Metody Inteligencji Obliczeniowej w Analizie Danych 2021

SPRAWOZDANIE Z PROCESU IMPLEMENTACJI SIECI KOHONENA

26 kwietnia 2021

Elżbieta Jowik, 298821

Cele

W ramach drugiej części laboratorium należało zaimplementować sieć Kohonena, a następnie zastosować ją do dwóch zbiorów obrazków: *MNIST* i *Human Activity Recognition Using Smartphones* (bez etykiet).

Krótkie wprowadzenie teoretyczne

Sieci Kohonena są jednym z podstawowych typów sieci samoorganizujących się i jednocześnie szczególnym przypadkiem algorytmu realizującego ideę uczenia się bez nadzoru. Trenowanie tego wariantu odbywa się na drodze tzw. samouczenia, gdzie dla podawanych danych wejściowych nie są przedstawiane żadne wzorce wyjścia i są one samodzielnie klasyfikowane jedynie na podstawie występujących między nimi wewnętrznych zależności. Cechą charakterystyczną sieci Kohonena, wyróżniającą tę grupę od innych wariantów sieci typu 'unsupervised', jest uczenie się **metodą samoorganizującą typu konkurencyjnego**, polegającą na podawaniu na wejściu sieci wektorów stanowiących próbę uczącą a następnie sekwencyjnym wybieraniu w drodze konkurencji zwycięskich neuronów, które najlepiej odpowiadają kolejnym sygnałom wejściowym. Założeniem działania sieci jest organizacja wielowymiarowej informacji w przestrzeni o znacznie mniejszej liczbie wymiarów, przy czym topologia sieci Kohonena ma odpowiadać topologii docelowej przestrzeni. Konsekwencją otrzymanych mapowań powinno być umiejscowienie obok siebie neuronów reprezentujących podobne klasy i utworzenie w ten sposób uporządkowanej mapy, na podstawie której możliwe powinno być określenie pewnych międzyklasowych relacji.

Etap 1.

Opis wykonanych prac

W ramach pierwszego etapu przeprowadzonych rozważań zaimplementowano sieć Kohonena złożoną z neuronów w prostokątnej siatce parametryzowaną przez:

- $\mapsto$ wymiary tej siatki: $M,\ N$,
- $\mapsto$ szerokość sąsiedztwa: $neigh\_width$,
- $\mapsto$ wymiar danych z próby uczącej: $data\_dim$,
- $\mapsto$ metrykę w przestrzeni danych: $dist\_class$.

Przygotowana implementacja działała dla zbioru wektorów o tej samej długości i dwóch funkcji sąsiedztwa: funkcji gaussowskiej i minus drugiej pochodnej funkcji gaussowskiej (tzw. meksykański kapelusz). Za funkcję wygaszającą (*ang. decay function*) przyjęto:

$$\alpha(t, \lambda) = e^{t/\lambda},$$

gdzie za t brano numer bieżącej epoki a za λ podaną przez użytkownika liczbę wszystkich epok.

Zwięzły schemat działania przygotowanej implementacji, która miała realizować funkcjonowanie sieci samoorganizującej:

1. Na poziomie funcji inicjalizującej obiekty klasy *KohonenNetwork* tworzona jest macierz neuronów pobudzanych przez sygnały wejściowe i przeprowadzona losowa inicjalizacja wag z rozkładu standardowego normalnego.

