Metody sztucznej inteligencji 2
Projekt 2. — Rozpoznawanie kształtów w czasie rzeczywistym
Raport końcowy

Bartłomiej Dach Tymon Felski

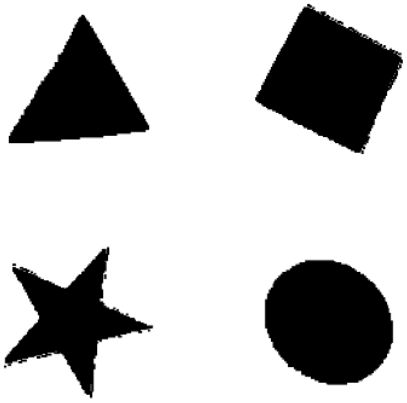
28 maja 2018

Poniższy dokument zawiera końcowy opis projektu, którego celem jest zaimplementowanie rozwiązania pozwalającego na detekcję i rozpoznawanie prostych kształtów na obrazach za pomocą metod geometrycznych oraz z użyciem sieci neuronowych i porównanie skuteczności oraz wydajności obu wariantów.

1. Opis problemu badawczego

Zagadnienie rozpoznawania obiektów na obrazach z użyciem metod sztucznej inteligencji stanowi duży obszar zainteresowań naukowych. Efektywne rozwiązanie tego zagadnienia pomogłoby usprawnić wiele procesów, dotychczas wymagających ludzkiej kontroli i interwencji.

W niniejszym projekcie zbiór rozpoznawanych typów obiektów został zawężony do czterech rodzajów kształtów geometrycznych: trójkąta, kwadratu, koła i gwiazdy pięcioramiennej (patrz rys, 1). Kształty mogą znajdować się w dowolnej orientacji i mieć dowolny rozmiar. Dla uproszczenia zakładamy również, że kolor rozpoznawanych obiektów jest jednakowy i znany *a priori*.



Rysunek 1: Przykładowe kształty ze zbioru treningowego używanego do treningu i oceny jakości klasyfikatorów

1.1. Cel badań

Kształty geometryczne wykazują pewien stopień regularności, co powoduje, że do ich rozpoznawania można próbować zastosować algorytmy dokładne. Przykładem są zaimplementowane w bibliotece OpenCV [5]:

- algorytm Suzukiego [12] do wyznaczania konturów na podstawie obrazów rastrowych,
- algorytm Douglasa-Peuckera [4] do upraszczania konturów i redukcji liczby punktów.

Dzięki tym dwóm algorytmom można na podstawie binarnego obrazu wejściowego wyznaczyć kontur kształtu — łamaną zamkniętą przybliżającą kształt wyznaczony przez piksele kształtu na tle.

Oprócz metod dokładnych, do zagadnienia rozpoznawania można również użyć sieci neuronowych, które zamiast wiedzy dokładnej używają przygotowanych zbiorów treningowych oraz metod optymalizacji, aby dokonać klasyfikacji. Celem badań jest porównanie wydajności i skuteczności wybranych metod.

1.2. Wykorzystane techniki

Do zaimplementowania klasyfikatorów użyty został język skryptowy **Python** w wersji 3.5.2. Ponadto wykorzystane zostały dodatkowe biblioteki, wymienione w tabeli 1. Podejście geometryczne opiera się na funkcjach udostępnianych przez **OpenCV**, podczas gdy sieci neuronowe stworzone zostały za pośrednictwem API dostarczanego przez bibliotekę **Keras**.

Nr	**Komponent, wersja**	**Opis**	**Licencja**	
1	h5py, 2.7.1	Interfejs dla binarnego formatu danych HDF5, wymagany przez Keras	BSD License	[3]
2	Keras, 2.1.4	Biblioteka udostępniająca API do sieci neuronowych	MIT License	[2]
3	Matplotlib, 2.1.0	Umożliwia tworzenie wykresów	Matplotlib License	[6]
4	NumPy, 1.13.3	Używana do efektywnych obliczeń na wektorach n-wymiarowych	BSD License	[10]
5	OpenCV, 3.3.0	Biblioteka do obróbki obrazów	New BSD License	[5]
6	pandas, 0.21.0	Wspomaga ładowanie danych z plików CSV oraz ich analizę	BSD License	[7]
7	TensorFlow, 1.8.0	Framework do uczenia maszynowego używany przez Keras	Apache 2.0	[13]

Tablica 1: Wykorzystane biblioteki wraz z określeniem licencji

1.2.1. Wycięcie kształtów z tła

Ponieważ projekt zakłada, że kolor kształtów jest zadany z góry, do ekstrakcji kształtów z tła wystarczy zastosować operację progowania obrazu. Pozwala ona na zamianę obrazu w kolorze na obraz binarny. Progowanie wykonywane jest w przestrzeni kolorów HSV, która dobrze odwzorowuje podobieństwo odcieni barw; wystarczy zdefiniować przedział dla parametru H, który określa sąsiedztwo tonalne żądanego koloru.

