Splątane sieci neuronowe CNN – architektura LeNet5

Grzegorz Kossakowski

5 sierpnia 2024

1. Opis

Celem projektu jest ustalenie jak różne architektury głębokiego uczenia, sprawują się na danych astronomicznych. Do tego celu zostaną wykorzystane dane z projektu AstroNN [1]. Projekt ten został zbudowany z danych zgromadzonych w projekcie Sloan Digital Sky Survey [2] (SDSS), prowadzący szczegółowy obraz nieba. Na podstawie tych danych zostały przygotowane obrazy galaktyk, które są w rozmiarze 64 * 64 * 3. Dodatkowo w projekcie AstroNN są wykorzystane dane zgromadzone w projekcie Galaxy Zoo [3], gdzie dokonywana jest klasyfikacja galaktyk. Na podstawie tych danych powstały etykiety, które są wykorzystywane w projekcie AstroNN.

LeNet5 jest to architektura przedstawiona przez Yann LeCun w 1989 roku [4]. Jest jedną z najstarszych, a zarazem najprostszych architektur sieci splątanych. Służyła do rozpoznawania kodów pocztowych napisanych ręcznie na listach i dobrze sprawdzała się w tej roli, udowadniając, że jest przydatna również w zastosowaniach komercyjnych.

2. Pobranie potrzebnych bibliotek

Kolejnym krokiem jest wczytanie wszystkich potrzebnych bibliotek, dzięki którym będzie możliwe wykorzystanie ich w procesie klasyfikacji.

[1]:
```
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Conv2D, AveragePooling2D, Flatten, Dense
from keras.callbacks import ReduceLROnPlateau
from keras.optimizers import Adam

from astroNN.datasets import galaxy10sdss
import pandas as pd
from datetime import date
import pathlib
```

3. Utworzenie daty

Następnie ustawiamy datę. Jest to potrzebne do generowania plików wynikowych. W obecnym pliku zostaną wykonane jedynie badania, a wyniki zostaną opracowane w późniejszym czasie. Ma to pozwolić zmniejszyć wymagania sprzętowe przy wykonywaniu projektu.

[2]:
```
today = date.today()
day = today.strftime("%Y-%m-%d")
pathlib.Path('./Results/'+day).mkdir(parents=True, exist_ok=True)
```

4. Przygotowanie danych

W tym kroku pobieramy dane, a następnie przygotowujemy je do klasyfikacji. Modele głębokiej sieci neuronowej [5] 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.

[3]:
```
images, labels = galaxy10sdss.load_data()
x_train, x_test, y_train, y_test = train_test_split(images, labels, test_size=0.
 ↪2)
x_train, x_valid, y_train, y_valid = train_test_split(x_train, y_train,␣
 ↪test_size=0.2)

features = ['Disk, Face-on, No Spiral', 'Smooth, Completely round', 'Smooth,␣
 ↪in-between round', 'Smooth, Cigar shaped', 'Disk, Edge-on, Rounded Bulge',␣
 ↪'Disk, Edge-on, Boxy Bulge','Disk, Edge-on, No Bulge','Disk, Face-on, Tight␣
 ↪Spiral', 'Disk, Face-on, Medium Spiral', 'Disk, Face-on, Loose Spiral']

x_train = x_train / 255.0
x_valid = x_valid / 255.0
x_test = x_test / 255.0
```

```
/home/grzegorz/.astroNN/datasets/Galaxy10.h5 was found!
```

[4]:
```
reduceLR = ReduceLROnPlateau(monitor='accuracy', factor=.001, patience=1,␣
 ↪min_delta=0.01, mode="auto")
x_train.shape, x_valid.shape, x_test.shape
```

[4]:
```
((13942, 69, 69, 3), (3486, 69, 69, 3), (4357, 69, 69, 3))
```

5. 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 [6] [7]. 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.

