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

Bartłomiej Dach Tymon Felski

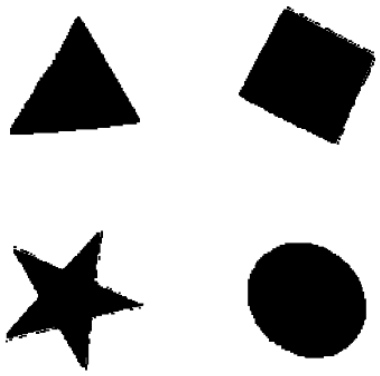
31 maja 2018

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.

Komponent programu – nazywany dalej ekstraktorem – znajdujący figury na obrazie i wycinający je z niego do późniejszej analizy przez klasyfikatory jest niezależnym modułem. Wykorzystanie go we wszystkich klasyfikatorach pozwala wykluć jego wpływ na jakość klasyfikacji w przypadku każdego z klasyfikatorów.

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*, w celu znormalizowania wartości wyjściowej do przedziału $[0, 1]$.

Głównymi dobieranymi parametrami podczas treningu parametrami była liczba epok oraz tzw. *batch size*. Eksperymenty, których wyniki widoczne są na wykresie 2, pokazują, że mają one niewielki wpływ na efektywność sieci. W związku z tym przyjęto wartości:

- 50 epok i *batch size* 256 — dla sieci w modelu płaskim,
- 25 epok i *batch size* 256 — dla sieci w modelu konwolucyjnym.

Oprócz weryfikacji wpływu parametrów na efektywność klasyfikatorów zbadany został również wpływ procesów stochastycznych będących częścią treningu, tj.:

- losowania początkowych wartości wag i *bias*,
- losowego działania warstw *dropout*,
- mieszania kolejności obrazów ze zbioru testowego.

W tym celu uruchomiono trening w obu modelach dziesięciokrotnie i zbadano wartości funkcji straty i dokładności klasyfikacji. Wyniki zaprezentowane zostały na wykresie 3. Patrząc na wygląd wykresów, można wywnioskować, że procesy niedeterministyczne mają mały wpływ na działanie klasyfikatorów i ich istnienie można w dalszych rozważaniach pominąć.