1.2.2. Klasyfikator geometryczny

Główną ideą działania klasyfikatora geometrycznego jest dopasowanie łamanej zamkniętej do rozpoznanego kształtu i analiza kształtu tej łamanej.

Do odróżniania kształtów wykorzystywana jest przede wszystkim liczba wierzchołków w wyznaczonym konturze. Pozwala ona na proste odróżnienie trójkątów, czworokątów i dziesięciokątów od pozostałych kształtów. Ponadto, dla każdego kształtu innego niż trójkąt, stosowane są odpowiednie heurystyki:

- W przypadku kwadratów, sprawdzana jest proporcja wysokości do szerokości kształtu (powinna być bliska jedności). Ponadto bok kwadratu wyznaczany jest na podstawie pola i obwodu figury; jeśli dwie wyznaczone wartości nie są bliskie sobie, kształt jest odrzucany.
- Rozpoznawanie gwiazd korzysta z faktu, że wierzchołki parzyste i nieparzyste gwiazdy znajdują się w innych odległościach od środka figury. Zatem, jeśli odchylenie standardowe odległości od środka wierzchołków parzystych i nieparzystych jest zbyt duże, figura jest odrzucana.
- Dla kół, podobnie jak dla kwadratów, promień koła wyznaczany jest dwukrotnie na podstawie pola i obwodu figury. W przypadku zbyt dużej różnicy kształt nie jest klasyfikowany.

1.2.3. Klasyfikatory oparte na sieciach neuronowych

Klasyfikatory oparte na sieciach były trenowane na podstawie zbioru treningowego [8]. Zostały opracowane dwa warianty sieci, operujące na obrazie wejściowym rozmiarów 64×64:

- W pierwszym wariancie wejściowy binarny obraz dwuwymiarowy spłaszczany jest do jednowymiarowego wektora, który przekazywany jest na wejście sieci. Użyta sieć korzysta z trzech warstw ukrytych z funkcją aktywacji ReLU (ang. *Rectified Linear Unit*). Ponadto, po każdej warstwie ukrytej dodana jest warstwa *dropout*, zapobiegająca zjawisku *overfitting* [11].

  Warstwa wyjściowa jako funkcji aktywacji używa funkcji *softmax*.
- Drugi wariant korzysta z pojedynczej warstwy konwolucyjnej z rozmiarem okna 8×8 oraz warstwy *dropout*. Wynik warstwy konwolucyjnej jest spłaszczany i dostarczany do warstwy wyjściowej z funkcją *softmax*.

2. Instrukcja użytkowania

W skład projektu wchodzą trzy główne skrypty Pythona, odpowiadające kolejno za trenowanie klasyfikatorów opartych na sieciach neuronowych i testowanie działania wszystkich trzech klasyfikatorów na obrazach statycznych oraz strumieniach wideo z kamery komputera.

2.1. Trenowanie sieci neuronowej

Do wykonania treningu sieci neuronowej należy wykorzystać skrypt `train.py`. Dokonuje on wyznaczenia modelu dla sieci w obu wariantach (spłaszczonym i konwolucyjnym). Skrypt należy wywołać poleceniem

```
$ python train.py -d train-image-directory
```

gdzie `train-image-directory` oznacza katalog zawierający zbiór treningowy udostępniony w [8]. Obrazy w zbiorze treningowym muszą zawierać czarne kształty na białym tle.

Wykonanie skryptu rozpocznie proces uczenia sieci, który może potrwać do kilkunastu minut. W wyniku działania skryptu w bieżącym katalogu zostanie stworzony podkatalog `model`, zawierający dwa pliki:

- Plik `shapes_model_1d_vec.h5` zawiera model dla sieci neuronowej operującej na spłaszczonym obrazie.
- Plik `shapes_model_2d_img.h5` zawiera model dla konwolucyjnej sieci neuronowej.

2.2. Testy na obrazach wczytywanych z pliku

Klasyfikacji dla folderu zawierającego obrazy można wykonać za pomocą skryptu `image_test.py`. Skrypt ten wczyta wszystkie obrazy (z białymi kształtami na czarnym tle) znajdujące się w danym folderze i dokona ich zbiorczej identyfikacji. Do uruchomienia go należy użyć polecenia

```
$ python image_test.py -d test-image-directory
                       -c {geometric,vector-network,convolutional-network}
```

- W parametrze `-d` podajemy ścieżkę do folderu, w którym znajdują się obrazy testowe.
- Parametr `-c` służy wyborowi klasyfikatora, który powinien być użyty dla danych testowych. Możliwe opcje to:
  - `geometric` — używa klasyfikatora geometrycznego,
  - `vector-network` — używa klasyfikatora opartego o płaską sieć neuronową,
  - `convolutional-network` — używa klasyfikatora opartego o sieć konwolucyjną.

Na wyjściu program wypisze liczbę rozpoznanych kształtów każdego rodzaju na wszystkich obrazach znajdujących się w wyspecyfikowanym katalogu.