[5]:
```

```

```
model = Sequential()
model.add(Conv2D(filters=6, kernel_size=(5,5), strides=(1,1), activation='tanh',␣
 ↪input_shape=(69,69,3)))
model.add(AveragePooling2D(pool_size=(2,2), strides=(2,2)))
model.add(Conv2D(filters=16, kernel_size=(5,5), strides=(1,1),␣
 ↪activation='tanh'))
model.add(AveragePooling2D(pool_size=(2,2), strides=(2,2)))
model.add(Flatten())
model.add(Dense(units=120, activation='tanh'))
model.add(Dense(units=84, activation='tanh'))
model.add(Dense(units=10, activation='softmax'))
model_optimizer = Adam(learning_rate=0.001)

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

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. Kolejny zbiór danych zostanie wykorzystany na końcu celem ostatecznego sprawdzenia poprawności działania modelu.

[6]:
```
history = model.fit(x_train, y_train, epochs=10,␣
 ↪callbacks=[reduceLR],validation_data=(x_valid, y_valid))
```

```
Epoch 1/10
436/436 [==============================] - 13s 29ms/step - loss: 1.3536 -
accuracy: 0.4757 - val_loss: 1.0059 - val_accuracy: 0.6474 - lr: 0.0010
Epoch 2/10
436/436 [==============================] - 12s 28ms/step - loss: 0.8956 -
accuracy: 0.6764 - val_loss: 0.8191 - val_accuracy: 0.7123 - lr: 0.0010
Epoch 3/10
436/436 [==============================] - 12s 28ms/step - loss: 0.7713 -
accuracy: 0.7196 - val_loss: 0.7807 - val_accuracy: 0.7217 - lr: 0.0010
Epoch 4/10
436/436 [==============================] - 12s 27ms/step - loss: 0.6842 -
accuracy: 0.7496 - val_loss: 0.7625 - val_accuracy: 0.7226 - lr: 0.0010
Epoch 5/10
436/436 [==============================] - 12s 28ms/step - loss: 0.6099 -
accuracy: 0.7773 - val_loss: 0.7422 - val_accuracy: 0.7326 - lr: 0.0010
Epoch 6/10
436/436 [==============================] - 11s 26ms/step - loss: 0.5360 -
accuracy: 0.8028 - val_loss: 0.7711 - val_accuracy: 0.7194 - lr: 0.0010
Epoch 7/10
```

```
436/436 [==============================] - 11s 25ms/step - loss: 0.4749 -
accuracy: 0.8300 - val_loss: 0.7687 - val_accuracy: 0.7281 - lr: 0.0010
Epoch 8/10
436/436 [==============================] - 11s 25ms/step - loss: 0.4121 -
accuracy: 0.8541 - val_loss: 0.7986 - val_accuracy: 0.7217 - lr: 0.0010
Epoch 9/10
436/436 [==============================] - 12s 27ms/step - loss: 0.3467 -
accuracy: 0.8824 - val_loss: 0.7965 - val_accuracy: 0.7344 - lr: 0.0010
Epoch 10/10
436/436 [==============================] - 11s 25ms/step - loss: 0.2916 -
accuracy: 0.8992 - val_loss: 0.8072 - val_accuracy: 0.7447 - lr: 0.0010
```

7. 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.

[7]:
```
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('./Results/'+day+'/
 ↪HistoryModelLearningLeNet5-'+day+'.csv', index=True)
```

8. Sprawdzenie modelu na danych testowych.

To jest ostateczne sprawdzenie danych. W tym sprawdzeniu otrzymamy również nie tylko informację jak jest współczynnik błędu klasyfikacji, ale również będziemy mogli przeglądać, które obrazy faktycznie zostały źle zaklasyfikowane i dzięki temu będzie możliwe poprawienie modelu.

[8]:
```
predict = model.predict(x_test).argmax(axis=1)
```

```
137/137 [==============================] - 0s 2ms/step
```

9. Zapis wyników testów do pliku CSV.

To na postawie tych danych będziemy w stanie dokładniej stwierdzić, z czego wynikają problemy z klasyfikacją.

[9]:
```
result = pd.DataFrame()
result['predict'] = predict
result['test'] = y_test
result.to_csv('./Results/'+day+'/ResultLeNet5-'+day+'.csv', index=False)
```

Literatura

1. https://astronn.readthedocs.io/en/stable/galaxy10sdss.html

2. https://www.sdss4.org/

3. https://www.zooniverse.org/projects/zookeeper/galaxy-zoo/

4. Bartosz Michalski, Małgorzata Plechawska-Wójcik, Porównanie modeli LeNet-5, AlexNet i GoogLeNet w rozpoznawaniu pisma ręcznego, 2022

5. Paweł Krakowiak, Deep learning w języku Python — Konwolucyjne Sieci Neuronowe

6. https://builtin.com/machine-learning/relu-activation-function

7. https://datascience.eu/pl/uczenie-maszynowe/relu-funkcja-aktywujaca/