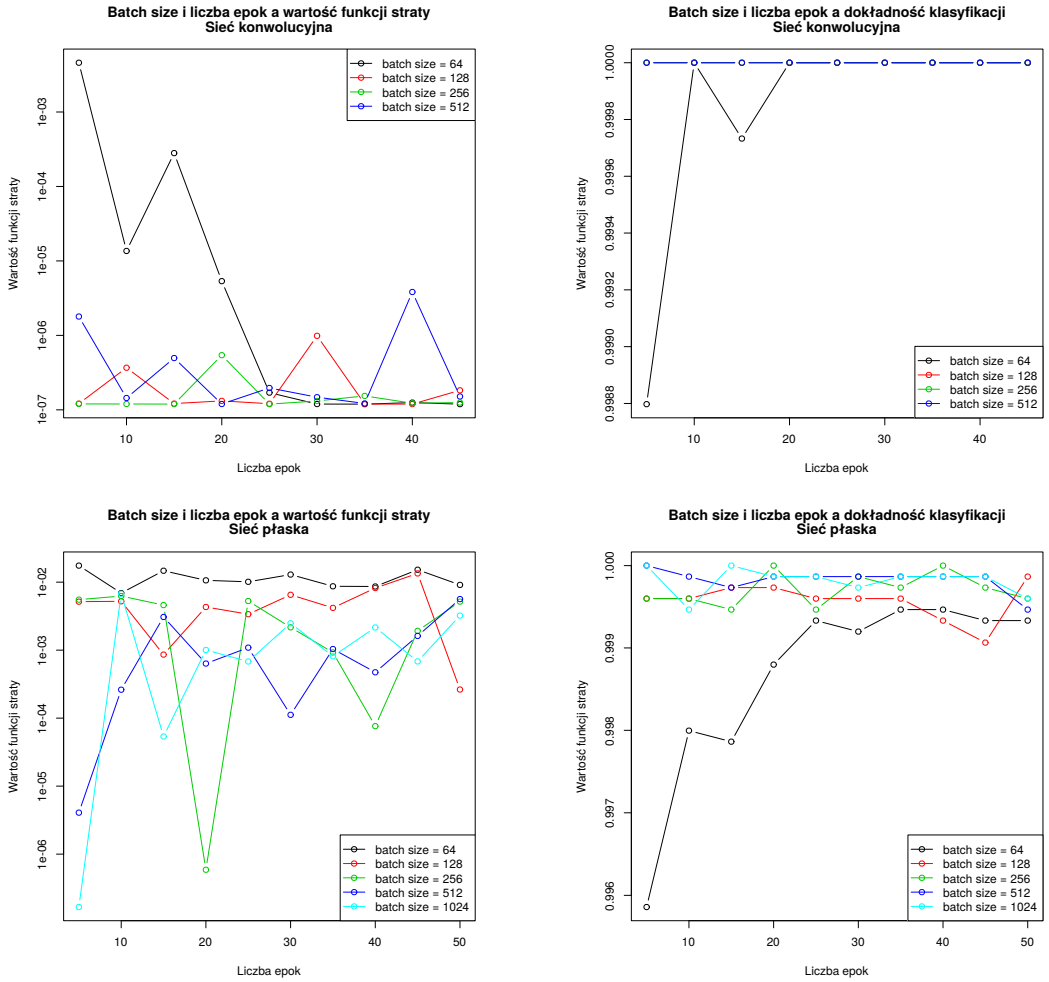
2. Instrukcja użytkowania

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.