2.3. Rozpoznawanie kształtów w czasie rzeczywistym

Ostatni skrypt, `capture.py`, służy demonstracji działania klasyfikatorów na strumieniu wideo. Obraz pobierany jest z kamerki komputera, jeśli taką posiada. Skrypt uruchamiamy bez parametrów poleceniem:

```
$ python capture.py
```

Skrypt ma charakter interaktywnego okna — za pomocą klawiszy klawiatury można przełączać się między klasyfikatorami lub dokonać próby czasowej. Dokładny opis wszystkich opcji znajduje się poniżej.

- Klawisze numeryczne służą przełączaniu się między klasyfikatorami, których wyniki wyświetlane są na podglądzie.
  - Klawisz [1] powoduje przełączenie podglądu na klasyfikator geometryczny.
  - Klawisz [2] powoduje przełączenie podglądu na wektorowy klasyfikator z siecią neuronową.
  - Klawisz [3] powoduje przełączenie podglądu na klasyfikator z konwolucyjną siecią neuronową.
- Za pomocą skryptu można wykonać próbę czasową wszystkich klasyfikatorów dla ustalonego rodzaju kształtu. Dla każdego klasyfikatora liczony jest czas poprawnej klasyfikacji. Czas zatrzymywany jest, jeśli klasyfikator przez dłużej niż 24 klatki (1 sekundę) nie rozpoznaje kształtu lub rozpoznaje go błędnie.
  - Klawisze [Z], [X], [C], [V] ustawiają docelowy rodzaj kształtu.
    - Wciśnięcie klawisza [Z] oznacza, że rozpoznawane będą kwadraty.
    - Wciśnięcie klawisza [X] oznacza, że rozpoznawane będą gwiazdy.
    - Wciśnięcie klawisza [C] oznacza, że rozpoznawane będą koła.
    - Wciśnięcie klawisza [V] oznacza, że rozpoznawane będą trójkąty.
  - Klawisz [A] powoduje rozpoczęcie liczenia czasu.
  - Klawisz [S] powoduje zatrzymanie rozpoczętej próby.
  - Klawisz [R] powoduje zrestartowanie próby.

  Podczas trwania próby można zarówno zmieniać docelowy typ rozpoznawanego kształtu, jak i przełączać podgląd wyników klasyfikatora między dostępnymi opcjami.
- Klawisz [H] powoduje wyświetlenie pomocy na ekranie.
- Klawisz [D] powoduje włączenie trybu *debug*. W tym trybie w osobnych oknach na ekranie wyświetlane są rozpoznawane po progowaniu kontury, a na konsoli wyświetlane są: średni, minimalny i maksymalny czas działania poszczególnych klasyfikatorów.
- Klawisz [Q] powoduje zakończenie działania skryptu.

3. Wyniki eksperymentalne

3.1. Kryteria oceny klasyfikatorów

3.2. Uzyskane wyniki

4. Podsumowanie

4.1. Możliwości dalszych badań

Literatura

[1] C.M. Bishop: *Neural Networks for Pattern Recognition*. Oxford University Press: Nowy Jork, 1995.

[2] F. Chollet, Keras Team: Keras – Deep Learning for humans.
Oficjalna strona: `https://keras.io` [Dostęp 17 marca 2018]

[3] A. Collette, HDF5 for Python.
Oficjalna strona: `https://www.h5py.org/` [Dostęp 27 maja 2018]

[4] D.H. Douglas, T.K. Peucker, „Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature", *Cartographica: The International Journal for Geographic Information and Geovisualization*, tom 10, s. 112–122, 1973.

[5] Intel Corporation, Willow Garage, Itseez: OpenCV – Open Source Computer Vision.
Oficjalna strona: `https://opencv.org`. [Dostęp 17 marca 2018]

[6] Matplotlib Development Team: Matplotlib.
Oficjalna strona: `https://matplotlib.org`. [Dostęp 17 marca 2018]

[7] W. McKinney: pandas – Python Data Analysis Library.
Oficjalna strona: `https://pandas.pydata.org`. [Dostęp 17 marca 2018]

[8] S. Meschke: Four Shapes dataset.
Dostępny: `https://www.kaggle.com/smeschke/four-shapes`. [Dostęp 17 marca 2018]

[9] M.N. Murty, V.S. Devi: *Introduction to Pattern Recognition and Machine Learning*. World Scientific Publishing: Singapur, 2015.

[10] T. Oliphant: NumPy.
Oficjalna strona: `http://www.numpy.org`. [Dostęp 17 marca 2018]

[11] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, „Dropout: A Simple Way to Prevent Neural Networks from Overfitting", *Journal of Machine Learning Research*, tom 15, s. 1929–1958, 2014.

[12] S. Suzuki, K. Abe, „Topological structural analysis of digitized binary images" *Computer Vision, Graphics and Image Processing*, tom 30, s. 32–66, 1985.

[13] TensorFlow Team: TensorFlow – An open source machine learning framework for everyone.
Oficjalna strona: `https://www.tensorflow.org/` [Dostęp 27 maja 2018]