

Sieci Kohonena

Szymon Gut, 313361

April 2023

Spis treści

1 Wprowadzenie	4
2 Implementacja	6
3 Podstawowa sieć Kohonena KOH1	6
3.1 Hexagon.csv	7
3.1.1 Architektura (2,3) z funkcją Gaussa oraz promieniem sąsiedztwa równym 1	7
3.1.2 Architektura (2,3) z funkcją Gaussa oraz promieniem sąsiedztwa równym 0.8	8
3.1.3 Architektura (2,3) z funkcją Gaussa oraz promieniem sąsiedztwa równym 0.5	8
3.1.4 Architektura (2,3) z funkcją Gaussa oraz promieniem sąsiedztwa równym 0.2	8
3.1.5 Architektura (2,3) z funkcją Gaussa oraz promieniem sąsiedztwa równym 0.1	9
3.1.6 Podsumowanie dla Architektury (2,3) oraz funkcji Gaussa	10
3.1.7 Architektura (6,1) z funkcją Gaussa oraz promieniem sąsiedztwa równym 1	10
3.1.8 Architektura (6,1) z funkcją Gaussa oraz promieniem sąsiedztwa równym 0.8	11
3.1.9 Architektura (6,1) z funkcją Gaussa oraz promieniem sąsiedztwa równym 0.5	11
3.1.10 Architektura (6,1) z funkcją Gaussa oraz promieniem sąsiedztwa równym 0.2	12
3.1.11 Architektura (6,1) z funkcją Gaussa oraz promieniem sąsiedztwa równym 0.1	12
3.1.12 Podsumowanie dla architektury (6,1) z funkcją Gaussa . .	12
3.1.13 Architektura (2,3) z funkcją "Meksykański kapelusz" . .	13
3.1.14 Architektura (6,1) z funkcją meksykański kapelusz . .	13
3.1.15 Podsumowanie dla zbioru hexagon.csv	14
3.2 Cube.csv	15
3.2.1 Architektura (1,8) z funkcją Gaussa	15
3.2.2 Architektura (1,8) z funkcją Mexican hat	16
3.2.3 Architektura (4,2) z funkcją Gaussa	16
3.2.4 Architektura (4,2) z funkcją Mexican hat	16
3.2.5 Przykładowa wizualizacja etapu uczenia	17
3.2.6 Podsumowanie dla zbioru cube.csv	19
4 Sieć Kohonena na siatce sześciokątnej	21
4.1 Zbiór danych MNIST	21
4.1.1 Funkcja Gaussa jako funkcja sąsiedztwa	21
4.1.2 Funkcja Meksykański kapelusz	27
4.2 Zbiór danych Smartphone Activity	33

4.2.1	Funkcja Gaussa	34
4.2.2	Funkcja Meksykański kapelusz	40
4.3	Podsumowanie	46
4.3.1	MNIST	46
4.3.2	Smartphone activity	47
4.3.3	Znalezienie najbardziej optymalnej liczby klastrów	47
5	Wnioski oraz podsumowanie	50

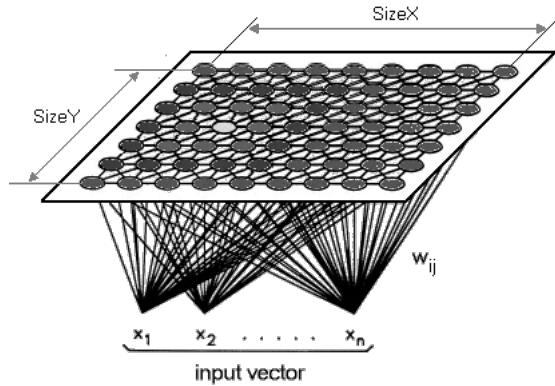
1 Wprowadzenie

W ramach laboratoriów została zaimplementowana sieć Kohonena oraz została ona przetestowana na kilku zbiorach danych. Sieć Kohonena, nazywana również mapą Kohonena, to jedna z popularnych metod nienadzorowanego uczenia maszynowego, stosowana do analizy danych i redukcji wymiarów. Sieć ta została opracowana przez fińskiego naukowca Teuvo Kohonena w latach 80-tych XX wieku.

Sieć Kohonena składa się z jednowarstwowej liniowej siatki neuronów 2D. Każdy neuron w sieci jest charakteryzowany przez wektor wag, które określają jego zachowanie. W procesie uczenia, sieć Kohonena szuka wzorców w danych wejściowych, a następnie grupuje je w klastry na podstawie podobieństw między nimi.

Podczas uczenia, każdy neuron jest dostosowywany tak, aby reagować na określony zestaw danych wejściowych. Wagi neuronów są modyfikowane na podstawie odległości między nimi, a wektorem wejściowym. Neuron, którego wektor wag jest najbliższym do wektora wejściowego, zostaje uznany za zwycięzcę i wygrywa w konkurencji z innymi neuronami w sieci. Dzięki temu, że neurony są połączone w sieć, możliwe jest grupowanie danych wejściowych na podobieństwo do siebie.

Po zakończeniu procesu uczenia, każdy neuron reprezentuje jeden klaszt lub grupę danych wejściowych. Sieć Kohonena jest więc przydatna do klasyfikacji danych lub redukcji wymiarów, gdyż pozwala na zgrupowanie podobnych danych w jednym klastrze.

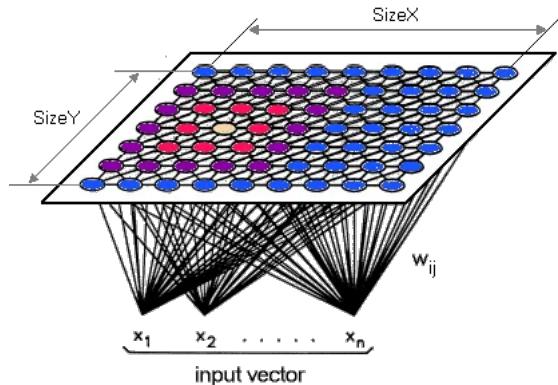


Rysunek 1: Wizualizacja Sieci Kohonena.

Na rysunku powyżej została przedstawiona wizualizacja Sieci Kohonena. Sieć Kohonena jest również nazywana samoorganizującą się mapą, gdyż całe uczenie odbywa się bez nadzoru. Różni się ona od typowych podejść do sztucznych sieci neuronowych, zarówno architekturą, jak i właściwościami algorytmicznymi. Pierwszą wyróżniającą ją cechą jest wcześniejsza wspomniana struktura tej sieci.

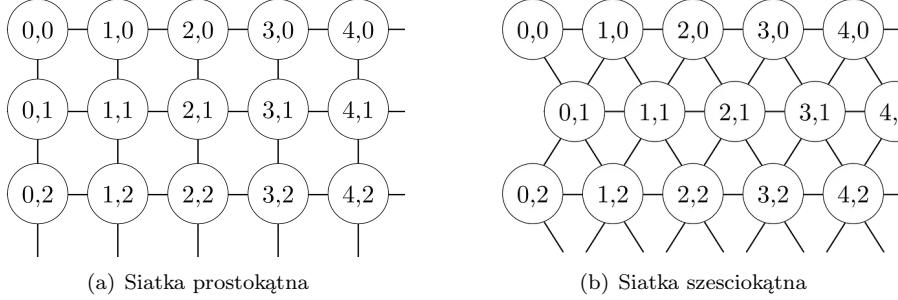
Składa się ona z jednowarstwowej liniowej siatki neuronów 2D, zamiast szeregu warstw. Wszystkie wezły w tej siatce połączone są bezpośrednio z wektorem wejściowym, lecz co ważne, nie są one połączone ze sobą, co oznacza że neurony nie znają wartości swoich sąsiadów, a wagi swoich połączeń aktualizują w funkcji danych wejściowych. Wspomniana siatka 2D jest mapą organizującą się podczas każdej iteracji algorytmu. Po każdej ietarcji każdy z neuronów posiada swoje współrzędne (i,j) na tej siatce, co pozwala do policzenia odległości Euklidesowej pomiędzy dwoma węzłami za pomocą twierdzenia Pitagorasa.

Przechodząc następnie do właściwości wyróżniających samoorganizujące się mapy, ważny jest fakt, że wykorzystują one konkurencyjne uczenie, w przeciwieństwie do uczenia z korekcją błędu w propagacji wstępnej. W rzeczywistości oznacza to, że w każdej iteracji aktywowany jest tylko jeden neuron, ponieważ wszystkie neurony konkurują o prawo do odpowiedzi na dane wejściowe. Wspomniany neuron nosi nazwę Best Match Unit (BMU) i wybierany jest na podstawie podobieństwa między aktualnymi wartościami wejściowymi, a wszystkimi węzłami w siatce. Węzeł, który posiada najmniejszą odległość Euklidesową w porównaniu do wektora wejściowego, jest wybierany wraz z sąsiednimi neuronami w sterowalnym, pewnym promieniu, aby ich położenie zostało nieznacznie dopasowane, w taki sposób aby pasowało do wektora wejściowego. Iterując po wszystkich neuronach w siatce, cała siatka zostaje dopasowana do zbioru danych wejściowych, z podobnymi węzłami zgrupowanymi razem oraz oddzielonymi tymi odmiennymi. Neurony w siatce mogą być ułożone w topologii siatki sze-



Rysunek 2: Model samoorganizującej się mapy z BMU zaznaczonym kolorem żółtym, neuronami wewnątrz promienia sąsiedztwa w kolorze różowym oraz fioletowym oraz neurony na zewnątrz tego promienia zaznaczone kolorem niebieskim.

ściołatej oraz prostokątnej. Warianty te zostały zaimplementowane w ramach laboratorium a rezultaty będą umieszczone w dalszej części raportu. Zdjęcia wiadoczne w tej sekcji zostały zapożyczone ze strony Towards data science w celach edukacyjnych.



2 Implementacja

W ramach laboratorium został zaimplementowany algorytm samoorganizującej się mapy Kohonena, z możliwością wyboru funkcji sąsiedztwa: funkcja Gaussa lub jej druga pochodna zwana również funkcją "meksykański kapelusz" z faktu, że w dużym stopniu przypomina ona "sombrero". Została również zaimplementowana możliwość zmieniać topologię na siatkę sześciokątną lub prostokątną. Do testowania wyników klasteryzacji wykorzystywane były różne architektury, gdzie zarówno zmienialiśmy wymiary siatki, jak i promień sąsiedztwa.

3 Podstawowa sieć Kohonena KOH1

W ramach tej pracy domowej został zaimplementowany model całej sieci Kohonena (jeszcze bez możliwości wyboru topologii siatki sześciokątnej) wraz z dwoma wcześniej wspomnianymi funkcjami sąsiedztwa. W obu implementacjach została dodana możliwość zmiany szerokości sąsiedztwa, a następnie przetestowano wyniki klasteryzacji dla kilku wartości z przedziału [0.1,1]. Jako funkcja wygaszająca została wykorzystana $\alpha(t) = e^{-\frac{t}{\lambda}}$. Sieć na tym etapie została przetestowana na dwóch zbiorach danych:

- **hexagon.csv**, czyli danych 2d skupionych w wierzchołkach sześciokąta
- **cube.csb**, czyli danych 3d skupionych w wierzchołkach sześciianu

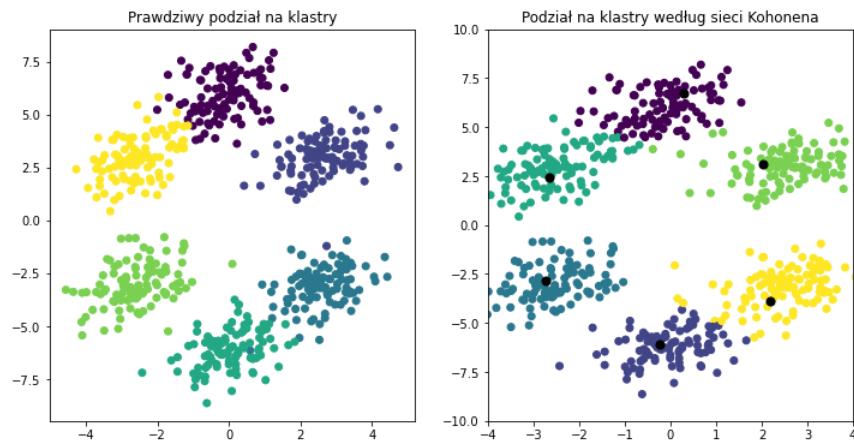
Na pierwszym zbiorze danych rozważono po dwie architektury dla każdej z funkcji sąsiedztwa tj. odpowiednio (2,3) oraz (6,1) tak aby liczba centroidów zgadzała się z faktyczną liczbą klastrów wymienioną w tym zbiorze danych. Rozważano również kilka wartości promienia sąsiedztwa, a wyniki testowania zostały zaprezentowane poniżej. Została również zbadana wartość metryki Adjusted Rand Score.

Adjusted Rand Score to miara podobieństwa między dwoma wynikami grupowania. Jest to wariant indeksu Randa, który porównuje podobieństwo między dwoma zestawami etykiet przypisanych do tego samego zbioru danych, gdzie etykiety reprezentują przypisania do klastrów. Różni się on od indeksu Randa,

gdyż ten może być wrażliwy na przypadkowe przypisanie do klastrów, natomiast Adjusted Rand Score koryguje losowe przyporządkowanie i dla losowego przypisania przyjmuje wartość zero. Zatem oblicza on miarę podobieństwa między dwoma skupieniami, biorąc pod uwagę wszystkie pary próbek i licząc pary, które są przypisane do tych samych lub różnych skupień w przewidywanych i prawdziwych skupieniach. $ARI = (RI - Expected_RI) / (max(RI) - Expected_RI)$

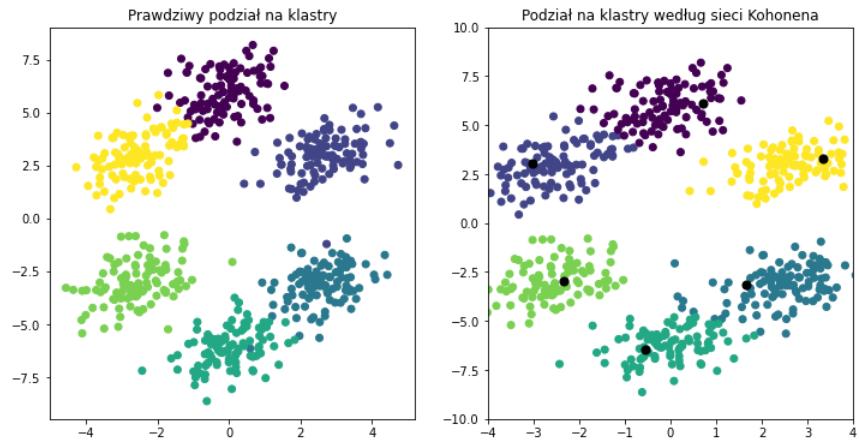
3.1 Hexagon.csv

3.1.1 Architektura (2,3) z funkcją Gaussa oraz promieniem sąsiedzwa równym 1



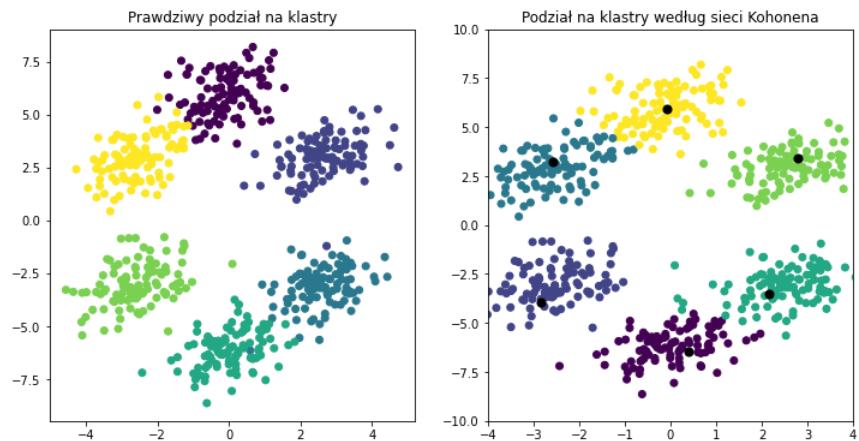
Rysunek 3: Wizualizacja klasteryzacji, uzyskany adjusted rand score: 0.926