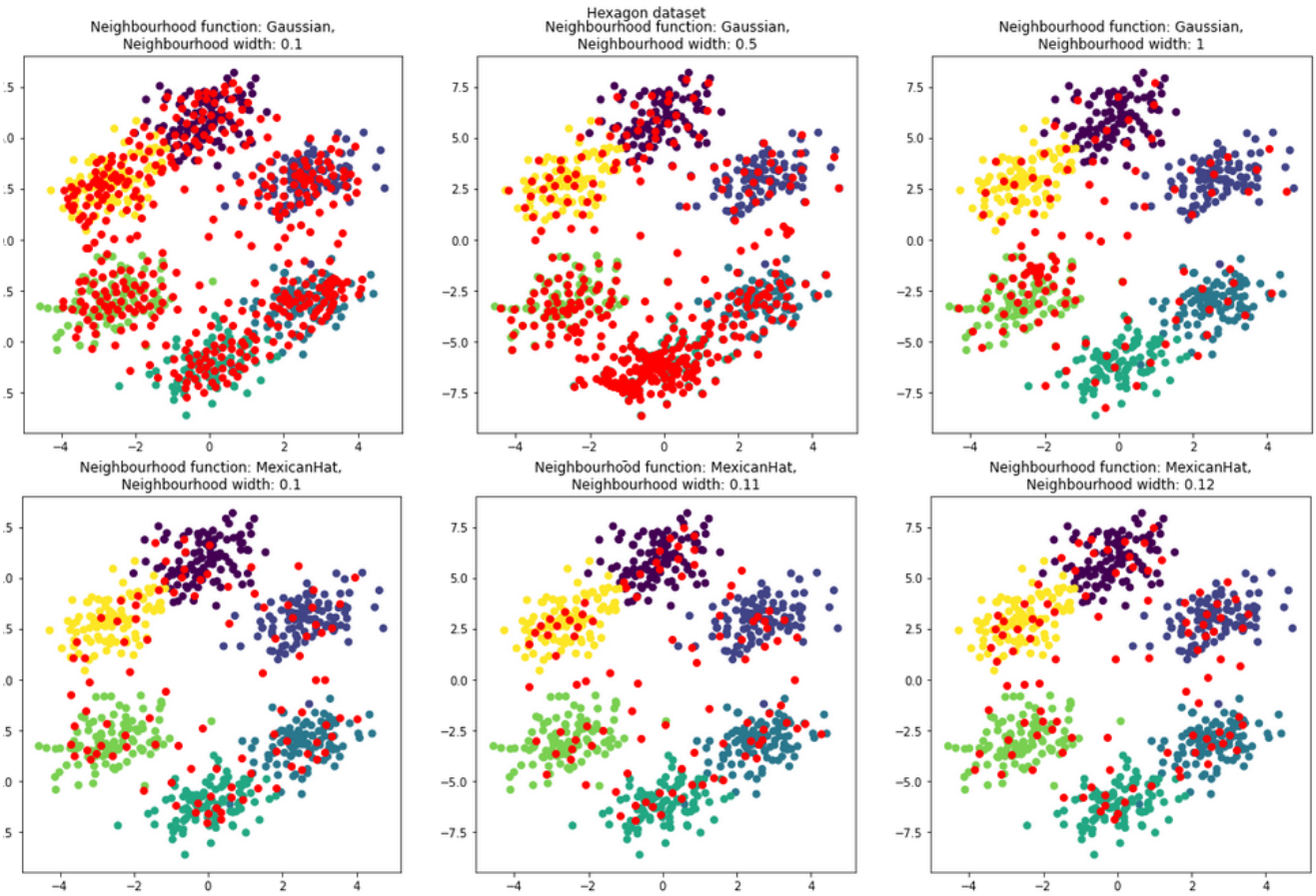
2. Neurony pierwszej warstwy przekazują dane wprowadzone na wejścia sieci do warstwy konkurencyjnej nie dokonując przy tym żadnych przekształceń.

3. Funkcja *detect_closest_neuron()* dla każdego neuronu drugiej warstwy określa stopień podobieństwa jego wag do danego sygnału wejściowego oraz wyznacza ten z największym dopasowaniem względem zadanej metryki.

4. W procesie uczenia, za przeprowadzenie, którego odpowiada funkcja *train()*, wagi neuronu wyznaczonego w poprzednim kroku oraz neuronów znajdujących się w jego sąsiedztwie są modyfikowane w ramach wywołania funkcji *weights_update()* w taki sposób, aby najlepiej odzwierciedlać wewnętrzną strukturę podanego właśnie wektora uczącego.
   Wspomniane sąsiedztwo parametryzowane wartością *neigh_width* rozumiane jest w sensie geometrycznym jako położenie neuronu względem zwycięzcy.

Działanie sieci przetestowano na dostarczonych prostych zbiorach danych: danych 2d skupionych w wierzchołkach sześciokąta, danych 3d skupionych w wierzchołkach sześcianu.
We wszystkich eksperymentach za metrykę w przestrzeni danych przyjęto odległość euklidesową a parametrami szerokości sąsiedztwa w kolejnych próbach były wartości z przedziału $[0.1, 1]$.