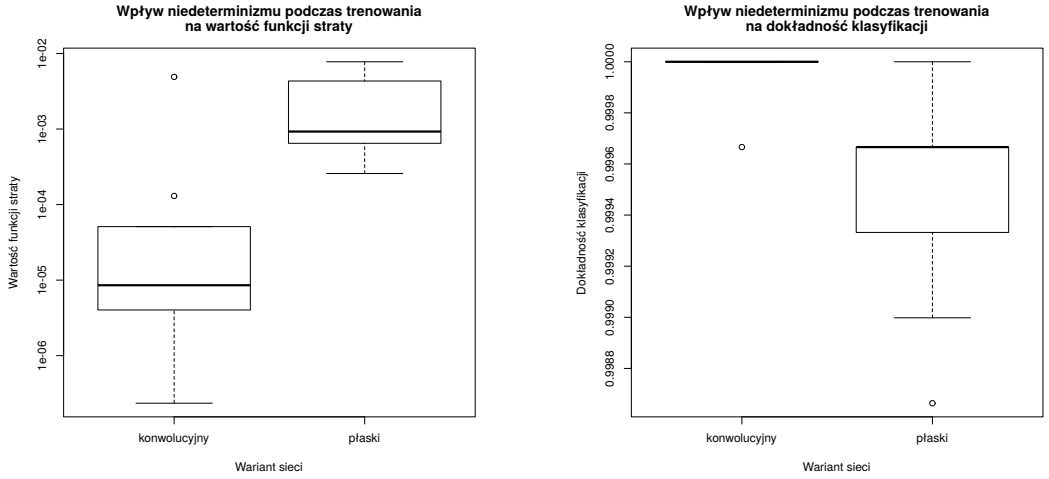
2.1. Trenowanie sieci neuronowej

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
```



Rysunek 2: Wykresy przedstawiające wpływ dobieranych parametrów (liczby epok i *batch size*) na wartość funkcji straty i dokładność klasyfikacji. Na wykresie przedstawiona wartość średnia z trzech osobnych uruchomień dla każdego zestawu parametrów. Niedeterminizm został zminimalizowany poprzez ustalenie ziarna generatora liczb pseudolosowych.



Rysunek 3: Wykresy pudełkowe przedstawiające wpływ niedeterminizmu w procesie treningu sieci neuronowych w obu modelach na wartości funkcji straty i dokładność klasyfikacji.

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 wytrenowany przy użyciu 50 epok.
- Plik `shapes_model_2d_img.h5` zawiera model dla konwolucyjnej sieci neuronowej wytrenowany przy użyciu 25 epok.

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 oraz liczba klatek, na której kształt został zidentyfikowany pomyślnie. Czas zatrzymywany jest, jeśli klasyfikator przez dłużej niż 8 klatek podejmował próbę rozpoznania kształtu i rozpoznawał inny kształt niż był ustawiony obecnie oczekiwany lub gdy stwierdził, że wykryty obszar nie zawiera żadnego ze znanych mu kształtów. Zliczanie poprawnych klatek w każdym klasyfikatorze nie zatrzymuje się wraz z zatrzymaniem jego czasomierza. Analogicznie, ogólny czasomierz zatrzyma się tylko, jeżeli zatrzymana zostanie cała rozpoczęta próba czasowa.

  - Klawisze [Z], [X], [C], [V] pozwalają zmienić rodzaj kształtu.
    * Wciśnięcie klawisza [Z] spowoduje, że oczekiwanym kształtem będzie kwadrat.
    * Wciśnięcie klawisza [X] spowoduje, że oczekiwanym kształtem będzie gwiazda.
    * Wciśnięcie klawisza [C] spowoduje, że oczekiwanym kształtem będzie koło.
    * Wciśnięcie klawisza [V] spowoduje, że oczekiwanym kształtem będzie trójkąt.
  - Klawisz [A] powoduje rozpoczęcie próby czasowej.
  - Klawisz [S] powoduje zatrzymanie rozpoczętej próby czasowej.
  - Klawisz [R] powoduje zrestartowanie próby czasowej.

  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 osobnym oknie 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

W celu oceny zaimplementowanych klasyfikatorów wykonane zostało kilka eksperymentów, których celem było zbadanie ich efektywności i wydajności. Niniejszy rozdział zawiera opis wykonanych testów oraz przedstawia uzyskane wyniki.

3.1. Kryteria oceny klasyfikatorów

Ocenie zdecydowano się poddać kilka kryteriów. Wyniki klasyfikacji postanowiono zbadać poprzez wyznaczenie dokładności *(ang. accuracy)*, która jest jedną z miar oceny jakości klasyfikacji. Wydajność klasyfikatorów zweryfikowano mierząc czas wywołań klasyfikacji dla każdego z nich. Ponadto klasyfikatory poddano próbie o charakterze empirycznym, polegającej na teście w czasie rzeczywistym.

3.1.1. Dokładność

Dokładność jest najbardziej intuicyjnym pojęciem pozwalającym na zmierzenie szeroko rozumianej skuteczności klasyfikatora. Dokładność jest bowiem stosunkiem liczby poprawnych klasyfikacji do ogólnej liczby podjętych prób.

Dla każdego obrazu ze zbioru [8] uruchomiono klasyfikację kształtu przy pomocy klasyfikatora geometrycznego i każdej z sieci neuronowych. Następnie zgodnie z definicją dokładności zliczono poprawne wyniki i wyznaczono szukany współczynnik.