3.1.2 Architektura (2,3) z funkcją Gaussa oraz promieniem sąsiedz-twa równym 0.8



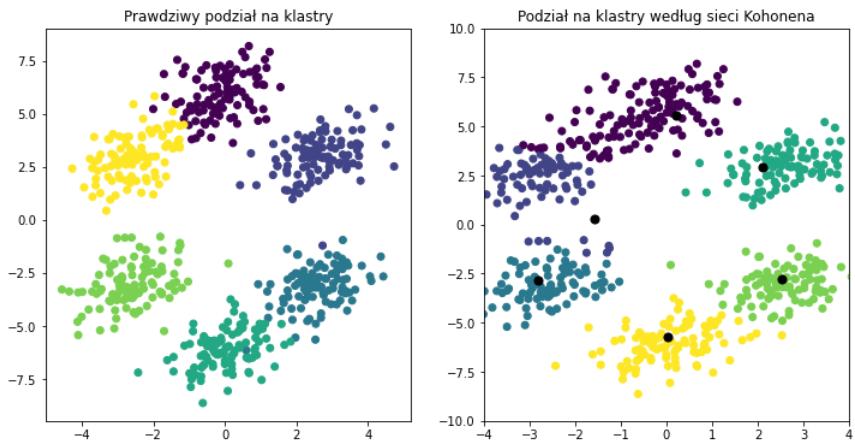
Rysunek 4: Wizualizacja klasteryzacji, uzyskany adjusted rand score: 0.937

3.1.3 Architektura (2,3) z funkcją Gaussa oraz promieniem sąsiedz-twa równym 0.5



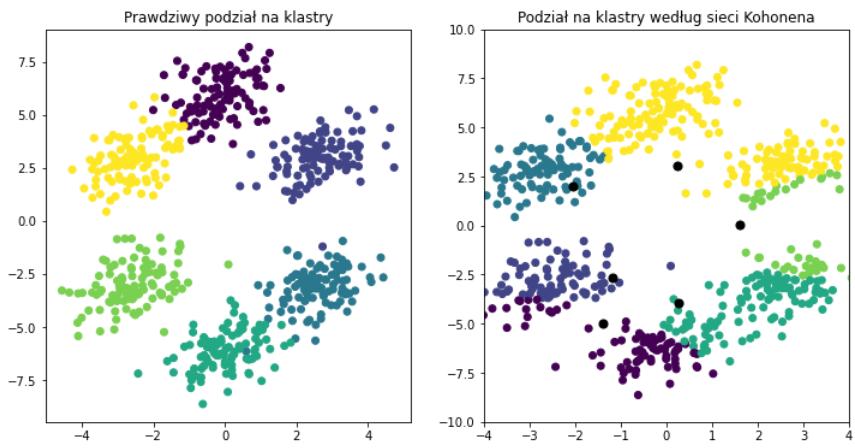
Rysunek 5: Wizualizacja klasteryzacji, uzyskany adjusted rand score: 0.945

3.1.4 Architektura (2,3) z funkcją Gaussa oraz promieniem sąsiedz-twa równym 0.2



Rysunek 6: Wizualizacja klasteryzacji, uzyskany adjusted rand score: 0.845

3.1.5 Architektura (2,3) z funkcją Gaussa oraz promieniem sąsiedz-twa równym 0.1



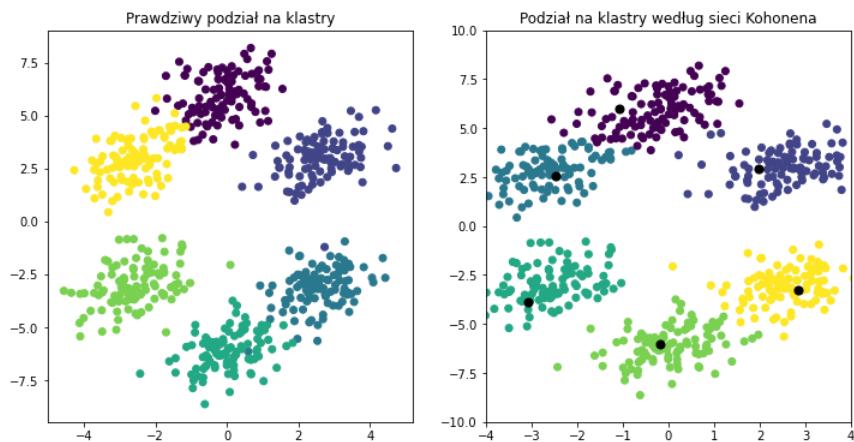
Rysunek 7: Wizualizacja klasteryzacji, uzyskany adjusted rand score: 0.577

3.1.6 Podsumowanie dla Architektury (2,3) oraz funkcji Gaussa

Liczba epok	9	9	9	9	9
Skala sąsiedztwa	1	0.8	0.5	0.2	0.1
Adjusted rand score	0.925671	0.937236	0.944928	0.845137	0.577262

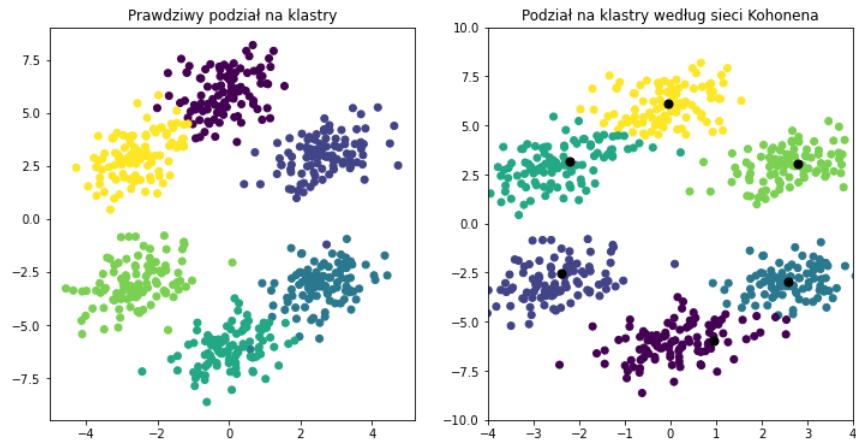
Co można zauważyć to fakt, że najwyższa wartość adjusted score została przyjęta dla skali sąsiedztwa równej 0.5. Dla mniejszych skal klasteryzacja nie przebiegala tak efektownie jak dla wartości większych. Dodatkowo był wtedy obserwowany efekt przypisywania większości punktów do jednego klastra w pierwszej epoce, podczas gdy dla większych wartości skali sąsiedztwa już po pierwszej epoce były widoczne ładne, uporządkowane klastry.

3.1.7 Architektura (6,1) z funkcją Gaussa oraz promieniem sąsiedztwa równym 1



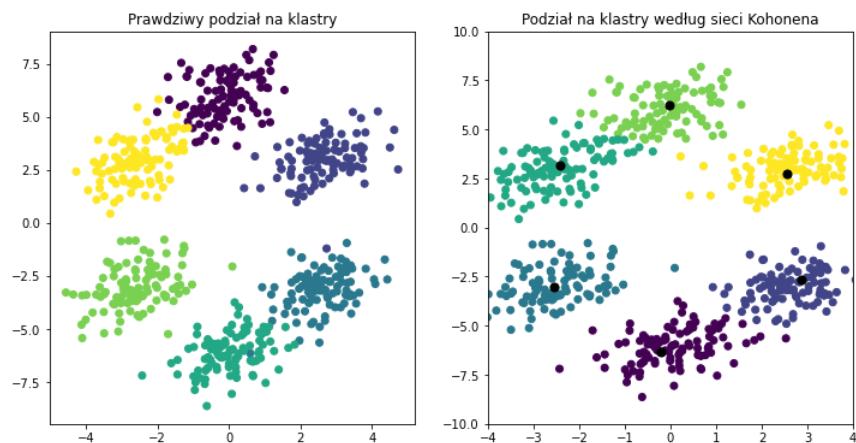
Rysunek 8: Wizualizacja klasteryzacji, uzyskany adjusted rand score: 0.92

3.1.8 Architektura (6,1) z funkcją Gaussa oraz promieniem sąsiedz-twa równym 0.8



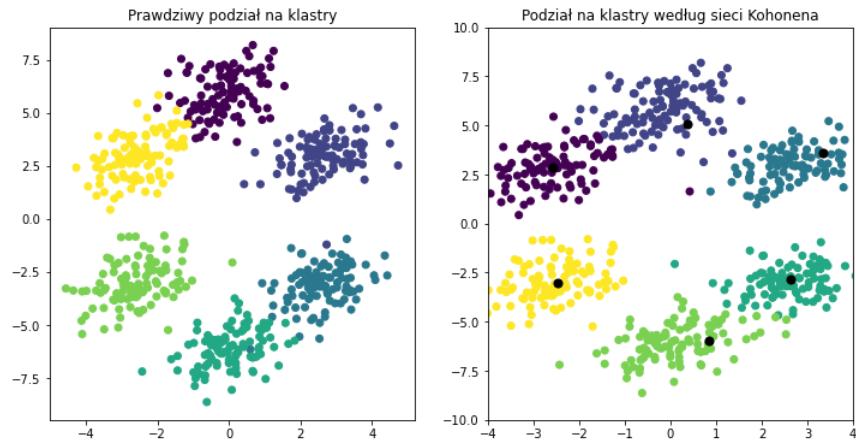
Rysunek 9: Wizualizacja klasteryzacji, uzyskany adjusted rand score: 0.93

3.1.9 Architektura (6,1) z funkcją Gaussa oraz promieniem sąsiedz-twa równym 0.5



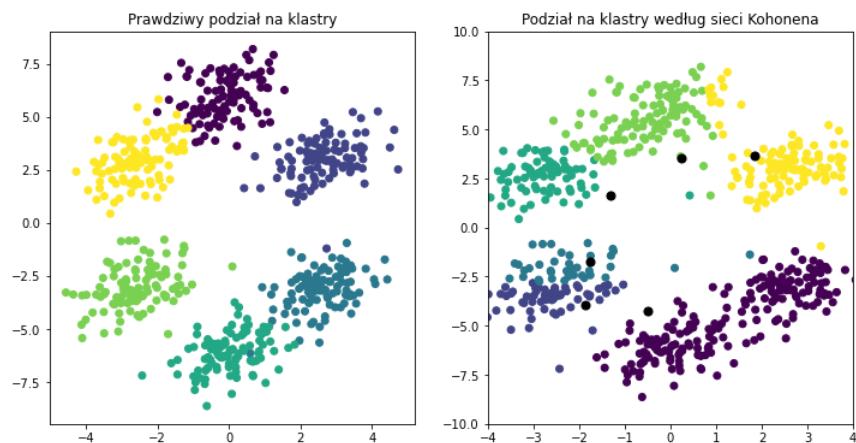
Rysunek 10: Wizualizacja klasteryzacji, uzyskany adjusted rand score: 0.94

3.1.10 Architektura (6,1) z funkcją Gaussa oraz promieniem sąsiedz-twa równym 0.2



Rysunek 11: Wizualizacja klasteryzacji, uzyskany adjusted rand score: 0.93

3.1.11 Architektura (6,1) z funkcją Gaussa oraz promieniem sąsiedz-twa równym 0.1



Rysunek 12: Wizualizacja klasteryzacji, uzyskany adjusted rand score: 0.64

3.1.12 Podsumowanie dla architektury (6,1) z funkcją Gaussa

Liczba epok	9	9	9	9	9
Skala sąsiedztwa	1	0.8	0.5	0.2	0.1
Adjusted rand score	0.919109	0.930128	0.941381	0.933394	0.644498

Tutaj również wyniki klasteryzacji mamy bardzo podobne. Najwyższą wartość adjusted rand score uzyskano dla skali sąsiedztwa równej 0.5 oraz 0.2. Sieć zdecydowanie nie radzi sobie po ustawieniu tej skali na wartość 0.1. Dla tej wartości skali uzyskane adjusted rand score jest równe 0.64. Błedy wynikające z klasteryzacji są tam już bardzo mocno zauważalne.

3.1.13 Architektura (2,3) z funkcją "Meksykański kapelusz"

W dalszej części tego sprawozdania pominięto już wizualne efekty klasyfikacji, gdyż są one bardzo podobne i nie wnoszą zbyt dużo informacji. W celu przedstawienia efektów testowania dla nowej funkcji sąsiedztwa sporządzono tabelę. Podsumowując, wybierając funkcję Meksykański kapelusz jako funkcję sąsiedz-

Liczba epok	16	16	16	16	16
Skala sąsiedztwa	0.5	0.4	0.3	0.2	0.1
Adjusted rand score	0.904022	0.830770	0.919152	0.899238	0.713055

twa najlepsze rezultaty otrzymaliśmy przyjmując skalę sąsiedztwa równą 0.3. Wyniki zazwyczaj są dość zbliżone jednakże dla zbyt małego promienia sąsiedztwa adjusted score znacznie odbiega od pozostałych, uzyskanych wartości (dla neighbourhood scale = 0.1 adjusted rand score wynosił zaledwie 0.71).

3.1.14 Architektura (6,1) z funkcją meksykański kapelusz

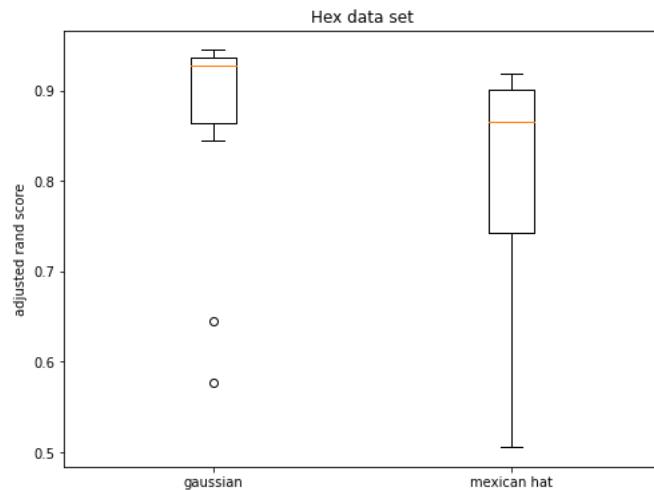
Liczba epok	16	16	16	16	16
Skala sąsiedztwa	0.5	0.4	0.3	0.2	0.1
Adjusted rand score	0.610430	0.852217	0.880340	0.901586	0.505897

Warto zauważyć, iż pomimo że mamy nadal tyle samo klastrów co wcześniej tj. 6, to bardzo ważny jest wybór architektury. Pomimo, że dla wcześniejszej architektury uzyskaliśmy najwyższą wartość metryki adjusted rand score dla skali sąsiedztwa równej 0.3, to w tym przypadku najwyższy wynik tj. 0.90 uzyskano

dla skali sąsiedztwa równej 0.2. Co więcej w wcześniejszej architekturze wysoką wartość metryki uzyskalismy również dla skali równej 0.5, co w tym przypadku nie ma żadnego przełożenia gdyż dla tej architektury metryka odpowiadająca tej skali jest jedną z najniższych.

3.1.15 Podsumowanie dla zbioru hexagon.csv

Podsumowując na zbiorze hexagon.csv przetestowaliśmy łącznie 20 architektur, zmieniając w nich wymiary siatki, funkcję sąsiedztwa oraz skalę sąsiedztwa. Która funkcja sąsiedztwa okazała się lepsza dla tego zbioru danych? Odpowiedź na to pytanie została przedstawiona poniżej w formie graficznej.



Rysunek 13: Boxplot przedstawiający wartości adjusted rand score dla dwóch funkcji sąsiedztwa na podstawie eksperymentów

Pierwszą najważniejszą obserwacją wynikającą z tego wykresu jest rozstęp kwartylowy dla uzyskanych wartości metryk. Widzimy, że dla funkcji Gaussowskiej otrzymywane wartości metryki są w większości wyższe, niż te uzyskane dla funkcji Mexican hat. Wynika to zarówno z mediany, która na podstawie uzyskanego wykresu jest wyższa dla pierwszej funkcji, jak i również analizując dolny oraz górny kwartył.