Wyniki eksperymentów przeprowadzonych dla dwuwymiarowych danych



Rysunek 1: Klastry znalezione w procesie uczenia (czerwone punkty) nałożone na oryginalny podział danych uczących dla różnych funkcji i szerokości sąsiedztwa

Powyższe rezultaty uzyskano dla 10 epok. Dla gaussowskiej funkcji sąsiedztwa (pierwszy wiersz wizualizacji) zastosowano siatkę 20×20, w drugim przypadku (drugi wiersz): 10×10.

Wnioski na temat mapowań otrzymanych z użyciem sieci Kohonena

↦ Dla zbioru danych 2d skupionych w wierzchołkach sześciokąta

- Klastry w odwzorowaniu znalezionym przez sieć z gaussowską funkcją sąsiedztwa pokrywają się w liczbą klastrów w faktycznych danych już dla zaledwie 10 iteracji przy czym dokładność mapowania wyraźnie zależy od wartości parametru szerokości sąsiedztwa - dla ograniczonego sąsiedztwa rezultaty wydają się wyraźnie lepsze niż wówczas, gdy wagi modyfikowane są dla pełnej siatki.
- Klastry w odwzorowaniu znalezionym przez sieć z funkcją sąsiedztwa *MexicanHat* również pokrywają się w liczbą klastrów w faktycznych danych, ale wyłącznie dla małych wartości parametru *neigh_width* - dla większych (już od 0.13) wagi rosną do nieskończoności. W przeciwieństwie do poprzedniego przypadku wpływ wartości parametru sąsiedztwa na jakość mapowania wydaje się być znikomy, ale w mojej ocenie wynika to z nieznacznych różnic wartości tego atrybutu przyjętych w poszczególnych eksperymentach (różnice rzędu $0.01 - 0.02$)

↦ Dla zbioru danych 3d skupionych w wierzchołkach sześcianu.

- Sieć z gaussowską funkcją sąsiedztwa radzi sobie tak jak w przypadku zbioru dwuwymiarowego już dla 10 iteracji, ale trudno rozstrzygnąć czy klastry w znalezionym odwzorowaniu pokrywają się w liczbą klastrów w faktycznych danych. W kontekście wpływu wartości parametru sąsiedztwa na jakość mapowania można zauważyć, że wraz ze zmniejszaniem szerokości sąsiedztwa rośnie koncentracja neuronów w siatce, ale trudno jednoznacznie rozstrzygnąć o jego wpływie na stopień odseparowania klastrów.
- Przy zastosowaniu funkcji MexicanHat również trudno rozstrzygnąć czy klastry w znalezionym odwzorowaniu pokrywają się w liczbą klastrów w faktycznych danych. Ponadto wraz ze wzrostem szerokości sąsiedztwa klastry obecne w danych wejściowych wydają się być odwzorowywane coraz gorzej, przy czym wniosek ten nie może być traktowany jako wiążący (ponownie) ze względu na nieistotne różnice wartości tego atrybutu przyjętych w poszczególnych eksperymentach (różnice rzędu 0.05).

Wnioski końcowe

↦ Liczba iteracji nie wpływa znacząco na jakość odwzorowania znajdowanego przez sieć z gaussowską funkcją sąsiedztwa.