W przypadku rozpatrywanych sieci neuronowych dokładność jest dodatkowo jedną z miar wypisywanych na ekran w trybie *verbose* podczas treningu. Podczas trenowania zbiór danych wejściowych dzielony jest na dwa zbiory — co piąty element dodawany jest do zbioru testowego. Zbiór treningowy zawiera wobec tego 80% obrazów, a po każdej epoce następuje walidacja obecnego stanu sieci przy pomocy pozostałych 20% obrazów. Na samym końcu treningu wykonywana jest ewaluacja wytrenowanego modelu, w celu ustalenia ostatecznej dokładności klasyfikacji na zbiorze testowym.

3.1.2. Wydajność

Stworzony na potrzeby projektu skrypt `capture.py` pozwala na włączenie trybu *debug*, dzięki któremu wyniki wydajnościowe każdego klasyfikatora są wypisywane na ekran. Podawane są wartości takie jak:

- minimalny czas klasyfikacji od początku działania skryptu,
- maksymalny czas klasyfikacji od początku działania skryptu,
- średni czas klasyfikacji od początku działania skryptu.

Zdecydowano się nie liczyć mediany, ponieważ wymagałoby to zapamiętania wszystkich czasów klasyfikacji, co wpłynęłoby negatywnie na zużycie pamięci.

3.1.3. Próba czasowa

W tym teście zbadano zachowanie wszystkich trzech klasyfikatorów jednocześnie w tym samym scenariuszu, obejmującym rozpoznawanie kształtów w czasie rzeczywistym na strumieniu wideo. Wykorzystano w tym celu opisywany wcześniej skrypt `capture.py`, przy pomocy którego ustawiono na rozpoznawanie konkretnego kształtu i uruchomiono próbę czasową. Zadaniem jest poprawna klasyfikacja pokazywanego kształtu przez jak najdłuższy okres czasu, przy czym dopuszcza się pomyłkę przez nie dłużej niż 8 klatek z rzędu. Zwrócenie błędnego wyniku przez dłużej niż ustalony limit powoduje zakończenie pomiaru czasu dla danego klasyfikatora. Dodatkowo podczas trwania próby dla każdego klasyfikatora zliczano klatki, na których doszło do poprawnej klasyfikacji w tym samym okresie czasu.

3.2. Uzyskane wyniki

W poniższym podrozdziale opisano wyniki uzyskane przy pomocy opisanych powyżej testów.

3.2.1. Dokładność

W zbiorze [8] znajduje się 14970 obrazów — 3720 kół, 3765 kwadratów, 3765 gwiazd i 3720 trójkątów. Dla każdego z klasyfikatorów wyznaczono dokładność uruchamiając proces klasyfikacji na każdym z obrazów przy pomocy skryptu `image_test.py` i zliczając stosunek poprawnych dopasowań do liczby wszystkich prób. Wyniki wyglądają następująco:

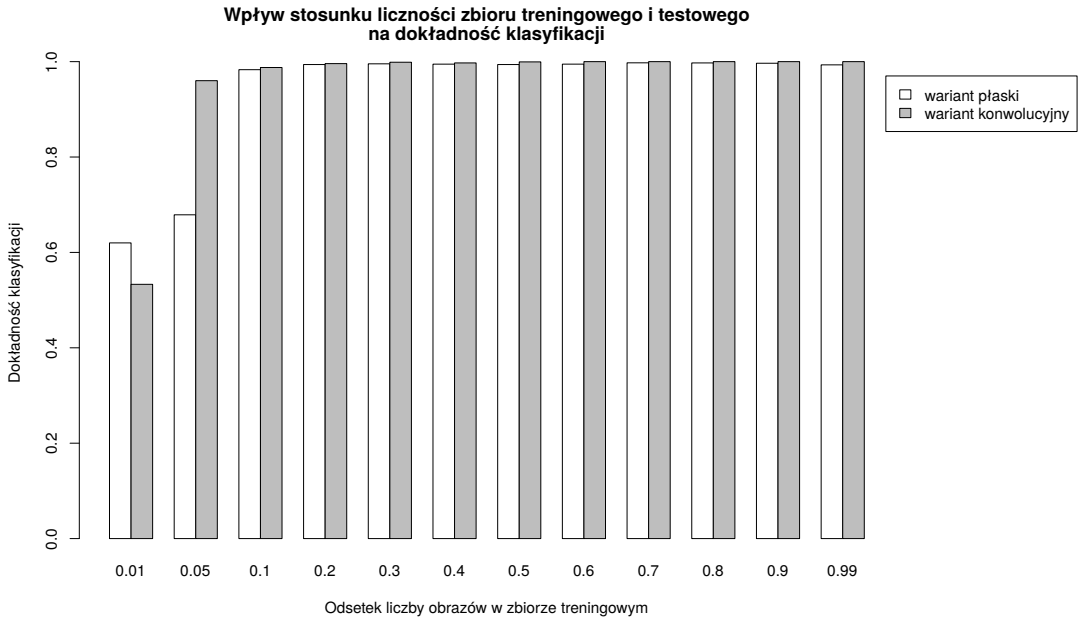
- Klasyfikator geometryczny zakwalifikował poprawnie 14911 kształtów, co daje dokładność na poziomie 0.9961 w przybliżeniu. W tym przypadku najgorzej wypadły trójkąty — były tam aż 153 pomyłki.
- Klasyfikator korzystający z sieci neuronowej klasyfikującej wektory zidentyfikował poprawnie 14813 kształtów, co daje dokładność na poziomie 0.9895. Najwięcej błędów zostało popełnionych w przypadku klasyfikacji gwiazd (46).
- Klasyfikator korzystający z konwolucyjnej sieci neuronowej rozpoznał poprawnie 14915 kształtów, co daje dokładność na poziomie 0.9963 w przybliżeniu. Podobnie jak w przypadku klasyfikatora geometrycznego, najgorzej wypadła klasyfikacja trójkątów (53 pomyłki).

Warto zauważyć, że poza siedmioma przypadkami błędnej identyfikacji trójkątów jako kwadraty i gwiazdy przez sieć konwolucyjną, nie wystąpiły żadne błędy pierwszego typu (ang. *false positive errors*). Wszystkie popełnione błędy dotyczyły nierozpoznania znanego kształtu.

Jak już wspomniano wcześniej, dokładność była także wypisywana na ekran w trakcie treningu sieci neuronowych. Wartość wyznaczona podczas ewaluacji wytrenowanego modelu w przypadku sieci klasyfikującej wektory to 0.9980, natomiast dla sieci konwolucyjnej zwrócona wartość była 1.0 (wyniosła tyle już po siedemnastej z 25 epok).