Na sam koniec sprawdziliśmy również jaką liczbę klastrów dostaliśmy za najbardziej optymalną, gdybyśmy przeprowadzili uczenie nienadzorowane. W tym celu stworzyliśmy kilka architektur klasyfikujących odpowiednio do 3, 4, 5, 6, 7 oraz 8 klastrów i zobaczyliśmy dla której architektury uzyskamy najlepsze wartości metryk Shilhouette oraz Calinski Harabasz. Wyniki przedstawione poniżej.

Zanim przejdziemy do interpretacji uzyskanych wartości metryk warto opisać jaka jest interpretacja powyższych metryk. Zaczynając od metryki Silhouette,

Architektura	Liczba Klastrów	Silhouette Score	Calinski Harabasz score
(3,1)	3	0.490620	887.555609
(2,2)	4	0.486331	978.144153
(5,1)	5	0.511300	1064.043198
(2,3)	6	0.592296	1802.240350
(6,1)	6	0.591400	1796.484258
(7,1)	7	0.551921	1642.492581
(4,2)	8	0.478589	1405.200349

która określa, jak dobrze punkty wewnętrz klastra są ze sobą powiązane oraz jak dobrze punkty między klastrami są oddzielone. Wartości tej metryki mieszczą się w przedziale od -1 do 1. Wartości bliskie 1 oznaczają, że punkty w klastrze są dobrze ze sobą powiązane, a punkty między klastrami są słabo ze sobą powiązane. Wartości bliskie -1 oznaczają, że punkty w klastrze są słabo ze sobą powiązane, a punkty między klastrami są dobrze ze sobą powiązane.

Przechodząc natomiast do drugiej metryki tj. Calinski-Harabasz, która jest miarą jakości klastrów w analizie skupień. Wartość tej metryki jest tym większa, im klastry są lepiej oddzielone. W szczególności metryka ta oblicza stosunek wariancji między klastrami do wariancji wewnętrz klastrów, co oznacza, że im większa wartość metryki, tym lepiej klastry są separowalne.

Przechodząc już do interpretacji uzyskanych wyników, można zauważyc że najlepsze wartości tych metryk uzyskaliśmy dla architektury (2,3), czyli odpowiadającej 6 klastrom. Wynik zatem jest zgodny z oczekiwaniemi, gdyż oryginalnie ten zbiór właśnie zawierał 6 klastrów odpowiadających wierzchołkom sześciokąta.

3.2 Cube.csv

Następnym zbiorem na którym testowano zaimplementowaną sieć Kohonena był zbiór cube.csv, który był zbiorem danych 3D skupionych w wierzchołkach sześcianu. Występowało, więc w nim 8 klastrów, a rozważane architektury to (1,8) oraz (4,2).

3.2.1 Architektura (1,8) z funkcją Gaussa

Dla tej architektury stworzyliśmy 4 różne sieci, odpowiednio zmieniając przy tym skale sąsiedztwa. Sprawdzone wartości to: 1, 0.8, 0.5, 0.1. Wyniki testowania zaprezentowane w tabeli poniżej. Najwyższą wartość metryki Adjusted rand score uzyskano dla skali sąsiedztwa równej 0.8 oraz 0.5. W porównaniu natomiast do zbioru hexagon.csv dla skali sąsiedztwa równej 0.1 otrzymano również dosyć wysoką wartość wspomnianej metryki.

Liczba epok	16	16	16	16
Skala sąsiedztwa	1	0.8	0.5	0.1
Adjusted rand score	0.847591	0.870316	0.878707	0.800619

3.2.2 Architektura (1,8) z funkcją Mexican hat

Przechodząc do drugiej funkcji sąsiedztwa, tutaj także sprawdzono wyniki dla 4 różnych wartości skali. Najwyższe wartości metryki uzyskano dla skali równej 0.2 oraz 0.15. Dla mniejszych wartości skali wyniki metryki znacznie odbiegały od tych najlepszych i były równe około 0.59.

Liczba epok	16	16	16	16
Skala sąsiedztwa	0.2	0.15	0.1	0.05
Adjusted rand score	0.749900	0.728339	0.593969	0.562671

3.2.3 Architektura (4,2) z funkcją Gaussa

Przechodząc do innego wymiaru siatki oraz wracając do testowania funkcji Gaussa dla tej nowej architektury otrzymaliśmy porównywalne wyniki jak dla architektury z wymiarem siatki równym (1,8). Wyniki uzyskane dla tej architektury osiągają najwyższą wartość adjusted rand score równą 0.86 co jest nieznacznie gorszą wartością od tej uzyskanej dla wcześniejszej architektury.

Liczba epok	16	16	16	16
Skala sąsiedztwa	1	0.8	0.5	0.1
Adjusted rand score	0.857861	0.838205	0.859433	0.842178

3.2.4 Architektura (4,2) z funkcją Mexican hat

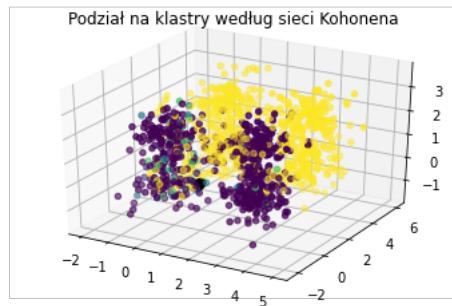
Tutaj również uzyskane wartości są dość zbliżone do wartości uzyskanych dla architektury (1,8). Pierwszą rzucającą się w oczy większą zmianą jest wynik uzyskany dla skali sąsiedztwa równej 0.1. W tej architekturze adjusted rand score uzyskany przy takim zestawieniu jest równy 0.76, podczas gdy dla architektury (1,8), wartość ta wynosiła zaledwie 0.59.

Liczba epok	16	16	16	16
Skala sąsiedztwa	0.2	0.15	0.1	0.05
Adjusted rand score	0.702870	0.648794	0.762420	0.466441

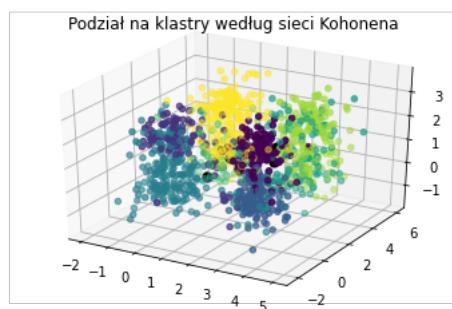
3.2.5 Przykładowa wizualizacja etapu uczenia

Z faktu, że klasteryzacja każdego modelu wygląda podobnie, w wcześniejszych sekcjach podczas prezentowania wyników również skupiłem się na tabelach i wynikach końcowych. W tej sekcji chciałbym zaprezentować wizualizację procedury uczenia dla dwóch różnych funkcji sąsiedztwa.

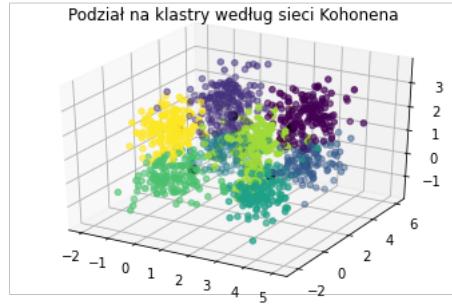
Gaussian function



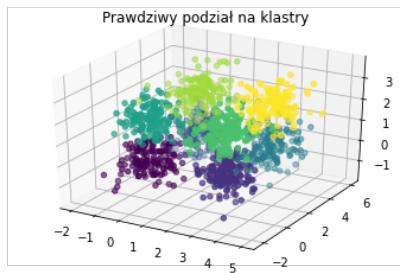
Rysunek 14: Klasteryzacja po 1 epoce dla funkcji Gauss



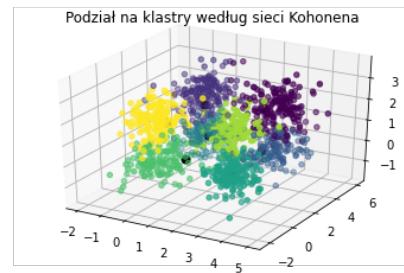
Rysunek 15: Klasteryzacja po 6 epokach dla funkcji Gauss



Rysunek 16: Klasteryzacja po 12 epoce dla funkcji Gauss

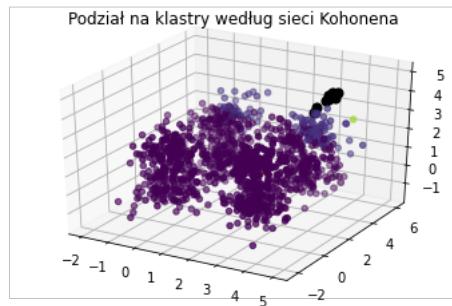


Rysunek 17: Prawdziwa klasteryzacja

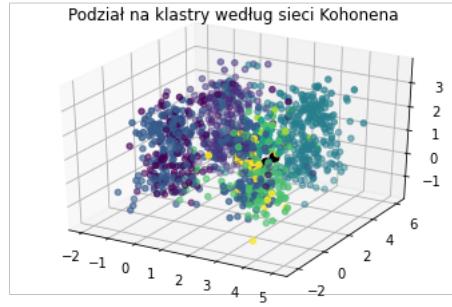


Rysunek 18: Klasteryzacja po 16 epokach dla funkcji Gauss

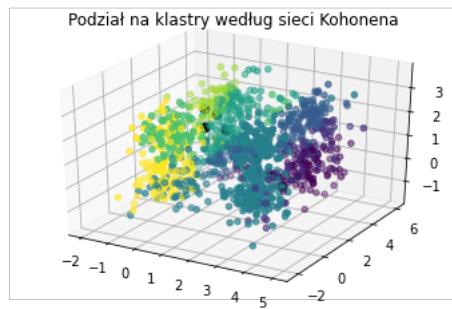
Mexican hat function



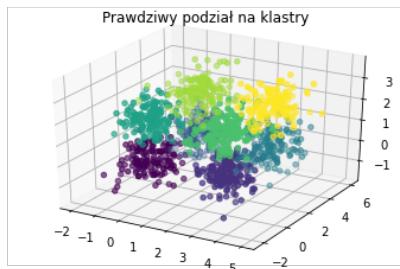
Rysunek 19: Klasteryzacja po 1 epoce dla funkcji mexican hat



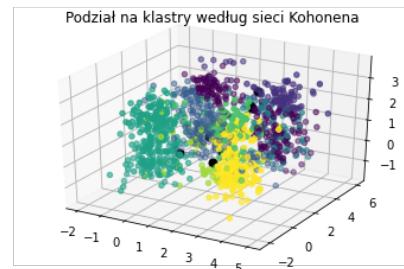
Rysunek 20: Klasteryzacja po 6 epokach dla funkcji mexican hat



Rysunek 21: Klasteryzacja po 12 epoce dla funkcji mexican hat



Rysunek 22: Prawdziwa klasteryzacja

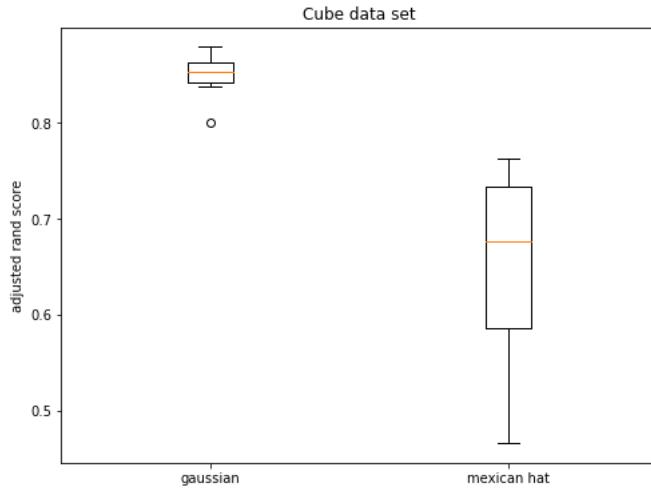


Rysunek 23: Klasteryzacja po 16 epokach dla funkcji mexican hat

3.2.6 Podsumowanie dla zbioru cube.csv

Po przetestowaniu wszystkich stworzonych architektur, również na tym zbiorze wyższe wartości metryki uzyskano dla funkcji Gaussa jako funkcji sąsiedztwa. Dla wszystkich uzyskanych wartości adjusted rand score sporządzono wykres

typu boxplot, z którego łatwo odczytać charakterystykę otrzymanych wyników.



Rysunek 24: Boxplot przedstawiający wartości adjusted rand score dla dwóch funkcji sasiedztwa na podstawie eksperymentów

Możemy zaobserwować, że mediana i dolny oraz górny kwartyl wartości metryki dla funkcji gaussowskiej układa się zdecydowanie wyżej, niż dla Mexican hat.

Następnie również stworzono kilka architektur i porównano wartości metryk Silhouette oraz Calinski-Harabasz w celu wyboru najbardziej optymalnej liczby klastrów. Następnie liczbę tą porównano z faktyczną liczbą klastrów zaprezentowanych w tym zbiorze. Porównując otrzymane wyniki, najwyższą wartość

Architektura	Liczba klastrów	Metryka Silhouette	Metryka Calinski-Harabasz
(2,2)	4	0.466048	1265.808011
(5,1)	5	0.445347	1132.109698
(2,3)	6	0.413953	1084.396983
(7,1)	7	0.409516	1103.611284
(4,2)	8	0.360055	1076.375251
(8,1)	8	0.407398	1228.909759

uzyskaliśmy dla 4 klastrów. Nie odpowiada to grupowaniu w zbiorze danych cube.csv, gdyż mamy tam aż 8 klastrów odpowiadających wierzchołkom sześcianu. Warto jednak zaważyć, że podczas gdy metryka Silhouette przyjmuje drugi najgorszy wynik dla 8 klastrów (architektura 8,1), to wartość metryki Calinski-Harabasz dla tego samego modelu jest drugim najlepszym uzyskanym wynikiem.

4 Sieć Kohonena na siatce sześciokątnej

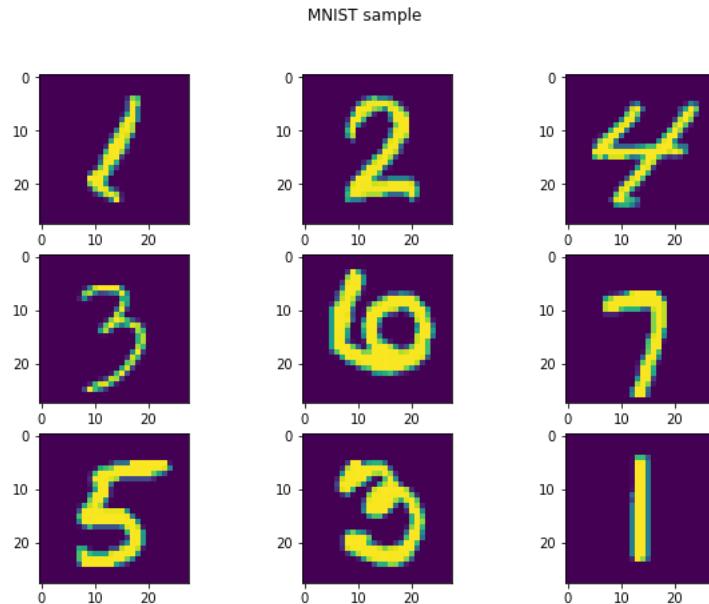
Podczas tej pracy domowej została dodana możliwość ułożenia neuronów w topologii siatki szesciokątnej. Następnie oba warianty topologii oraz funkcji sąsiedztwa zostały przetestowane na dwóch zbiorach:

- MNIST
- zbiór z odczytami czujników w smartphone

4.1 Zbiór danych MNIST

Zbiór danych MNIST to popularny zbiór danych zawierający zdjęcia odręcznie napisanych cyfr. W ramach tego zbioru danych zostały przetestowane różne architektury oraz zmierzzone dla nich popularne metryki: **silhouette score**, **Calinski-Harabasz score**, **Adjusted rand score**, **Rand score** oraz **V-score**. Wyniki testowania zostaną zaprezentowane w kolejnej sekcji.