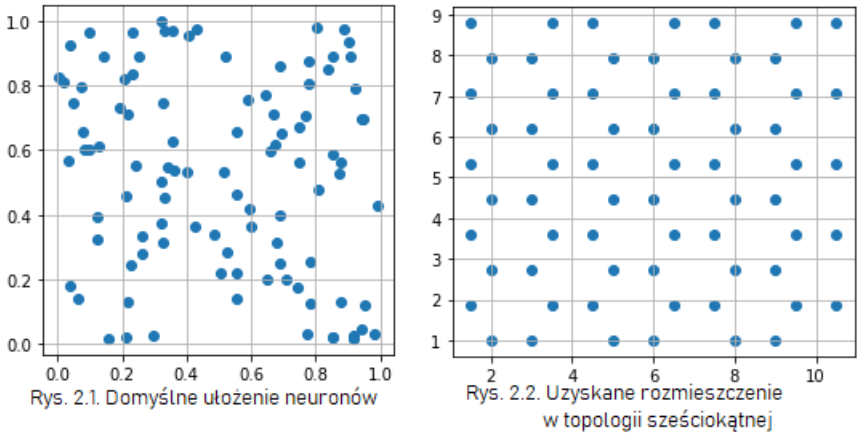
↦ Funkcja *MexicanHat* jest podatna na problem przepełnienia (*ang. overflow*) w związku z czym dobór parametrów uczenia i wymiarów siatki może okazać się newralgiczny w przypadku uczenia sieci z tą funkcją sąsiedztwa.

↦ Sieć z funkcją MexicanHat sprawdza się przy niewielkiej liczbie iteracji, ale tylko dla niewielkich wartości parametru *neigh_width*.

Etap 2.

Opis wykonanych prac

Założeniem drugiego etapu ewaluacji było rozszerzenie istniejącej implementacji o możliwość ułożenia neuronów w topologii siatki sześciokątnej. Jego realizacja wymusiła modyfikację funkcji inicjalizującej sieć oraz stworzenie dwóch dodatkowych funkcji: *assign_class()* i *predict()* przypisujące odpowiednio neurony i obserwacje ze zbioru testowego do klas postrzeganych jako najbliższe. Wspomniane modyfikacje funkcji *__init()__* obejmowały nałożenie na współrzędne neuronów ograniczeń (warunków logicznych) pozwalających na wymuszenie sześciokątnej topologii ich rozmieszczenia w siatce. Szczegóły wraz z komentarzami zostały umieszczone w dołączonej do sprawozdania implementacji. Działanie obydwu nowoutworzonych funkcji opiera się o podział neuronów na klastry znaleziony w procesie uczenia sieci. U podstaw działania wymienionej już funkcji *assign_class()* leżała obserwacja, że konsekwencją procesu uczenia sieci jest mapowanie na dany neuron wielu przykładów uczących, spośród których każdy ma przypisaną jakąś klasę. Przyjęto podejście, w którym aby przypisać klasę do danego neuronu wystarczyło, rozstrzygnąć obserwacji, której klasy było dla niego najwięcej i uznać, że ten neuron odpowiada tej klasie (ta klasa jest mu najbliższa). Z kolei wdrożenie funkcji klasyfikującej obserwacje ze zbioru testowego (*predict()*) oparte było na założeniu, że jeśli na podstawie części treningowej do neuronów przypisane zostaną klasy, to aby dokonać predykcji dla nowych obserwacji wystarczy sprawdzić, do którego neuronu z punktu jest najbliżej (względem przyjętej metryki) i nadać punktowi jego klasę.



Rysunek 2: Porównanie rozmieszczeń neuronów w siatce w istniejącej implementacji i uzyskanej w bieżącym etapie topologii sześciokątnej

W ramach zadanych eksperymetów należało zastosować obydwa warianty topologii i obie funkcje sąsiedztwa do wektorów danych ze zbiorów **MNIST** i **Human Activity Recognition Using Smartphones** a następnie przeanalizować otrzymane mapowanie danych uwzględniając ich etykiety.
Dla zbioru danych *MNIST* (załadowanych z openML) dla obydwu funkcji sąsiedztwa zastosowano te same liczby epok i szerokości sąsiedztwa równe odpowiednio 20 i 0.1.
Z kolei dla *Human Activity Recognition Using Smartphones* wymienione parametry wynosiły 10 i 0.1, przy czym aby sens miało rozważanie zamieszczonych w tabeli 2. statystyk, dla każdej funkcji sąsiedztwa proces uczenia powtórzono dziesięciokrotnie.