Dodatkowo przeprowadzono eksperyment mający na celu sprawdzić wpływ wielkości zbioru treningowego na dokładność klasyfikacji przy użyciu klasyfikatorów opartych o sieci neuronowe. Przetestowano wielkości: $[0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.99]$, gdzie każdy z ułamków oznacza jak duża część obrazów ze zbioru [8] zostanie użyta podczas treningu. Przykładowo, dla wielkości 0.8 będzie to 80% obrazów. Wyniki eksperymentu przedstawia wykres 4.

Łatwo zauważyć, że od wielkości zbioru treningowego wynoszącej około 150 zdjęć, wyniki są bardzo dobre i wszystkie na poziomie dokładności zbliżonym do 1.



Rysunek 4: Wykres przedstawiający wpływ stosunku liczności zbioru treningowego i testowego na dokładność klasyfikacji

3.2.2. Wydajność

Wydajność każdej z metod klasyfikacji zmierzono na dwa sposoby. Pierwszym źródłem wyników był wykorzystany do badania dokładności skrypt `image_test.py`, w którym zmierzono upływ czasu klasyfikacji wszystkich 14970 obrazów łącznie z ich obróbką przez ekstraktor. Otrzymane wyniki są następujące:

- Klasyfikator geometryczny skończył klasyfikację obrazów w 28.66s, co daje w przybliżeniu średnio 0.0019s na obróbkę i klasyfikację każdego kształtu.
- Klasyfikator korzystający z sieci neuronowej klasyfikującej wektory skończył klasyfikację obrazów w 43.67s, co daje w przybliżeniu średnio 0.0029s na obróbkę i klasyfikację każdego kształtu.
- Klasyfikator korzystający z konwolucyjnej sieci neuronowej skończył klasyfikację obrazów w 48.53s, co daje w przybliżeniu średnio 0.0032s na obróbkę i klasyfikację każdego kształtu.

Drugim źródłem wyników był działający w trybie *debug* skrypt `capture.py`. Dla każdego z klasyfikatorów otrzymano następujące wyniki klasyfikacji pojedynczego kształtu (bez wliczonej obróbki wstępnej):

- Klasyfikator geometryczny:

  - najkrótszy czas klasyfikacji wyniósł 0.000018s,
  - najdłuższy czas klasyfikacji wyniósł 0.001961s,
  - średni czas klasyfikacji wyniósł 0.000086s.
- Klasyfikator korzystający z sieci neuronowej klasyfikującej wektory:
  - najkrótszy czas klasyfikacji wyniósł 0.000981s,
  - najdłuższy czas klasyfikacji wyniósł 0.022804s,
  - średni czas klasyfikacji wyniósł 0.001918s.
- Klasyfikator korzystający z konwolucyjnej sieci neuronowej:
  - najkrótszy czas klasyfikacji wyniósł 0.001086s,
  - najdłuższy czas klasyfikacji wyniósł 0.035751s,
  - średni czas klasyfikacji wyniósł 0.002212s.

3.2.3. Próba czasowa

Testom poddano każdy z kształtów osobno przy dodatkowym zastosowaniu różnych utrudnień, którymi są:

- translacja (przesuwanie) kształtu wzdłuż płaszczyzny kamery,
- obrót kształtu,
- przechylenie kształtu względem płaszczyzny kamery,
- przybliżanie i oddalanie kształtu.

Wyniki wykonanych prób czasowych znajdują się w tabeli 2. Kolumny oznaczone jako *geometric*, *network (vec)* oraz *network (conv)* odpowiadają odpowiednio klasyfikatorom: geometrycznemu, opartemu na sieci neuronowej klasyfikującej wektory oraz sieci konwolucyjnej.

Test	Kształt	geometric		network (vec)		network (conv)	
Translacja	kwadrat	00:14.368	(0.960)	00:14.968	(1.000)	00:14.968	(1.000)
	koło	00:16.493	(1.000)	00:16.493	(1.000)	00:16.493	(1.000)
	trójkąt	00:13.210	(1.000)	00:13.210	(1.000)	00:13.210	(1.000)
	gwiazda	00:14.480	(1.000)	00:14.480	(1.000)	00:14.480	(1.000)