Nasza architektura została przetestowana na 6000 próbkach (10% całego zbioru danych) z tego zbioru w celu przyśpieszenia procesu klasteryzacji.



Rysunek 25: Przykładowa reprezentacja zbioru danych MNIST

4.1.1 Funkcja Gaussa jako funkcja sąsiedztwa

W ramach funkcji Gaussa jako funkcji sąsiedztwa zostały stworzone łącznie 4 architektury. Różniły się one od siebie rodzajem siatki: **sześciokątna** oraz

prostokątna, jak i liczbą neuronów. Rozważane wymiary siatki to (4,4) oraz (10,1).

Badając metryki nadzorowane np. Adjusted rand score, trzeba było zadbać aby wektor prawdziwych klas oraz wektor klas uzyskanych jako wynik klasteryzacji posiadały tyle samo unikalnych wartości. W tym celu napisano funkcję łączącą nadmiarowe centroidy, gdy ich odległość jest odpowiednio mała. Implementacja tej funkcji z uwagi na kluczową rolę przy badaniu skuteczności danych architektur została zaprezentowana poniżej.

```

def merge_closest_neurons_and_predict(self , data):
    centers = np.reshape(self.weights , (-1, self.input_shape))
    num_clases = len(np.unique(self.all_data[:, -1]))
    while centers.shape[0] > num_clases:
        distances = np.zeros((centers.shape[0], centers.shape[0]))
        np.fill_diagonal(distances , np.inf)
        for i in range(centers.shape[0]):
            for j in range(i+1, centers.shape[0]):
                distances[i, j] = euclidean_distance(centers[i] , centers[j])
                distances[j, i] = distances[i, j]

        i , j = np.unravel_index(np.argmin(distances) , distances.shape)

        new_neuron = (centers[i] + centers[j]) / 2

        centers = np.delete(centers , [i, j] , axis=0)
        centers = np.vstack([centers , new_neuron])

    data_clusters = []
    for row in data:
        distances = np.array(
            [euclidean_distance(row, center) for center in centers])
        data_clusters.append(np.argmin(distances))
    return np.array(data_clusters)

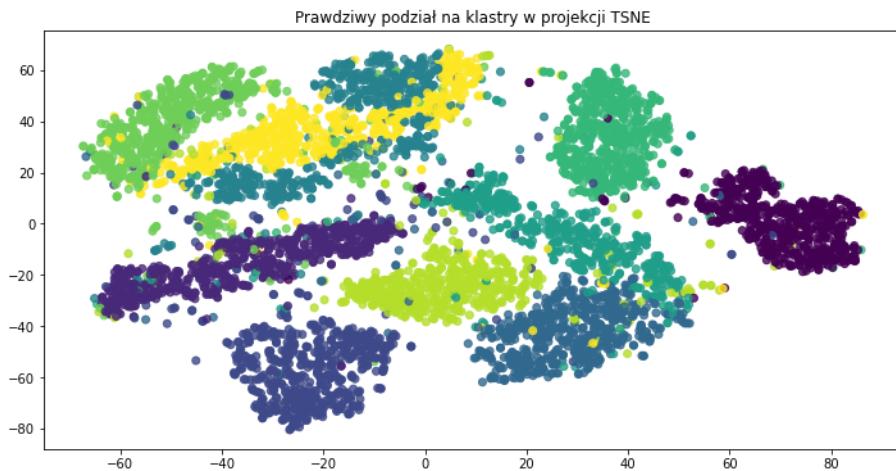
```

Pierwszą rozpatrywaną architekturą była:

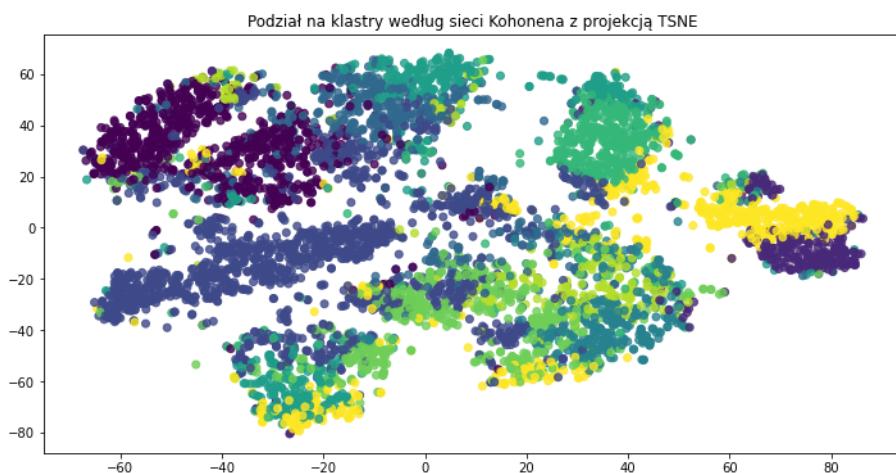
```

kohNet_mnist_gauss1 = Network(input_shape=784, shape=(10,1))
kohNet_mnist_gauss1.fit(mnist_df, 5, neighbourhood_scale=1,
                        grid_type='rectangle' , tsne_data=tsne_data)

```



Rysunek 26: Prawdziwe klastry



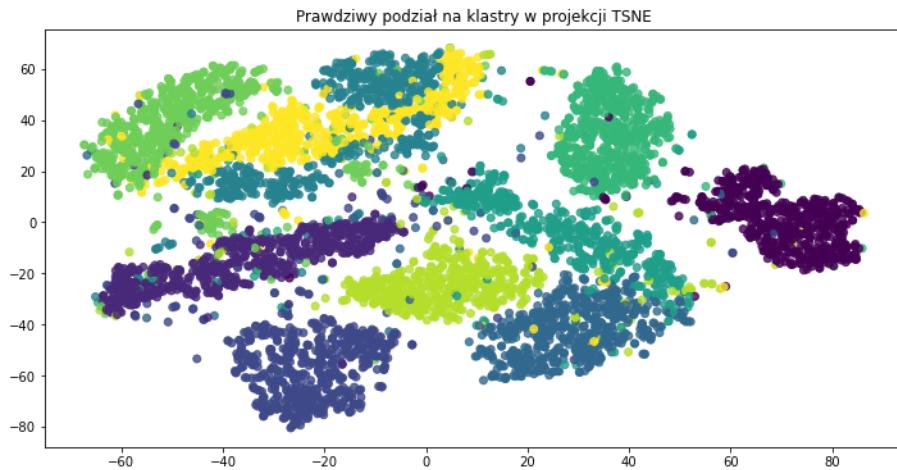
Rysunek 27: Klasteryzacja uzyskana przez sieć Kohonena

Wartości metryk dla tej sieci:

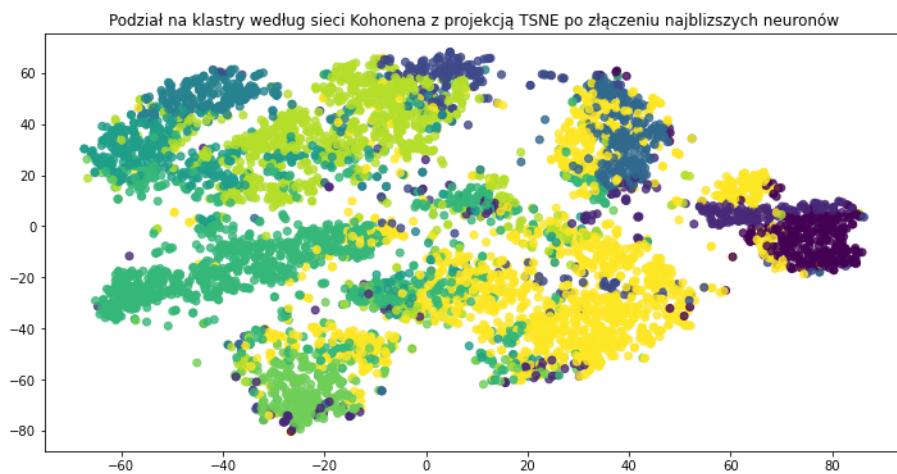
- Calinski-Harabasz metric: 187.0077008794153
- Silhouette metric: 0.06208088563998798
- Rand Score: 0.8404777462910485
- Adjusted Rand Score: 0.24518648285392133
- V-measure: 0.380734520265637

Kolejną rozpatrywaną architekturą była architektura z siatką (4,4).

```
kohNet_mnist_gauss2 = Network(input_shape=784, shape=(4,4))
kohNet_mnist_gauss2.fit(mnist_df, 15, neighbourhood_scale=0.5,
                        grid_type='rectangle', tsne_data=tsne_data)
```



Rysunek 28: Prawdziwe klastry



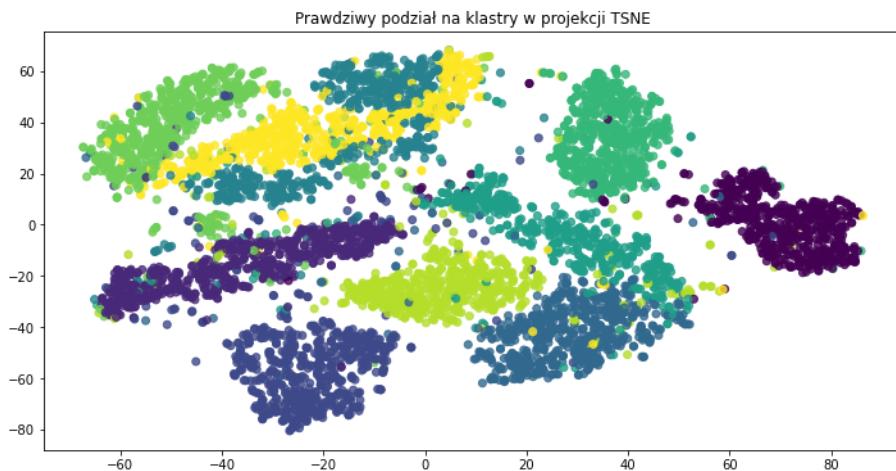
Rysunek 29: Klasteryzacja uzyskana przez sieć Kohonena po złączeniu najbliższych neuronów

Wartości uzyskanych metryk dla tej sieci:

- Calinski-Harabasz metric: 142.45320251027124
- Silhouette metric: 0.049189893999693285
- Rand Score: 0.8269127632383175
- Adjusted Rand Score: 0.24532357104325175
- V-measure: 0.39801743561941366

Trzecią architekturą była sieć Kohonena z hexagonalną siatką oraz z 10 neuronami.

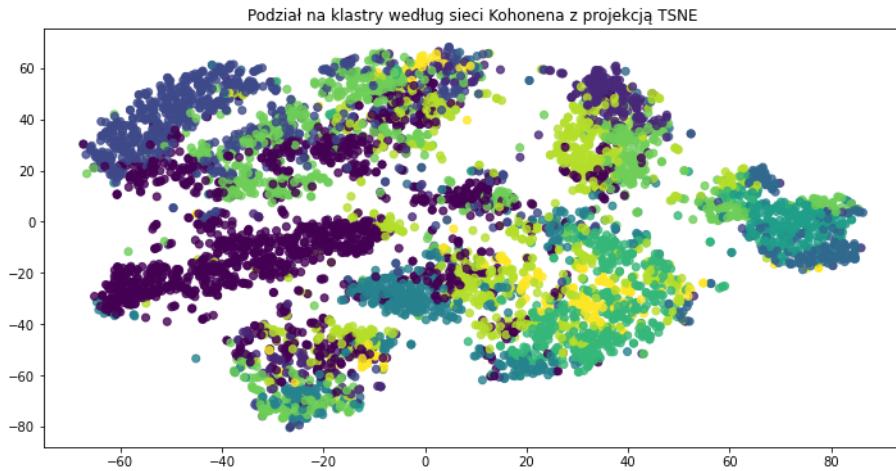
```
kohNet_mnist_gauss3 = Network(input_shape=784, shape=(10,1))
kohNet_mnist_gauss3.fit(mnist_df, 5, neighbourhood_scale=1,
                        grid_type='hex', tsne_data=tsne_data)
```



Rysunek 30: Prawdziwe klastry

Wartości metryk dla tej sieci:

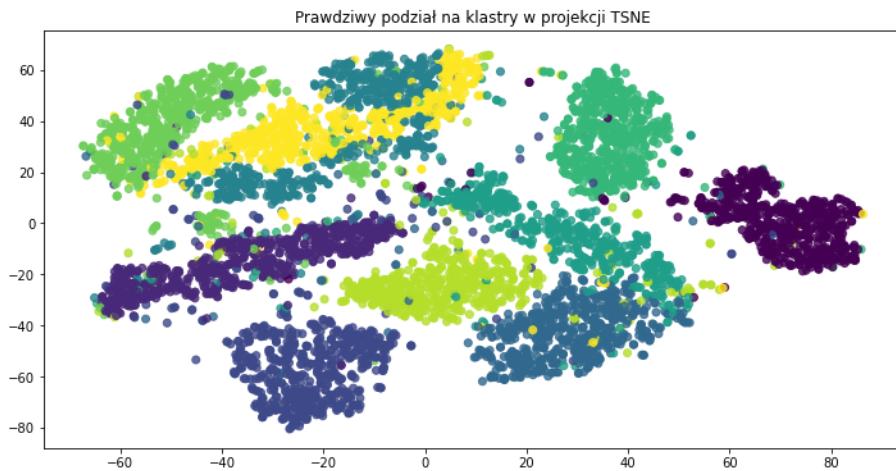
- Calinski-Harabasz metric: 166.09955308680475
- Silhouette metric: 0.05129647648869244
- Rand Score: 0.8247023392787687
- Adjusted Rand Score: 0.20389805033711322
- V-measure: 0.33169834640608675



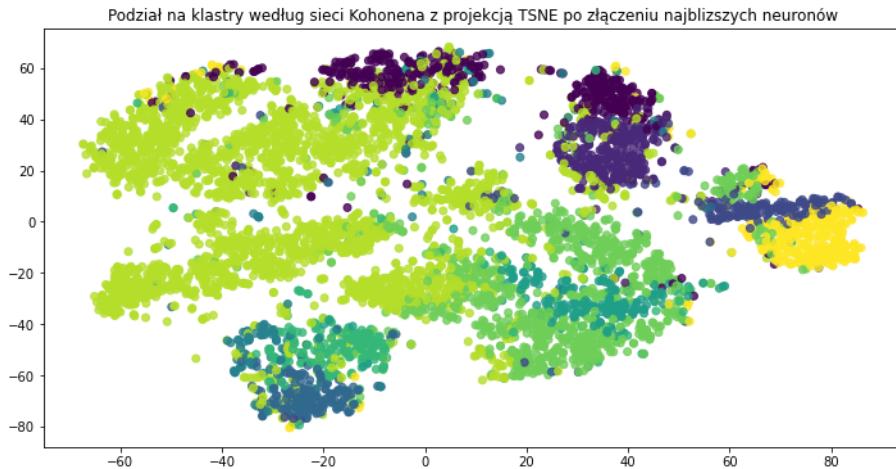
Rysunek 31: Klasteryzacja uzyskana przez sieć Kohonena

Czwartą i ostatnią architekturą testowaną na tym zbiorze danych była architektura z 16 neuronami oraz z siatką hexagonalną.

```
kohNet_mnist_gauss4 = Network(input_shape=784, shape=(4,4))
kohNet_mnist_gauss4.fit(mnist_df, 15, neighbourhood_scale=0.5,
                        grid_type='hex', tsne_data=tsne_data)
```



Rysunek 32: Prawdziwe klastry



Rysunek 33: Klasteryzacja uzyskana przez sieć Kohonena po złączeniu najbliższych neuronów