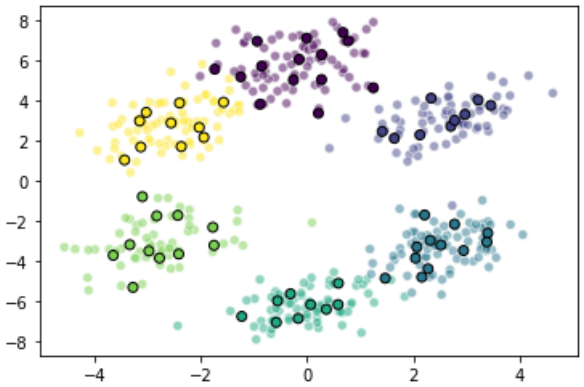
Wyniki eksperymentów

Tabela 1: Wartości miary *Accuracy* uzyskane dla predykcji sieci parametryzowanych przez poszczególne kombinacje topologii i funkcji sąsiedztwa (na przykładzie zbioru MNIST).

Zbiór MNIST			
Funkcja sąsiedztwa	**Topologia**	**Wymiary siatki**	**Accuracy**
Gaussowska	Sześciokątna	10×10	0.7516
Gaussowska	Domyślna	10×10	0.8814
Meksykański kapelusz	Sześciokątna	5×5	0.4641
Meksykański kapelusz	Domyślna	5×5	0.4748

Tabela 2: Podstawowe statystyki wyznaczone dla wartości miar względem, których oceniano otrzymane predykcje sieci na zbiorze *Human Activity Recognition Using Smartphones*

Human Activity Recognition Using Smartphones				
Funkcja sąsiedztwa	**Minimalne *Accuracy***	**Średnie *Accuracy***	**Maksymalne *Accuracy***	**Odchylenie std. *Accuracy***
Gaussowska	0.7788	0.81778	0.8582	0.025485
Meksykański kapelusz	0.5840	0.63851	0.7241	0.041311



Rysunek 3: Wizualizacja ukazująca jak dobrze znalezione klastry odpowiadały podziałowi na klasy (na przykładzie zbioru *Hexagon*)

Wnioski na temat mapowań otrzymanych z użyciem sieci Kohonena

↦ Dla zbioru *Hexagon* znalezione klastry odpowiadają rzeczywistemu podziałowi na klasy

↦ Wyniki czasowe i wydajnościowe dla obydwu topologii są zbliżone, ale tylko w przypadku prostych zbiorów. Wraz ze zwiększeniem stopnia skomplikowania danych uczących można zaobserwować, że czas uczenia sieci o topologii sześciokątnej dla tych samych wymiarów

siatki jest krótszy, ale jednocześnie wyniki są gorsze. Należy mieć na uwadze, że dla tych samych wymiarów siatki liczby neuronów w topologii sześciokątnej może być znacznie mniejsza niż w zwykłej.

↦ Wbrew wynikom umieszczonym w tabeli 1. nie jestem w stanie jednoznacznie określić, która funkcja aktywacji sprawdza się lepiej w przypadku danych z *MNIST*. Umieszczone w sprawozdaniu wyniki nie mogą być bowiem uznane za wiążące, ze względu na istotne różnice co do liczb neuronów w poszczególnych eksperymentach. Ze względu na ograniczoną moc obliczeniową i wolumen zbioru (rozważano całość zbioru w podziale na próbkę uczącą i testową w proporcjach 70 : 30) nie było bowiem możliwości przeprowadzenia bardziej miarodajnych testów.

↦ Zaobserwowano wydłużony czas obliczeń dla funkcji *MexicanHat* względem czasu dla funkcji gaussowskiej przy jednakowej parametryzacji.

↦ Dla zbioru *Human Activity Recognition Using Smartphones* zaobserwowane istotne różnice między wartościami podstawowych statystyk dla poszczególnych funkcji sąsiedztwa przy jednakowych wartościach pozostałych parametrów sieci.

Wnioski końcowe

↦ Dla prostych zbiorów topologia sześciokątna jest wystarczająca, dla bardziej złożonych gaussian sprawdza się lepiej pod względem jakości predykcji, przy czym jest znaczniej bardziej wymagająca pod względami czasowym/wydajnościowym.