Obrót	kwadrat	00:17.291	(1.000)	00:17.291	(1.000)	00:17.291	(1.000)
	koło	00:17.936	(1.000)	00:17.936	(1.000)	00:17.936	(1.000)
	trójkąt	00:18.347	(1.000)	00:18.347	(1.000)	00:18.347	(1.000)
	gwiazda	00:10.154	(0.652)	00:10.231	(0.657)	00:15.575	(1.000)
Przechylenie	kwadrat	00:03.113	(0.495)	00:06.254	(0.995)	00:06.285	(1.000)
	koło	00:05.658	(0.983)	00:05.568	(0.968)	00:05.753	(1.000)
	trójkąt	00:14.188	(1.000)	00:14.188	(1.000)	00:04.518	(0.318)
	gwiazda	00:08.886	(0.651)	00:03.590	(0.263)	00:13.652	(1.000)
Przybliżenie	kwadrat	00:17.115	(1.000)	00:17.115	(1.000)	00:17.115	(1.000)
	koło	00:08.984	(0.982)	00:09.150	(1.000)	00:08.921	(0.975)
	trójkąt	00:04.483	(0.289)	00:15.498	(1.000)	00:15.498	(1.000)
	gwiazda	00:04.558	(0.319)	00:04.590	(0.321)	00:14.308	(1.000)

Tablica 2: Wyniki przeprowadzonych prób czasowych. W nawiasach stosunek do najlepszego wyniku w obrębie danego wiersza.

Dla każdego z uzyskanych czasów obliczony został jego stosunek do najlepszego wyniku uzyskanego przez klasyfikatory w danej próbie. W tabeli 3 widoczny jest średni stosunek do najlepszego wyniku z wszystkich prób dla poszczególnych klasyfikatorów.

Klasyfikator	Średni stosunek
geometric	0.833
network (vec)	0.888
network (conv)	0.956

Tablica 3: Uśrednione wyniki względne z prób przedstawionych w tabeli 2

W teście empirycznym najlepiej zaprezentowała się sieć konwolucyjna, ze względu na największą stabilność działania.

4. Podsumowanie

Przeprowadzone eksperymenty pokazują, że wszystkie zastosowane podejścia dość dobrze radzą sobie z problemem rozpoznawania prostych kształtów. Zauważalne są jednak między nimi różnice w dziedzinie wydajności i efektywności. Mimo tego do klasyfikacji w każdym przypadku dochodzi na tyle szybko, że przeprowadzanie jej dla wszystkich trzech klasyfikatorów jednocześnie w czasie rzeczywistym nie stanowi problemu.

Klasyfikator oparty na konturach wykazuje w eksperymentach najkrótszy czas działania, lecz często działa dość niestabilnie, co wykazała próba czasowa. Z kolei sieci neuronowe mimo konieczności treningu na wcześniej przygotowanym zbiorze oraz dłuższego czasu klasyfikacji, nadal mogą być używane do rozpoznawania w czasie rzeczywistym, i działają mniej chaotycznie, niż podejście typowo geometryczne.

4.1. Możliwości dalszych badań

Istnieje kilka potencjalnych zagadnień, stanowiących naturalną kontynuację opracowanego projektu:

- Naturalnym rozszerzeniem istniejącego rozwiązania byłoby dodanie większej liczby rodzajów rozpoznawanych kształtów. Koszt takiego rozszerzenia różniłby się znacząco pomiędzy klasyfikatorami — podczas gdy rozszerzenie klasyfikatora geometrycznego byłoby stosunkowo bezproblemowe, dla pozostałych dwóch należałoby przygotować zbiór treningowy oraz ponownie wytrenować model sieci.
- Interesującym aspektem analizy stworzonych rozwiązań byłaby również ocena ich podatności na błędy pierwszego rodzaju (ang. *false positive*). Znane są rozwiązania używające np. sieci adwersaryjnych (ang. *adversarial network*) w celu wygenerowania obrazów powodujących błędne działanie klasyfikatorów opartych na sieciach neuronowych.
- Pewną wadą modeli dopasowanych w ramach projektów jest nieuwzględnienie w zbiorze treningowych obiektów niepasujących do żadnej z kategorii kształtów. Wobec tego pomysłem wartym rozważenia byłoby dodanie dodatkowej zbiorczej klasy na wszystkie pozostałe kształty niepasujące do ustalonych archetypów.

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]