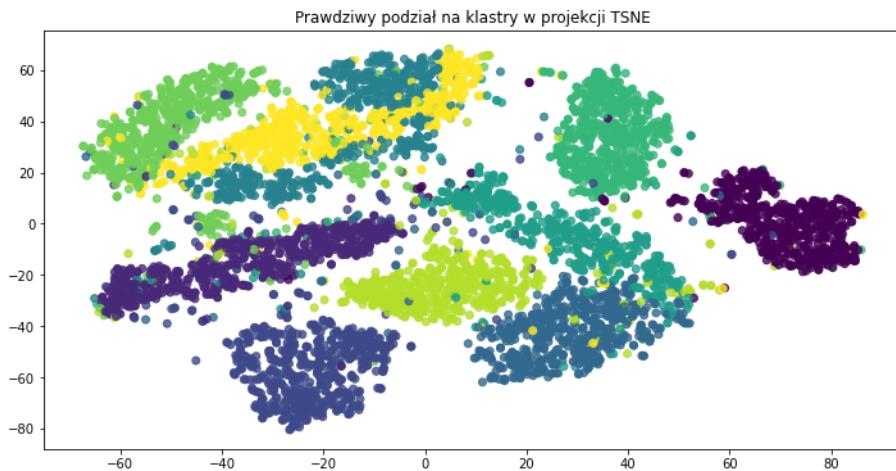
Wartości metryk dla tej sieci:

- Calinski-Harabasz metric: 150.61446524676555
- Silhouette metric: 0.06220018545879935
- Rand Score: 0.7326134911374118
- Adjusted Rand Score: 0.15930780994637872
- V-measure: 0.37723677561838964

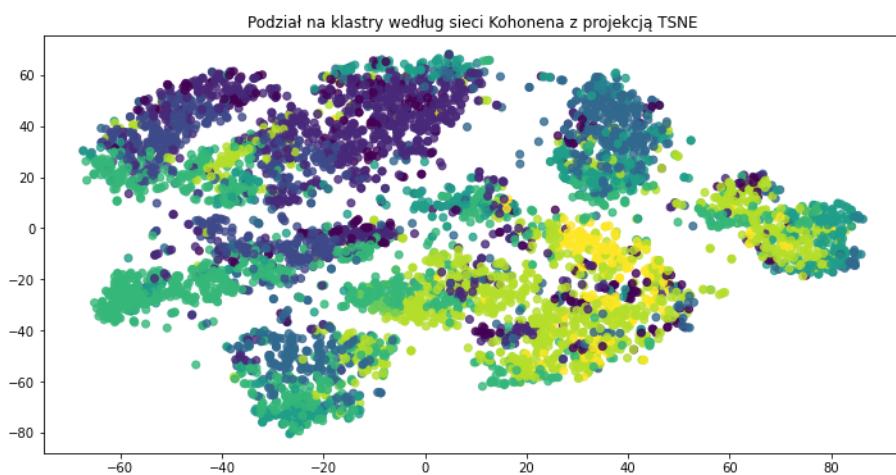
4.1.2 Funkcja Meksykański kapelusz

Pierwsza przetestowana architektura z siatką prostokątną oraz 10 neuronami:

```
kohNet_mnist_hat1 = Network(input_shape=784, shape=(10,1),
                             neighbourhood_func=gaussian_second_derivative)
kohNet_mnist_hat1.fit(mnist_df, 10, neighbourhood_scale=0.1,
                      grid_type='rectangle', tsne_data=tsne_data)
```



Rysunek 34: Prawdziwe klastry



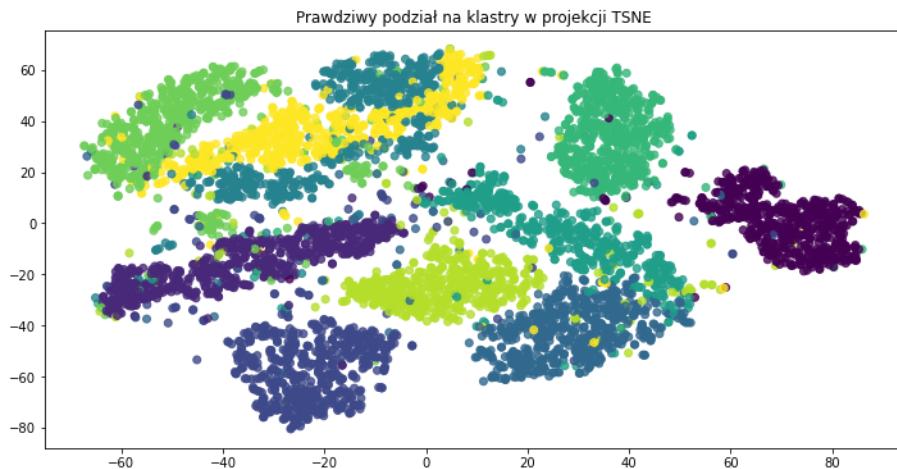
Rysunek 35: Klasteryzacja uzyskana przez sieć Kohonena

Wartości metryk dla tej sieci:

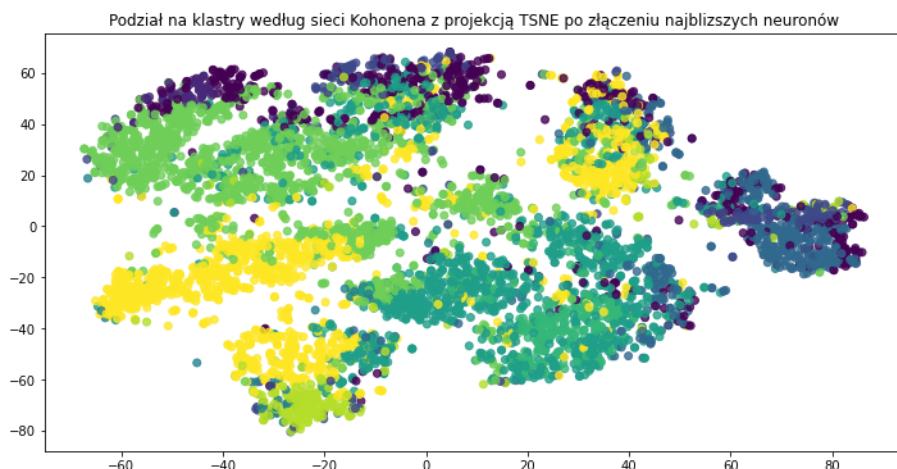
- Calinski-Harabasz metric: 107.01908969253245
- Silhouette metric: -0.011568978087454067
- Rand Score: 0.807258543090515
- Adjusted Rand Score: 0.13573715661681643
- V-measure: 0.2372190448538988

Druga architektura z 16 neuronami oraz siatką prostokątną:

```
kohNet_mnist_hat2 = Network(input_shape=784, shape=(4,4),  
    neighbourhood_func=gaussian_second_derivative)  
kohNet_mnist_hat2.fit(mnist_df, 7, neighbourhood_scale=0.15,  
    grid_type='rectangle', tsne_data=tsne_data)
```



Rysunek 36: Prawdziwe klastry



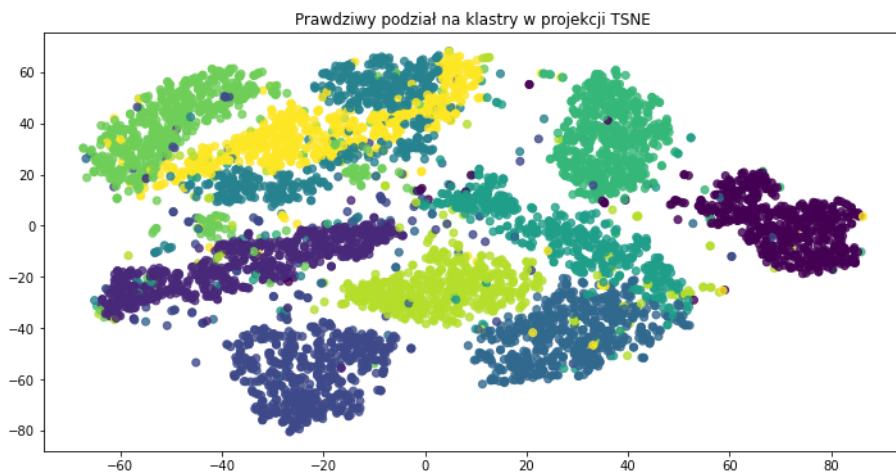
Rysunek 37: Klasteryzacja uzyskana przez sieć Kohonena po złączeniu najbliższych neuronów

Wartości metryk dla tej sieci:

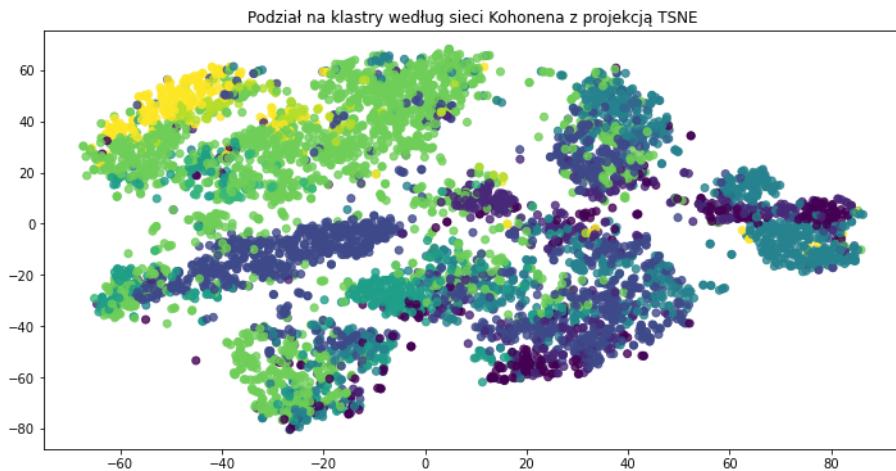
- Calinski-Harabasz metric: 94.05760209553644
- Silhouette metric: 0.005125564709326434
- Rand Score: 0.8009032060899038
- Adjusted Rand Score: 0.16783074393982067
- V-measure: 0.29587121257637566

Trzecia architektura przetestowana na tym zbiorze danych była architektura o hexagonalnej siatce oraz 10 neuronach.

```
kohNet_mnist_hat3 = Network(input_shape=784, shape=(10,1),  
    neighbourhood_func=gaussian_second_derivative)  
kohNet_mnist_hat3.fit(mnist_df, 10, neighbourhood_scale=0.1,  
    grid_type='hex', tsne_data=tsne_data)
```



Rysunek 38: Prawdziwe klastry



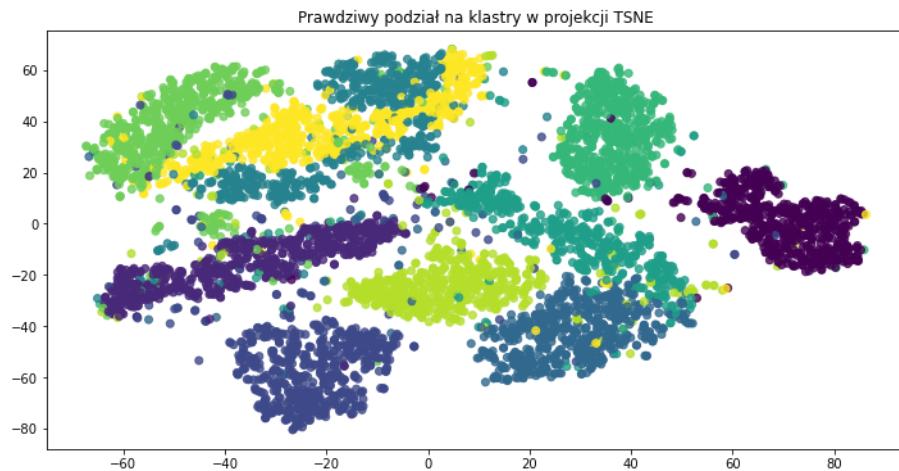
Rysunek 39: Klasteryzacja uzyskana przez sieć Kohonena

Wartości metryk dla tej sieci:

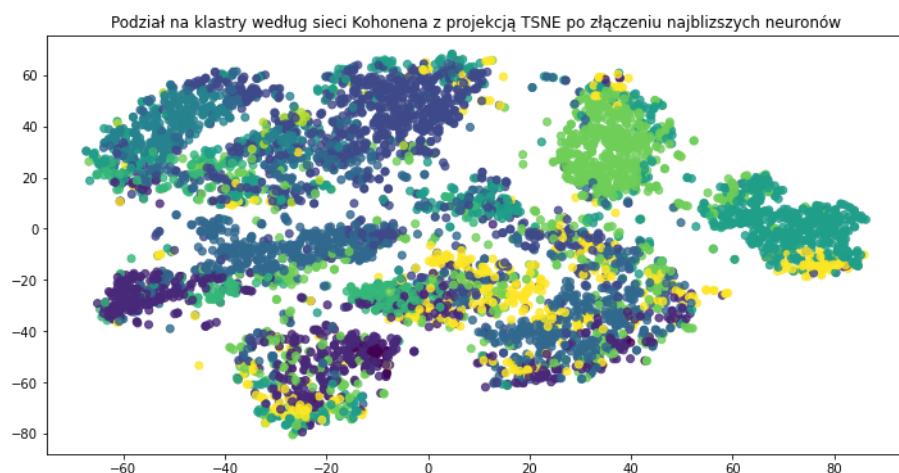
- Calinski-Harabasz metric: 109.44139113792758
- Silhouette metric: -0.018038663987285593
- Rand Score: 0.775513641162416
- Adjusted Rand Score: 0.1441009086466851
- V-measure: 0.2722937825109048

Czwartą i ostatnią architekturą z tą funkcją sąsiedztwa, była sieć z hexagonalną siatką oraz 16 neuronami.

```
kohNet_mnist_hat4 = Network(input_shape=784, shape=(4,4),
                             neighbourhood_func=gaussian_second_derivative)
kohNet_mnist_hat4.fit(mnist_df, 7, neighbourhood_scale=0.15,
                      grid_type='hex', tsne_data=tsne_data)
```



Rysunek 40: Prawdziwe klastry



Rysunek 41: Klasteryzacja uzyskana przez sieć Kohonena po złączeniu najbliższych neuronów

Wartości metryk dla tej sieci:

- Calinski-Harabasz metric: 80.18440999575162
- Silhouette metric: -0.019967305693189063
- Rand Score: 0.8271893093293327
- Adjusted Rand Score: 0.1941634408897761
- V-measure: 0.2967066055411918

4.2 Zbiór danych Smartphone Activity

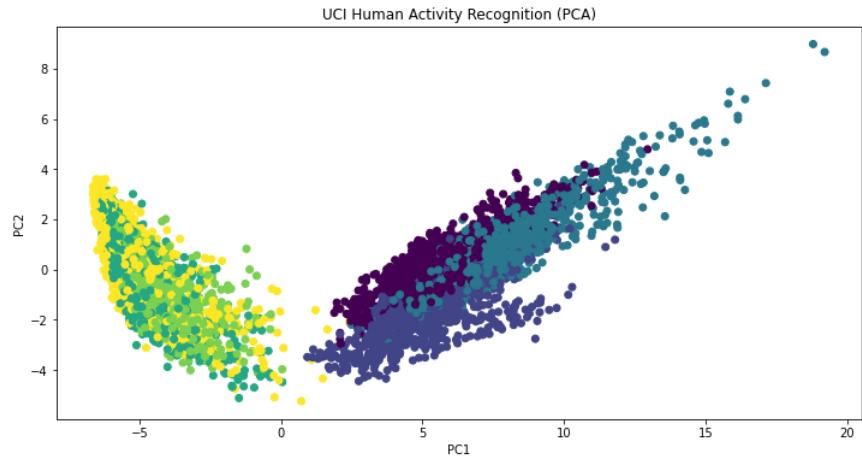
Zbiór danych Smartphone Activity zawiera odczyty z żyroskopu oraz akcelerometru z smartphone'a.

Dane są tam podane w zależności od czasu oraz przykształcane na dziedzinę częstotliwości za pomocą szybkiej transformaty Fouriera. Zbiór jest również wzbogacony o dodatkowe parametry co daje nam łącznie 561 kolumn.