↦ W porównywaniu ze sobą skuteczności poszczególnych topologii można zastosować dwa podejścia. W porównywanych siatkach można ustalić albo ich wymiary M, N albo liczbę neuronów. Na drodze przeprowadzonych eksperymentów stwierdzam, że drugi wariant tzn. porówywanie siatek, które mają zbliżoną lub tę samą liczbę neuronów, ale inaczej rozmieszczone wydaje się bardziej sensowny. U podstaw tego stwierdzenia leży obserwacja, że dla stworzonej implementacji ustalenie wymiarów siatki na 10×10 skutkowało otrzymaniem 100 neuronów dla standardowej topologii i 65 dla heksagonalnej.

↦ Poza wartością miar warto, o ile to możliwe, zwizualizować otrzymane mapowanie, ponieważ wartość metryki może okazać się myląca (w szczególności w przypadku danych niezbalansowanych).

↦ Dla prostego zbioru poza pojedynczymi przypadkami sieć Kohonena okazuje się być raczej stabilna ze względu na parametryzację (patrz suplement). Aby móc uogólnić ten wniosek bądź uznać go za jednoznacznie wiążący należałoby przeprowadzić analogiczny eksperyment dla zbioru o większym stopniu skomplikowania - z mniej jednoznaczną klasyfikacją obserwacji.

Suplement

Poza wymaganiami danymi w treściach zadań dla poszczególnych etapów przeanalizowano wpływ parametryzacji na dokładność precykcji ocenianą względem miary Accuracy na przykładzie zbioru Hexagon. Badanie to odbyło się poprzez wielokrotne uczenie sieci dla zadanej siatki hiperparametrów (została ona zdefiniowana w dołączonym do sprawozdania pliku *.ipynb), wykonanie predykcji na zbiorze testowym, którego wolumen stanowił 30% całości zbioru i porównanie otrzymanego mapowania danych z ich rzeczywistymi etykietami.

Tabela 1. Eksperyment przeprowadzony dla gaussowskiej funkcji sąsiedztwa

M	N	Topology	Neighbourhood width	Accuracy score
5	4	hexagon	0.1	0.9833
5	4	hexagon	0.2	0.9611
5	4	hexagon	0.3	0.9111
5	4	hexagon	0.4	0.9667
5	6	hexagon	0.1	0.9889
5	6	hexagon	0.2	0.9722
5	6	hexagon	0.3	0.9833
5	6	hexagon	0.4	0.9889
5	8	hexagon	0.1	0.9944
5	8	hexagon	0.2	0.9778
5	8	hexagon	0.3	0.9889
5	8	hexagon	0.4	0.9889
5	10	hexagon	0.1	0.9667
5	10	hexagon	0.2	0.9944
5	10	hexagon	0.3	0.9833
5	10	hexagon	0.4	0.9889
10	4	hexagon	0.1	0.9556
10	4	hexagon	0.2	0.9889
10	4	hexagon	0.3	0.9889
10	4	hexagon	0.4	0.9722
10	6	hexagon	0.1	0.9778
10	6	hexagon	0.2	0.9889
10	6	hexagon	0.3	0.9722
10	6	hexagon	0.4	0.9889
10	8	hexagon	0.1	0.9778
10	8	hexagon	0.2	0.9889
10	8	hexagon	0.3	0.9778
10	8	hexagon	0.4	0.9778
10	10	hexagon	0.1	0.9556
10	10	hexagon	0.2	0.9611
10	10	hexagon	0.3	0.9778
10	10	hexagon	0.4	0.9889
15	4	hexagon	0.1	0.9889
15	4	hexagon	0.2	0.9778
15	4	hexagon	0.3	0.9778
15	4	hexagon	0.4	0.9778

