240s 676ms/step - loss: 1.6024 -
accuracy: 0.3981 - val_loss: 2.1563 - val_accuracy: 0.2579 - lr: 0.0010
Epoch 5/10
355/355 [=====] - 240s 675ms/step - loss: 1.5221 -
accuracy: 0.4335 - val_loss: 1.6190 - val_accuracy: 0.4235 - lr: 0.0010
Epoch 6/10
355/355 [=====] - 240s 676ms/step - loss: 1.4773 -
accuracy: 0.4545 - val_loss: 2.1031 - val_accuracy: 0.3612 - lr: 0.0010
Epoch 7/10
355/355 [=====] - 239s 674ms/step - loss: 1.4113 -
accuracy: 0.4856 - val_loss: 3.2347 - val_accuracy: 0.3555 - lr: 0.0010
Epoch 8/10
355/355 [=====] - 240s 676ms/step - loss: 1.3597 -
accuracy: 0.5035 - val_loss: 1.2892 - val_accuracy: 0.5444 - lr: 0.0010
Epoch 9/10
355/355 [=====] - 238s 672ms/step - loss: 1.2944 -
accuracy: 0.5326 - val_loss: 1.3105 - val_accuracy: 0.5627 - lr: 0.0010
Epoch 10/10
355/355 [=====] - 240s 676ms/step - loss: 1.4049 -
```

accuracy: 0.4947 - val\_loss: 1.4003 - val\_accuracy: 0.4912 - lr: 0.0010

## 7. Zapis architektury

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

```
[8]: model.save('Models/AlexNet_full.keras')
```

## 8. Zapis otrzymanych danych podczas nauki

Po zakończeniu uczenia zapisujemy dane, które otrzymaliśmy podczas uczenia do pliku CSV. Pozwoli nam to później przeanalizować proces uczenia i walidacji i porównać te dane z różnymi modelami.

```
[9]: 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']
historyModelLearning.to_csv('ResultLearning/AlexNet_full.csv', index=True)
```

## Literatura

1. <https://medium.com/@siddheshb008/alexnet-architecture-explained-b6240c528bd5>   dostęp 4.10.2024
2. Bartosz Michalski, Małgorzata Plechawska-Wójcik, Porównanie modeli LeNet-5, AlexNet i GoogLeNet w rozpoznawaniu pisma ręcznego, 2022
3. <https://builtin.com/machine-learning/relu-activation-function>
4. <https://datascience.eu/pl/uczenie-maszynowe/relu-funkcja-aktywujaca>