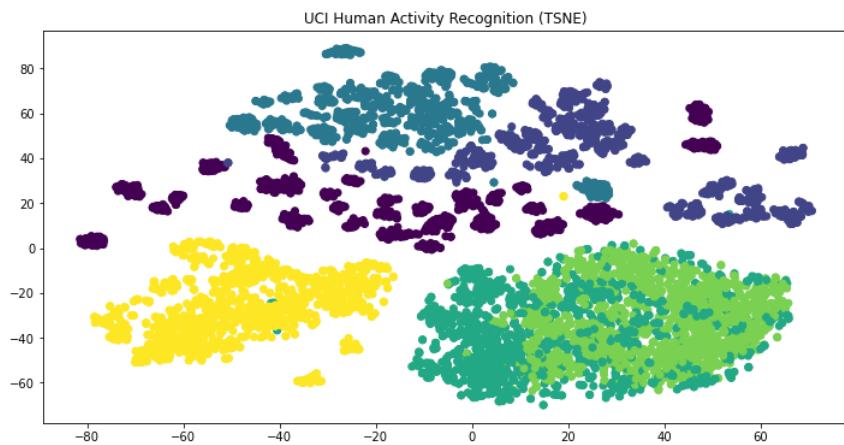
Etykiety dla zbioru treningowego oznaczają fizyczną aktywność użytkownika na podstawie odczytów z telefonu:

1. Walking (chodzenie)
2. Walking upstairs (wchodzenie po schodach do góry)
3. Walking downstairs (schodzenie po schodach w dół)
4. Sitting (siedzenie)
5. Standing (stanie)
6. Laying (leżenie)

Wizualizacja zbioru danych w projekcji PCA:



Rysunek 42: Wizualizacja zbioru danych z projekcją PCA



Rysunek 43: Wizualizacja zbioru danych z projekcją TSNE

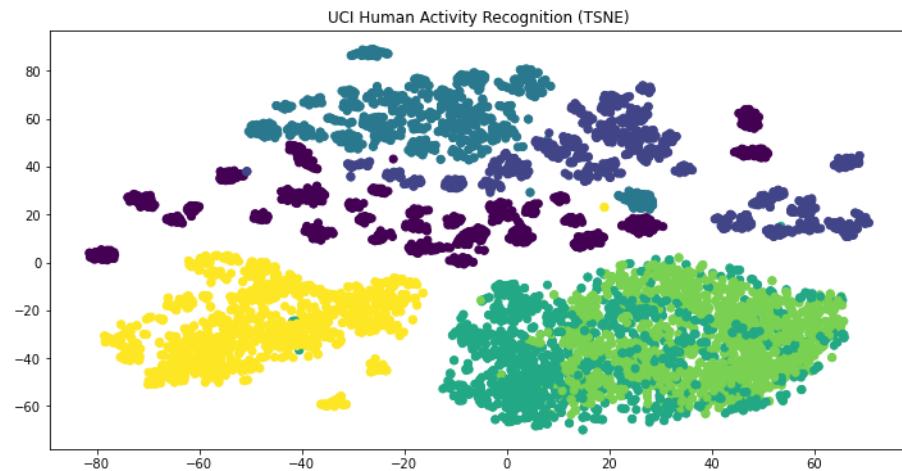
Podobnie jak na wcześniejszym zbiorze danych, tutaj także zbudowano łącznie 8 modeli (rozważano dwie siatki: hexagonalną oraz prostokątną, różną liczbę neuronów: (3,4) oraz (6,1), różne funkcje sąsiedztwa: funkcję Gaussa oraz Meksykańskiego kapelusza).

4.2.1 Funkcja Gaussa

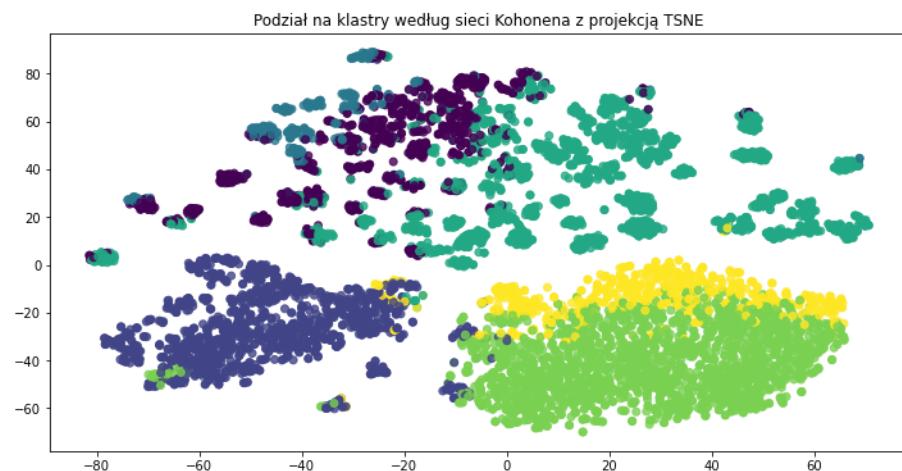
Pierwszy model (siatka prostokątna, 6 neuronów)

```
kohNet_ucihar_gauss1 = Network(input_shape=561, shape=(6,1),
                                neighbourhood_func=gaussian)
```

```
kohNet_ucihaar_gauss1.fit(ucihaar, 10, neighbourhood_scale=0.8,  
grid_type='rectangle', tsne_data=tsne_data2)
```



Rysunek 44: Prawdziwe klastry



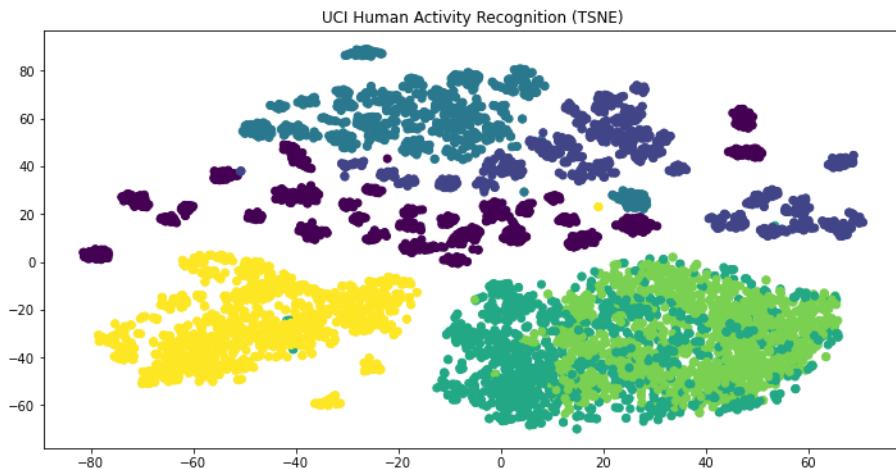
Rysunek 45: Klasteryzacja uzyskana przez sieć Kohonena

Wartości metryk dla tej sieci:

- Calinski-Harabasz metric: 3016.955028038348
- Silhouette metric: 0.1502611454236343
- Rand Score: 0.8402194915039725
- Adjusted Rand Score: 0.49142684137064924
- V-measure: 0.6111624684310624

Drugi model (siatka prostokątna, 12 neuronów)

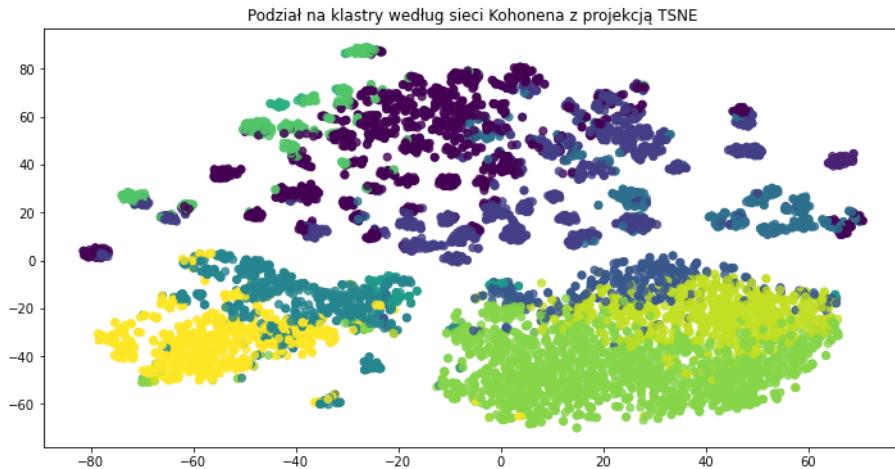
```
kohNet_ucihar_gauss2 = Network(input_shape=561, shape=(3,4),  
                                neighbourhood_func=gaussian)  
kohNet_ucihar_gauss2.fit(ucihar, 10, neighbourhood_scale=0.8,  
                        grid_type='rectangle', tsne_data=tsne_data2)
```



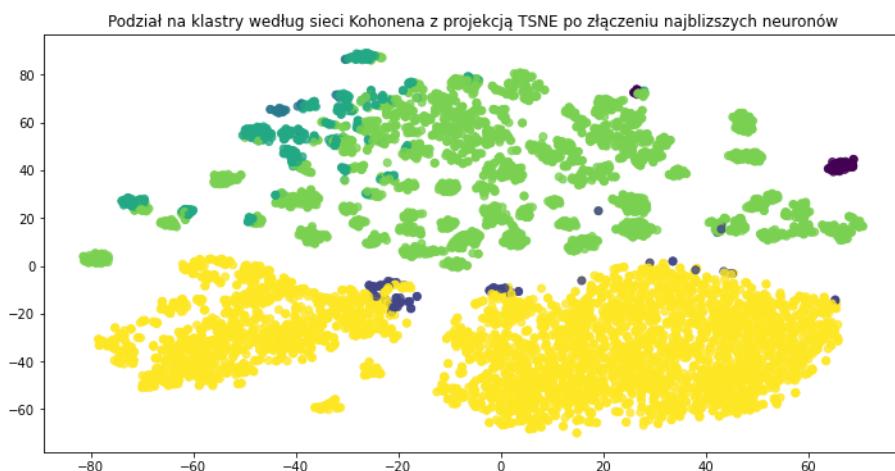
Rysunek 46: Prawdziwe klastry

Wartości metryk dla tej sieci:

- Calinski-Harabasz metric: 1604.4778997203168
- Silhouette metric: 0.08989094752303135
- Rand Score: 0.6870905322704868
- Adjusted Rand Score: 0.32439045649655346
- V-measure: 0.5255466216387522



Rysunek 47: Klasteryzacja uzyskana przez sieć Kohonena bezpośrednio po procesie uczenia

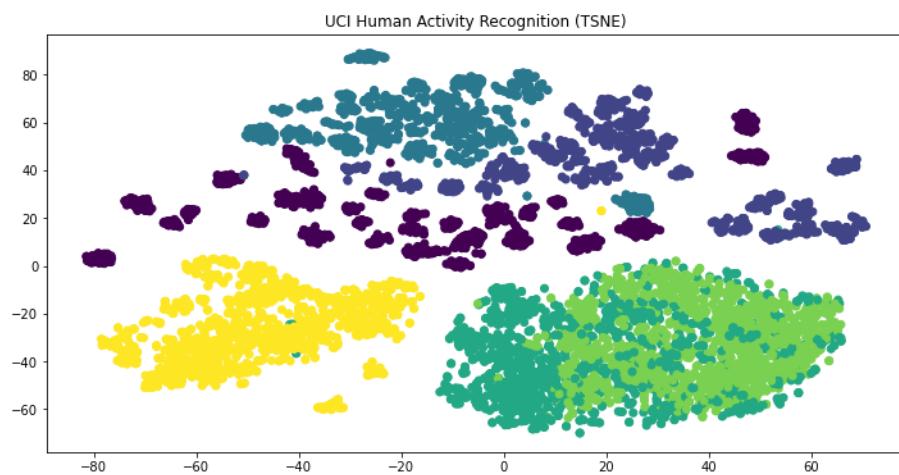


Rysunek 48: Klasteryzacja uzyskana przez sieć Kohonena po złączeniu najbliższych neuronów

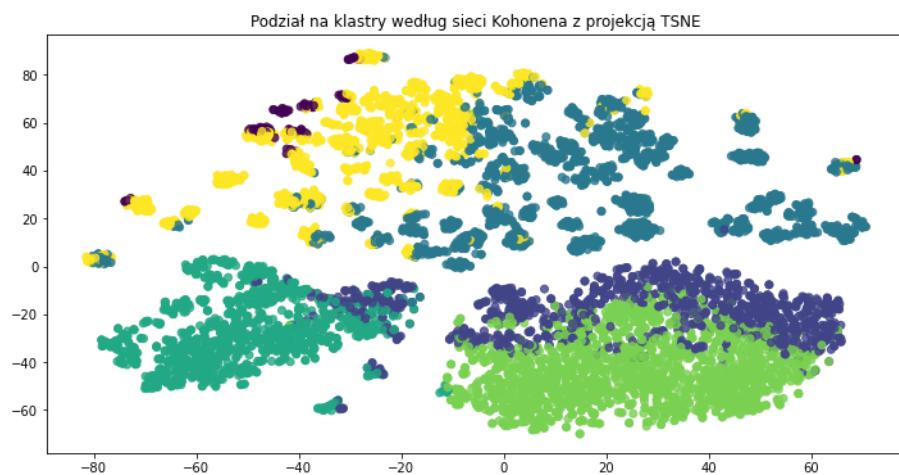
Zauważmy, że klasteryzacja uzyskana przez sieć Kohonena bezpośrednio po jej uczeniu jest dość poprawna. Następnie w celu policzenia adjusted rand score i pozostałych metryk nadzorowanych byłem zmuszony połączyć sąsiednie neurony, co było robione dla każdej większej, wcześniej wspomnianej architektury. Na skutek tego otrzymane klastrowanie nie jest już tak poprawne (zauważmy jak dużo obserwacji zostało przypisanych do klastra żółtego)

Następną, trzecią testowaną architekturą była architektura z hexagonalnym układem siatki oraz z 6 neuronami.

```
kohNet_ucihar_gauss3 = Network(input_shape=561, shape=(6,1),  
                                 neighbourhood_func=gaussian)  
kohNet_ucihar_gauss3.fit(ucihar, 10, neighbourhood_scale=0.8,  
                        grid_type='hex', tsne_data=tsne_data2)
```



Rysunek 49: Prawdziwe klastry



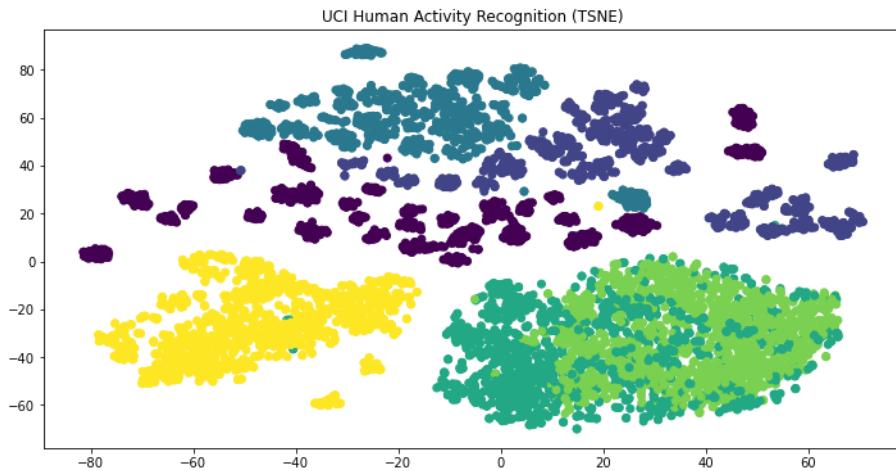
Rysunek 50: Klasteryzacja uzyskana przez sieć Kohonena

Wartości metryk dla tej sieci:

- Calinski-Harabasz metric: 3016.0297125085544
- Silhouette metric: 0.14817961177424083
- Rand Score: 0.8328827667958095
- Adjusted Rand Score: 0.45301400142688397
- V-measure: 0.5987767706227174

Ostatnią siecią z Gaussowską funkcją sąsiedztwa była architektura z 12 neuronami oraz hexagonalnym układem siatki.