15	6	hexagon	0.1	0.9778
15	6	hexagon	0.2	0.9833
15	6	hexagon	0.3	0.9778
15	6	hexagon	0.4	0.9778
15	8	hexagon	0.1	0.9833
15	8	hexagon	0.2	0.9778
15	8	hexagon	0.3	0.9778
15	8	hexagon	0.4	0.9889
15	10	hexagon	0.1	0.9778
15	10	hexagon	0.2	0.9833
15	10	hexagon	0.3	0.9667
15	10	hexagon	0.4	0.9778
20	4	hexagon	0.1	0.9778
20	4	hexagon	0.2	0.9722
20	4	hexagon	0.3	0.9722
20	4	hexagon	0.4	0.9778
20	6	hexagon	0.1	0.9667
20	6	hexagon	0.2	0.9722
20	6	hexagon	0.3	0.9722
20	6	hexagon	0.4	0.9778
20	8	hexagon	0.1	0.9833
20	8	hexagon	0.2	0.9778
20	8	hexagon	0.3	0.9778
20	8	hexagon	0.4	0.9944
20	10	hexagon	0.1	0.9389
20	10	hexagon	0.2	0.9444
20	10	hexagon	0.3	0.8889
20	10	hexagon	0.4	0.9667
5	4	default	0.1	0.9556
5	4	default	0.2	0.9778
5	4	default	0.3	0.9944
5	4	default	0.4	0.9889
5	6	default	0.1	0.9778
5	6	default	0.2	0.9889
5	6	default	0.3	0.9833
5	6	default	0.4	0.9833
5	8	default	0.1	0.9722
5	8	default	0.2	0.9833
5	8	default	0.3	0.9889
5	8	default	0.4	0.9667
5	10	default	0.1	0.8778
5	10	default	0.2	0.9611
5	10	default	0.3	0.9444
5	10	default	0.4	0.9611
10	4	default	0.1	0.9833
10	4	default	0.2	0.9833
10	4	default	0.3	0.9722

10	4	default	0.4	0.9778
10	6	default	0.1	0.9389
10	6	default	0.2	0.9778
10	6	default	0.3	0.9611
10	6	default	0.4	0.9722
10	8	default	0.1	0.9
10	8	default	0.2	0.9778
10	8	default	0.3	0.9833
10	8	default	0.4	0.9778
10	10	default	0.1	0.8444
10	10	default	0.2	0.9556
10	10	default	0.3	0.9889
10	10	default	0.4	0.9778
15	4	default	0.1	0.9778
15	4	default	0.2	0.9778
15	4	default	0.3	0.9778
15	4	default	0.4	0.9833
15	6	default	0.1	0.9333
15	6	default	0.2	0.9222
15	6	default	0.3	0.9833
15	6	default	0.4	0.9611
15	8	default	0.1	0.9056
15	8	default	0.2	0.9833
15	8	default	0.3	0.95
15	8	default	0.4	0.9889
15	10	default	0.1	0.8889
15	10	default	0.2	0.9333
15	10	default	0.3	0.9667
15	10	default	0.4	0.9778
20	4	default	0.1	0.9333
20	4	default	0.2	0.9722
20	4	default	0.3	0.9833
20	4	default	0.4	0.9889
20	6	default	0.1	0.8944
20	6	default	0.2	0.9222
20	6	default	0.3	0.9833
20	6	default	0.4	0.9833
20	8	default	0.1	0.8889
20	8	default	0.2	0.9333
20	8	default	0.3	0.9722
20	8	default	0.4	0.9722
20	10	default	0.1	0.8556
20	10	default	0.2	0.9333
20	10	default	0.3	0.9778
20	10	default	0.4	0.9722

Tabela 2. Eksperyment przeprowadzony dla funkcji sąsiedztwa Mexican Hat

M	N	Topology	Neighbourhood width	Accuracy score
5	4	hexagon	0.1	0.9611
5	4	hexagon	0.2	0.9333
5	4	hexagon	0.3	0.9944
5	4	hexagon	0.4	0.9833
5	6	hexagon	0.1	0.9722
5	6	hexagon	0.2	0.9833
5	6	hexagon	0.3	0.9889
5	6	hexagon	0.4	0.9667
5	8	hexagon	0.1	0.9944
5	8	hexagon	0.2	0.9778
5	8	hexagon	0.3	0.9667
5	8	hexagon	0.4	0.9778
5	10	hexagon	0.1	0.9722
5	10	hexagon	0.2	0.9778
5	10	hexagon	0.3	0.9889
5	10	hexagon	0.4	0.9889
10	4	hexagon	0.1	0.9556
10	4	hexagon	0.2	0.9611
10	4	hexagon	0.3	0.9944
10	4	hexagon	0.4	0.9944
10	6	hexagon	0.1	0.9611
10	6	hexagon	0.2	0.9944
10	6	hexagon	0.3	0.9833
10	6	hexagon	0.4	0.9778
10	8	hexagon	0.1	0.9722
10	8	hexagon	0.2	0.9722
10	8	hexagon	0.3	0.9833
10	8	hexagon	0.4	0.9944
10	10	hexagon	0.1	0.9889
10	10	hexagon	0.2	0.9778
10	10	hexagon	0.3	0.9722
10	10	hexagon	0.4	0.9722
15	4	hexagon	0.1	0.9778
15	4	hexagon	0.2	0.9722
15	4	hexagon	0.3	0.9778
15	4	hexagon	0.4	0.9778
15	6	hexagon	0.1	0.9833
15	6	hexagon	0.2	0.9833
15	6	hexagon	0.3	0.9611
15	6	hexagon	0.4	0.9722
15	8	hexagon	0.1	0.9944
15	8	hexagon	0.2	0.9778
15	8	hexagon	0.3	0.9833
15	8	hexagon	0.4	0.9556
15	10	hexagon	0.1	0.9833