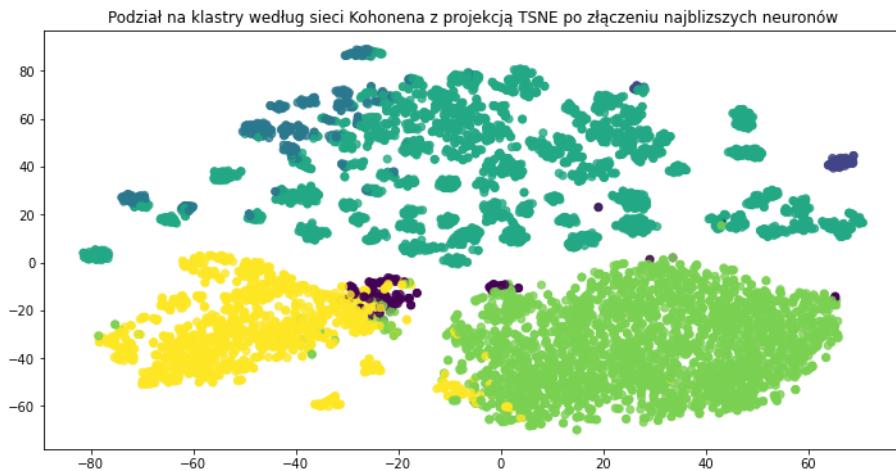
```
kohNet_ucihaar_gauss4 = Network(input_shape=561, shape=(3,4),  
                                 neighbourhood_func=gaussian)  
kohNet_ucihaar_gauss4.fit(ucihaar, 10, neighbourhood_scale=0.8,  
                         grid_type='hex', tsne_data=tsne_data2)
```



Rysunek 51: Prawdziwe klastry

Wartości metryk dla tej sieci:

- Calinski-Harabasz metric: 1629.7631614429647
- Silhouette metric: 0.09619072385551788
- Rand Score: 0.7223638749008411
- Adjusted Rand Score: 0.3355509755501063
- V-measure: 0.524794863220041

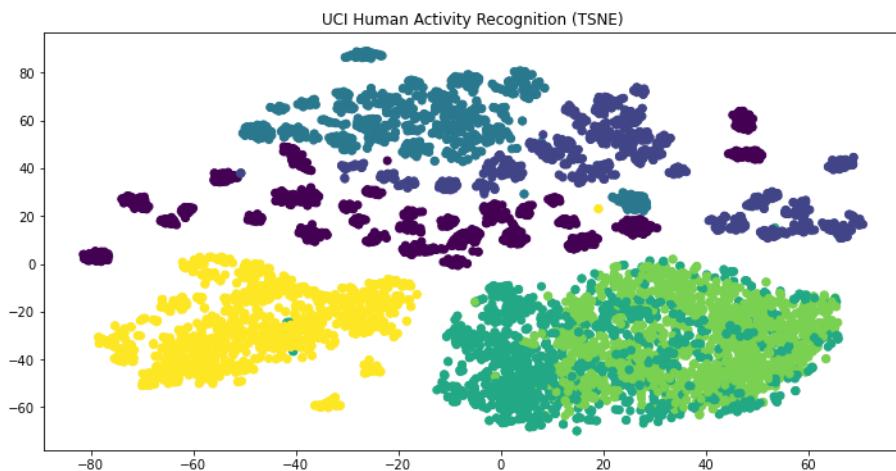


Rysunek 52: Klasteryzacja uzyskana przez sieć Kohonena po złączeniu najbliższych neuronów

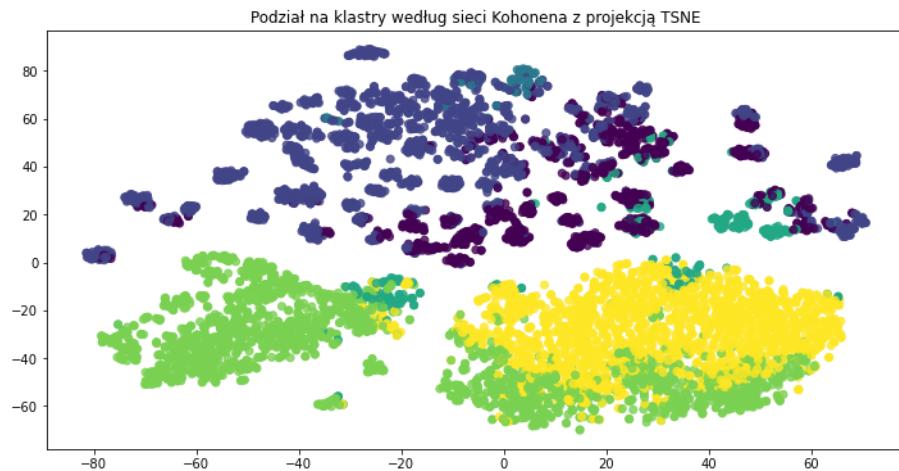
4.2.2 Funkcja Meksykański kapelusz

Pierwsza architektura (prostokątna siatka, 6 neuronów)

```
kohNet_ucihaar_hat1 = Network(input_shape=561, shape=(6,1),
                               neighbourhood_func=gaussian_second_derivative)
kohNet_ucihaar_hat1 . fit(ucihaar , 10, neighbourhood_scale=0.15,
                          grid_type='rectangle' , tsne_data=tsne_data2)
```



Rysunek 53: Prawdziwe klastry



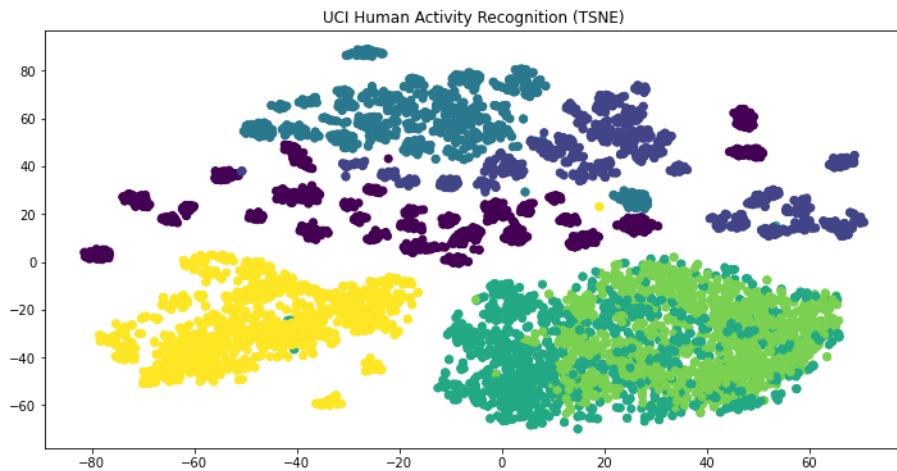
Rysunek 54: Klasteryzacja uzyskana przez sieć Kohonena

Wartości metryk dla tej sieci:

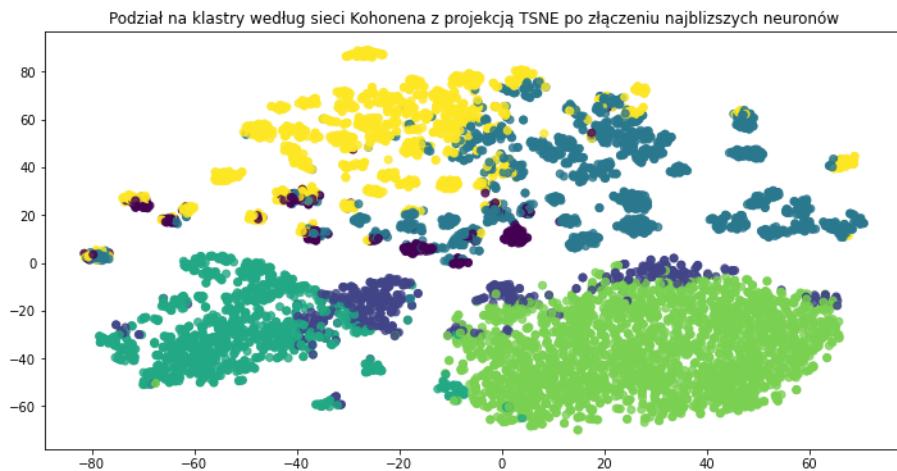
- Calinski-Harabasz metric: 2587.942115424464
- Silhouette metric: 0.0782240774163828
- Rand Score: 0.7986646646640719
- Adjusted Rand Score: 0.3795216482389627
- V-measure: 0.514027862416616

Druga architektura (prostokątna siatka, 12 neuronów)

```
kohNet_ucihar_hat2 = Network(input_shape=561, shape=(3,4),  
    neighbourhood_func=gaussian_second_derivative)  
kohNet_ucihar_hat2.fit(ucihar, 10, neighbourhood_scale=0.15,  
    grid_type='rectangle', tsne_data=tsne_data2)
```



Rysunek 55: Prawdziwe klastry



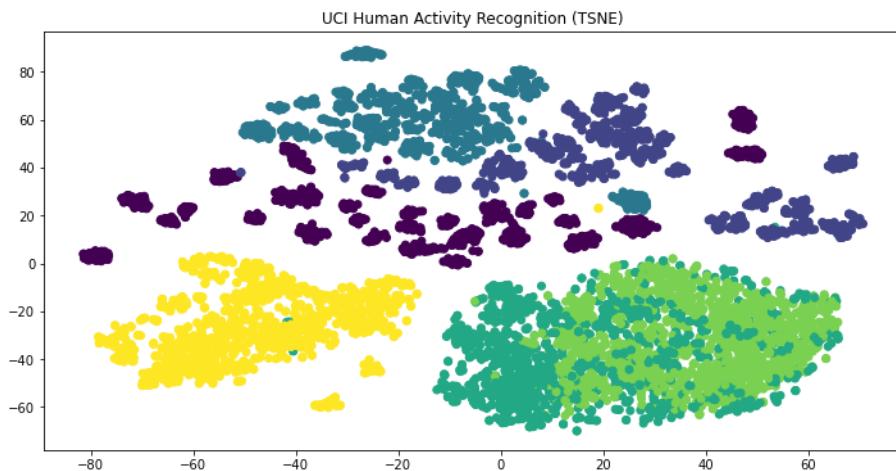
Rysunek 56: Klasteryzacja uzyskana przez sieć Kohonena po złączeniu najbliższych neuronów

Wartości metryk dla tej sieci:

- Calinski-Harabasz metric: 1481.3471971922702
- Silhouette metric: 0.028312669563982377
- Rand Score: 0.8371640864004202
- Adjusted Rand Score: 0.3795216482389627
- V-measure: 0.514027862416616

Trzecia architektura (hexagonalna siatka, 6 neuronów)

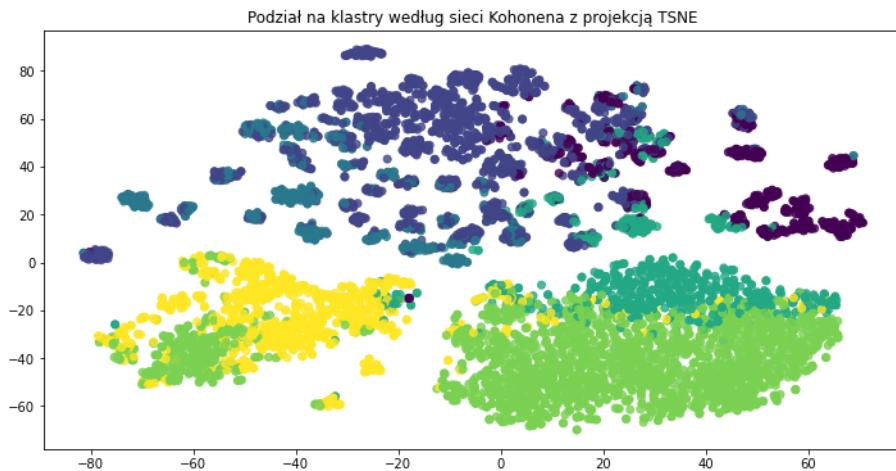
```
kohNet_ucihar_hat3 = Network(input_shape=561, shape=(6,1),  
                           neighbourhood_func=gaussian_second_derivative)  
kohNet_ucihar_hat3.fit(ucihar, 10, neighbourhood_scale=0.15,  
                       grid_type='hex', tsne_data=tsne_data2)
```



Rysunek 57: Prawdziwe klastry

Wartości metryk dla tej sieci:

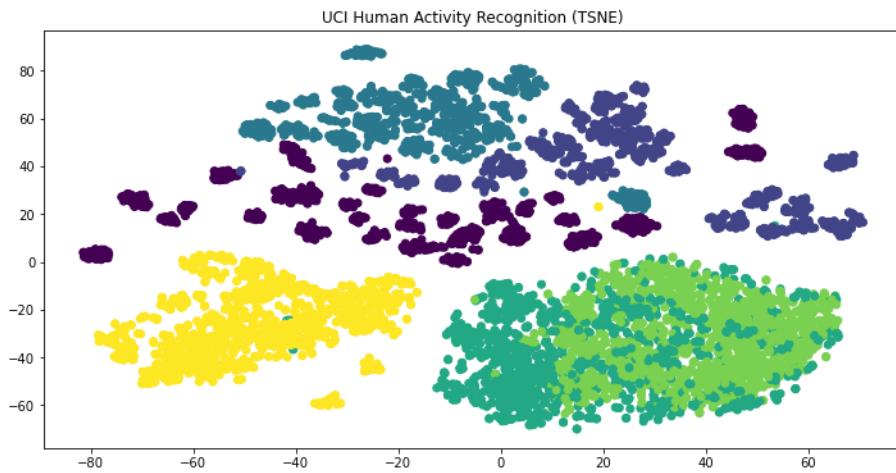
- Calinski-Harabasz metric: 2517.373831319886
- Silhouette metric: 0.10353441744980971
- Rand Score: 0.8001690161110041
- Adjusted Rand Score: 0.36033476968484685
- V-measure: 0.501510665057926



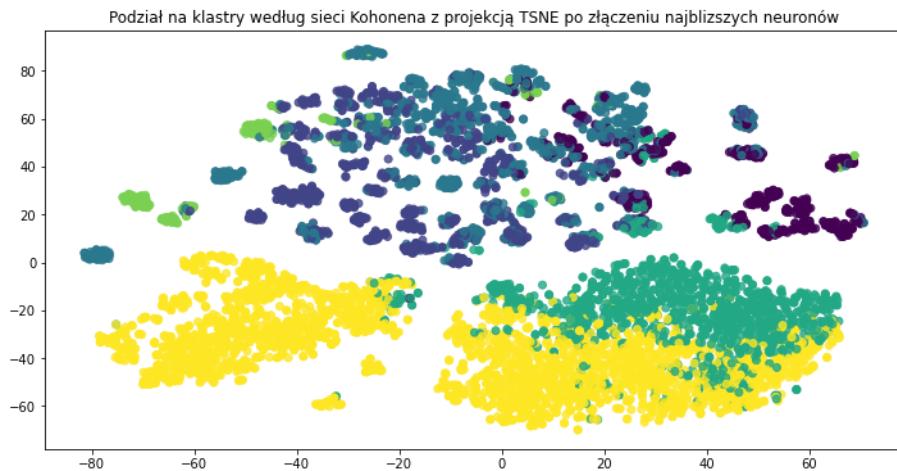
Rysunek 58: Klasteryzacja uzyskana przez sieć Kohoneną

Czwarta i ostatnia architektura (hexagonalna siatka, 12 neuronów)

```
kohNet_ucihaar_hat4 = Network(input_shape=561, shape=(3,4),
                                neighbourhood_func=gaussian_second_derivative)
kohNet_ucihaar_hat4.fit(ucihaar, 10, neighbourhood_scale=0.15,
                        grid_type='hex', tsne_data=tsne_data2)
```



Rysunek 59: Prawdziwe klastry



Rysunek 60: Klasteryzacja uzyskana przez sieć Kohonena po złączeniu najbliższych neuronów

Wartości metryk dla tej sieci:

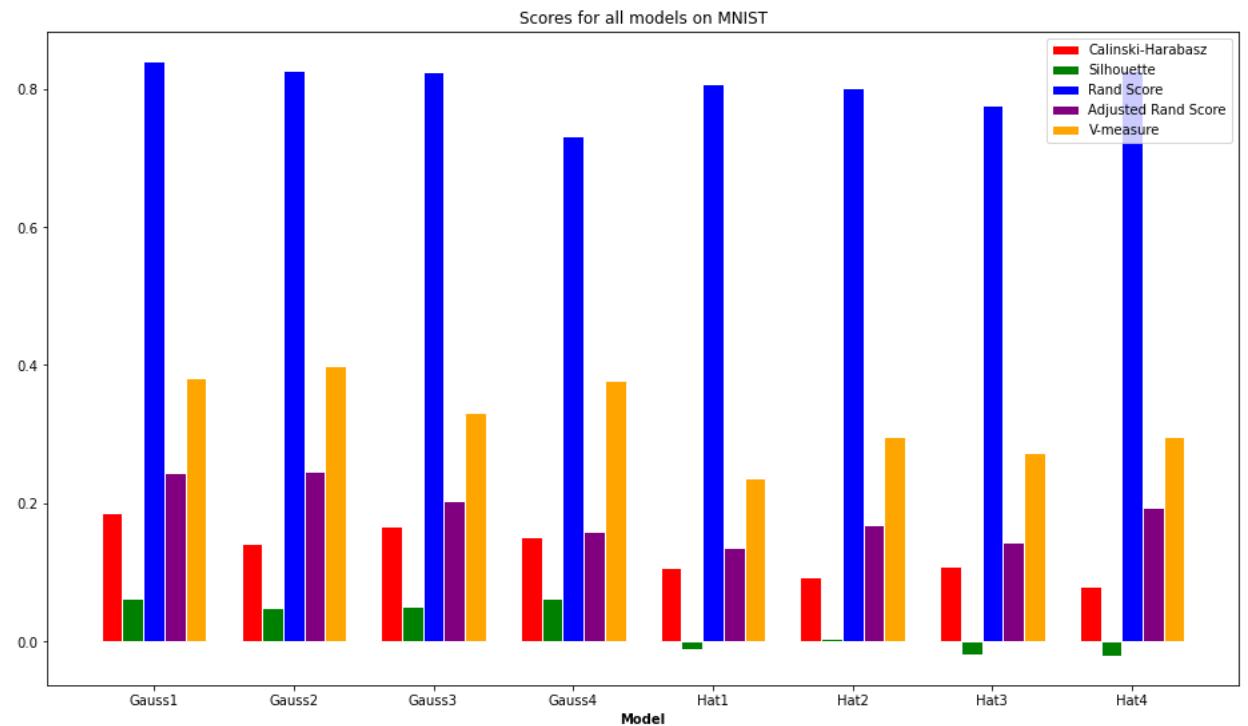
- Calinski-Harabasz metric: 1290.7080667341195
- Silhouette metric: 0.037329049070281915
- Rand Score: 0.7738459188263787
- Adjusted Rand Score: 0.3177746704360671
- V-measure: 0.4610069762111457

4.3 Podsumowanie

Na sam koniec podsumowano wszystkie testy w ramach tej pracy domowej. W tym celu stworzono wizualizację przedstawiającą graficznie wartości wszystkich metryk dla każdego modelu.

UWAGA metryka Calinski-Harabasz została przeskalowana przez 1000, aby można było ją przedstawić na tym samym wykresie z resztą metryk, które w porównaniu do niej przyjmują wartości z zakresu (-1,1)

4.3.1 MNIST



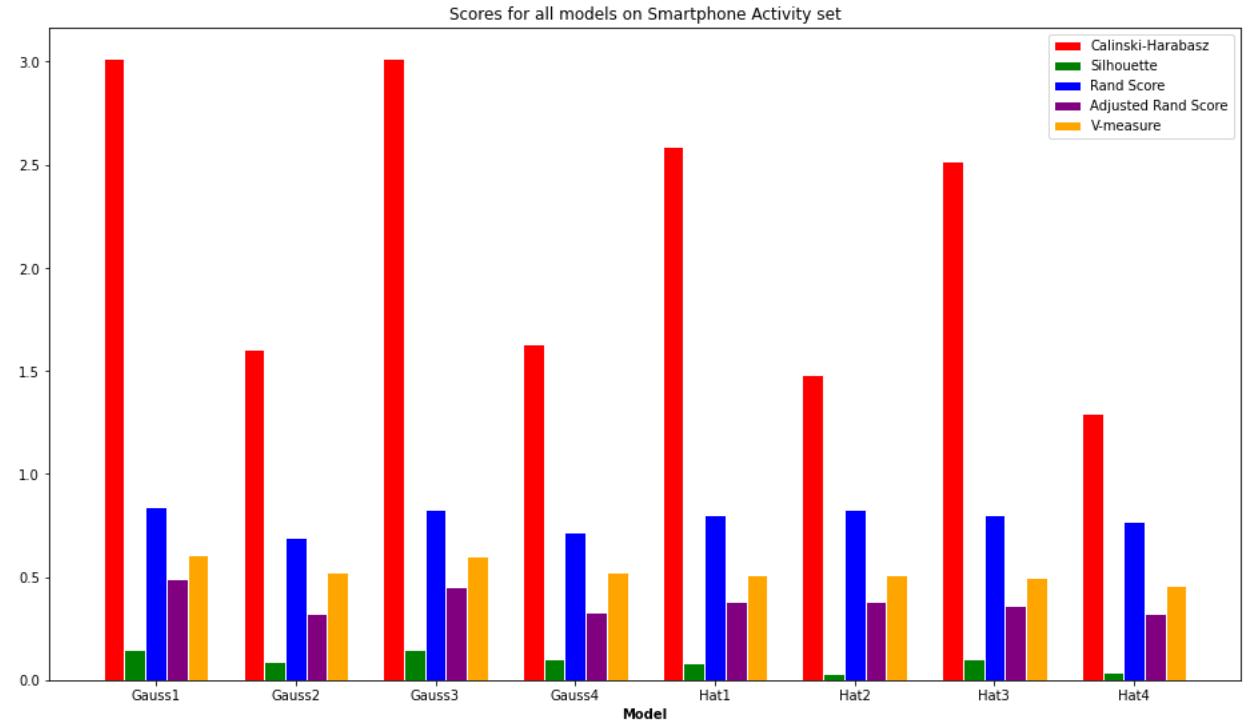
Rysunek 61: Podsumowanie wyników uzyskanych na zbiorze MNIST

Patrząc na zestawione wyniki ciężko jednoznacznie wskazać najlepszy model. Dwa pierwsze modele uzyskiwały raczej lepsze wartości metryk. Różnice te są praktycznie niezauważalne na wykresie, dopiero po analizie dokładnych wartości, jakie udało się uzyskać, które zostały przedstawione w wcześniejszej części sprawozdania jesteśmy w stanie stwierdzić, że pierwsze dwie architektury radzą sobie nieznacznie lepiej.

Warto tylko zauważyć, że dla funkcji sąsiedztwa typu Meksykański kapelusz otrzymywaliśmy zazwyczaj gorsze wartości wszystkich wymienionych metryk. Za-

tem niezależnie od wyboru ustawienia siatki oraz liczby neuronów lepsze rezultaty uzyskano dla Gaussowskiej funkcji sąsiedztwa.

4.3.2 Smartphone activity



Rysunek 62: Podsumowanie wyników uzyskanych na zbiorze Smartphone activity

Najlepszym modelem na tym zbiorze danych był model pierwszy tj. z 6 neuronami, funkcją Gaussowską oraz siatką prostokątną. Wynik zgadza się z charakterystyką zbioru danych, gdyż ten zbiór zawiera dokładnie 6 klas, zatem oczekiwaliśmy uzyskać lepszy score właśnie dla takiej architektury.

4.3.3 Znalezienie najbardziej optymalnej liczby klastrów

Na sam koniec przy użyciu metryk uczenia nienadzorowanego (Silhouette score oraz Calinski-Harabasz score) znaleziono najbardziej optymalną liczbę klastrów na obydwu zbiorach danych. W tym celu rozważono różne wymiary siatki oraz jej typ: prostokątna oraz hexagonalna, w celu znalezienia modelu maksymalizującego wspomniane metryki. Funkcja użyta do przeprowadzenia tych testów wyglądała następująco:

```

def cross_validate_network(params, fit_params, neurons):
    metrics = [silhouette_score, calinski_harabasz_score]
    scores = pd.DataFrame(columns=['Num_clusters',
                                    'silhouette_score',
                                    'calinski_harabasz_score'])
    for fa in fit_params:
        for neu in neurons:
            n = Network(**params, shape=neu)
            n.fit(**fa)
            temp=[]
            for metric in metrics:
                score = metric(n.data, n.assign_data_to_clusters(n.data))
                temp.append(score)
            scores = scores.append({ 'Num_clusters':n.shape[0]*n.shape[1],
                                     'silhouette_score':temp[0],
                                     'calinski_harabasz_score':temp[1]}, ignore_index=True)
    return scores

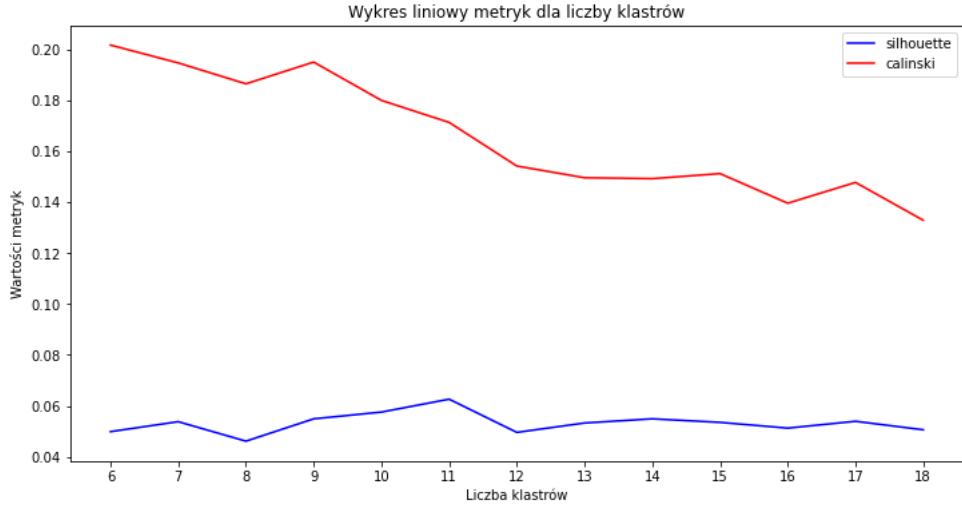
```

Jeśli chodzi o zbiór **MNIST** rozważono następujące kombinacje:

```

params_mnist = { 'input_shape':784, 'neighbourhood_func':gaussian}
fit_params_mnist = [
{'all_data':mnist_df, 'n_epochs':5, 'neighbourhood_scale':0.8,
'grid_type':'rectangle', 'print_results':False, 'tsne_data':tsne_data},
{'all_data':mnist_df, 'n_epochs':5, 'neighbourhood_scale':0.8,
'grid_type':'hex', 'print_results':False, 'tsne_data':tsne_data}
]
neurons_mnist = [(2,3),(7,1),(2,4),(9,1),(2,5),(11,1),(3,4),(13,1),(2,7),
(3,5),(4,4),(17,1),(3,6)]

```



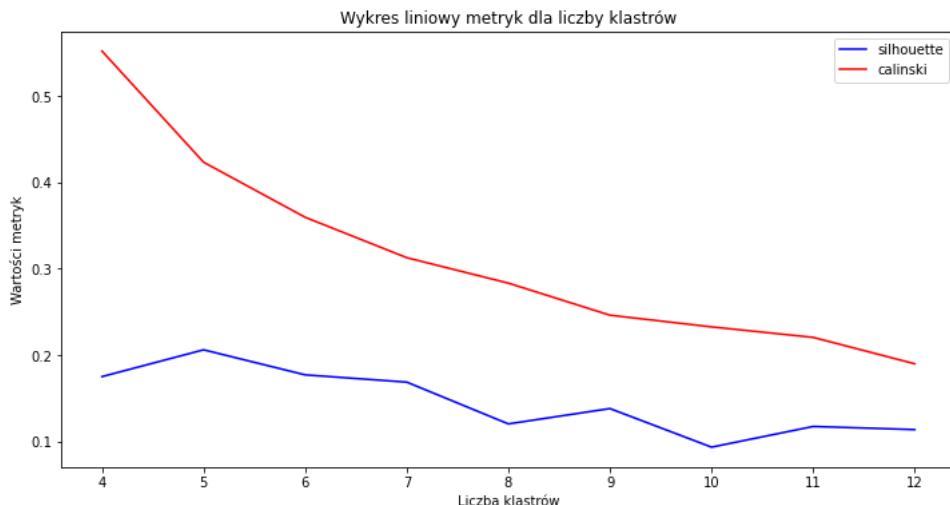
Rysunek 63: Wartości metryk w zależności od liczby klastrów dla zbioru **MNIST**. Wartość metryki Calinski-Harabasz została przeskalowana przez **1000**.

Patrząc na metrykę Silhouette widzimy wyraźny pik dla 11 klastrów. Wartość Calinski-Harabasz dla tej ilości również jest na wysokim poziomie. Wybierając liczbę klastrów bez żadnej wiedzy na temat ile faktycznych klastrów znajduje się w tym zbiorze danych, możnaby przyjąć, że jest ich właśnie jedenaście. W rzeczywistości wiemy, że wspomniany zbiór danych zawiera 10 klas zatem uzyskany przez nas wynik nieznacznie się różni od tego oczekiwanej.

Jeśli natomiast chodzi o zbiór **Smartphone activity** rozważono następujące kombinacje:

```
params_ucihar = { 'input_shape':561, 'neighbourhood_func':gaussian}
fit_params_ucihar = [
    { 'all_data':ucihar, 'n_epochs':5,
      'neighbourhood_scale':0.8, 'grid_type':'rectangle',
      'print_results':False, 'tsne_data':tsne_data2},
    { 'all_data':ucihar, 'n_epochs':5,
      'neighbourhood_scale':0.8, 'grid_type':'hex',
      'print_results':False, 'tsne_data':tsne_data2}
]
neurons_ucihar = [(2,2),(5,1),(6,1),(7,1),(2,4),(9,1),(2,5),(11,1),(3,4)]
```

Patrząc na wartości metryk dla różnej ilości klastrów widzimy wyraźne maksimum jeśli chodzi o metrykę Silhouette dla 5 klastrów. Wartość drugiej metryki jest również bardzo wysoka dla tej takiej liczby klastrów. Zatem nie mając informacji na temat faktycznej liczby klas, najbardziej optymalnym wyborem



Rysunek 64: Wartości metryk w zależności od liczby klastrów dla zbioru **Smart-phone activity**. Wartość metryki Calinski-Harabasz została przeskalowana przez **8000**.

wydaje się przyjęcie, że jest ich właśnie pięć. W rzeczywistości zbiór danych posiada 6 różnych klas, zatem otrzymany wynik nieznacznie się różni od tego oczekiwanej.

5 Wnioski oraz podsumowanie

W ramach tego projektu została zaimplementowana sieć Kohonena z możliwością wyboru wymiarów siatki, rodzaju siatki, funkcji sąsiedztwa oraz skali sąsiedztwa. Na podstawie przeprowadzonych testów lepsze wyniki otrzymywaliśmy dla Gaussowskiej funkcji sąsiedztwa. Dodatkowo najlepsze uzyskane rezultaty uzyskaliśmy dla skali sąsiedztwa pomiędzy 0.6 a 1.0. Jeśli chodzi o różnicę w wydajności dla siatki hexagonalnej oraz prostokątnej różnice są raczej niezauważalne. Najgorsze wyniki otrzymaliśmy na zbiorze MNIST, który przedstawiał zdjęcia odrečcznie napisanych cyfr. Może tak się dziać z uwagi na skomplikowaną strukturę tego zbioru danych oraz słabą separację pomiędzy klasami.

Literatura

- [1] Strona wykładu. <http://pages.mini.pw.edu.pl/karwowskij/metody-inteligencji-obliczeniowej-w-analizie-danych>.
- [2] Towards data science. <https://towardsdatascience.com/kohonen-self-organizing-maps-a29040d688da>.