15	10	hexagon	0.2	0.9778
15	10	hexagon	0.3	0.9667
15	10	hexagon	0.4	0.9778
20	4	hexagon	0.1	0.9778
20	4	hexagon	0.2	0.9722
20	4	hexagon	0.3	0.9778
20	4	hexagon	0.4	0.9833
20	6	hexagon	0.1	0.9833
20	6	hexagon	0.2	0.9833
20	6	hexagon	0.3	0.9833
20	6	hexagon	0.4	0.9889
20	8	hexagon	0.1	0.9778
20	8	hexagon	0.2	0.9778
20	8	hexagon	0.3	0.9833
20	8	hexagon	0.4	0.9333
20	10	hexagon	0.1	0.9722
20	10	hexagon	0.2	0.9667
20	10	hexagon	0.3	0.9833
20	10	hexagon	0.4	0.9444
5	4	default	0.1	0.9778
5	4	default	0.2	0.9944
5	4	default	0.3	0.9444
5	4	default	0.4	0.9944
5	6	default	0.1	0.9722
5	6	default	0.2	0.9889
5	6	default	0.3	0.9833
5	6	default	0.4	0.9778
5	8	default	0.1	0.9333
5	8	default	0.2	0.95
5	8	default	0.3	0.9611
5	8	default	0.4	0.9722
5	10	default	0.1	0.9
5	10	default	0.2	0.95
5	10	default	0.3	0.9722
5	10	default	0.4	0.9667
10	4	default	0.1	0.9778
10	4	default	0.2	0.9833
10	4	default	0.3	0.9833
10	4	default	0.4	0.9611
10	6	default	0.1	0.9833
10	6	default	0.2	0.9222
10	6	default	0.3	0.9667
10	6	default	0.4	0.9556
10	8	default	0.1	0.9056
10	8	default	0.2	0.9611
10	8	default	0.3	0.9778
10	8	default	0.4	0.9778

10	10	default	0.1	0.8278
10	10	default	0.2	0.9444
10	10	default	0.3	0.9778
10	10	default	0.4	0.9778
15	4	default	0.1	0.9722
15	4	default	0.2	0.9778
15	4	default	0.3	0.9111
15	4	default	0.4	0.9667
15	6	default	0.1	0.9111
15	6	default	0.2	0.9667
15	6	default	0.3	0.9722
15	6	default	0.4	0.9667
15	8	default	0.1	0.8278
15	8	default	0.2	0.9389
15	8	default	0.3	0.9556
15	8	default	0.4	0.9833
15	10	default	0.1	0.9278
15	10	default	0.2	0.9667
15	10	default	0.3	0.9833
15	10	default	0.4	0.9833
20	4	default	0.1	0.9778
20	4	default	0.2	0.9722
20	4	default	0.3	0.9722
20	4	default	0.4	0.9833
20	6	default	0.1	0.9222
20	6	default	0.2	0.9389
20	6	default	0.3	0.9722
20	6	default	0.4	0.9611
20	8	default	0.1	0.8389
20	8	default	0.2	0.9222
20	8	default	0.3	0.9667
20	8	default	0.4	0.9833
20	10	default	0.1	0.8944
20	10	default	0.2	0.9722
20	10	default	0.3	0.9611
20	10	default	0.4	0.9444