Deep Graph Convolutional Neural Network - dokumentacja końcowa

Maksim Makaranka, Michał Olejnik

1 Temat

Celem projektu jest przetestowanie architektury *Deep Graph Convolutional Neural Network*[1] poprzez zreplikowanie wyników uzyskanych przez autorów artykułu oraz porównanie uzyskanych wyników z wynikami z artykułu oraz wynikami innych, podobnych architektur. Projekt obejmuje również omówienie otrzymanych wyników.

2 Opis problemu

Sieci neuronowe zyskały na znaczeniu w różnych dziedzinach, takich jak klasyfikacja obrazów, przetwarzanie języka naturalnego, uczenie ze wzmocnieniem i analiza szeregów czasowych. Struktura połączeń między warstwami sprawia, że są one odpowiednie do przetwarzania sygnałów w formie tensorów, gdzie elementy są ułożone w znaczącej kolejności. Ta stała kolejność wejściowa jest kluczowa dla ekstrakcji cech wyższego poziomu przez sieci neuronowe.

Jednakże, istnieje kategoria danych strukturalnych, jakimi są grafy, które zazwyczaj nie posiadają tensorowej reprezentacji o ustalonej kolejności. Do danych takich można zakwalifikować struktury molekuł, grafy wiedzy, sieci biologiczne społeczne czy dokumenty tekstowe z zależnościami. Brak takiej reprezentacji ogranicza zastosowanie sieci neuronowych na grafach. W niniejszej pracy analizujemy specjalne struktury sieci neuronowych, które mogą akceptować grafy i uczyć się funkcji predykcyjnych.

3 Opis modelu

W odpowiedzi na rosnące zainteresowanie uogólnieniem sieci neuronowych na grafy, zaproponowana została architektura *DGCNN*, która może zachować więcej informacji wierzchołkowych i uczyć się z globalnej topologii grafu. Kluczową innowacją jest nowa warstwa *SortPooling*, która przyjmuje nieuporządkowane cechy wierzchołków z konwolucji grafowych i układa je w spójnej kolejności, umożliwiając tradycyjnym sieciom neuronowym odczytywanie wierzchołków w ustalonej kolejności i trenowanie na tej reprezentacji. Warstwa *SortPooling*, jako most między *warstwami konwolucji grafowej* a *tradycyjnymi warstwami konwolucyjnych sieci neuronowych*, integruje reprezentację grafu i uczenie się w jedną architekturę typu end-to-end.

3.1 Warstwy konwolucji grafowej

Warstwy konwolucji grafowej w modelu *DGCNN* służą do agregacji informacji o wierzchołkach w lokalnych sąsiedztwach, aby wydobyć informacje o lokalnych podstrukturach. Proces konwolucji na grafie składa się z czterech kroków: transformacji liniowej cech wierzchołków, propagacji informacji do sąsiednich wierzchołków, normalizacji cech w celu zachowania stałej skali oraz zastosowania nieliniowej funkcji aktywacji.

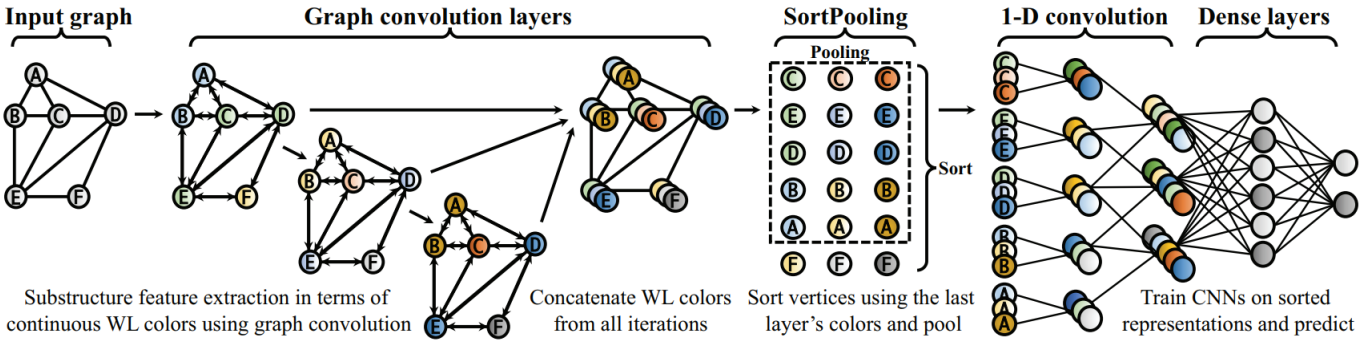
Aby wydobyć cechy podstruktur na wielu skalach, stosuje się wiele warstw konwolucji grafowej. Po przeprowadzeniu wielu warstw konwolucji, dodaje się warstwę do konkatenacji wyjść z każdej warstwy, tworząc "deskryptor cech" wierzchołka, który koduje informacje o jego lokalnej podstrukturze na wielu skalach.

Artykuł wskazuje, że taki sposób konwolucji grafowej jest podobny do *filtrów spektralnych* i może być postrzegany jako "miękka" wersja *algorytmu Weisfeilera-Lehmana*, co pomaga wyjaśnić jego wydajność w klasyfikacji na poziomie grafu. Ponadto, ostatnia warstwa konwolucji grafowej może być wykorzystana do sortowania wierzchołków grafu w spójnej kolejności na podstawie ich strukturalnych ról.

3.2 Warstwa SortPooling

Warstwa SortPooling służy do sortowania deskryptorów cech wierzchołków w spójnej kolejności, wykorzystując do tego kolory *WL (Weisfeilera-Lehmana)*, które są wynikiem etykietowania wierzchołków na podstawie ich topologii w grafie. Sortowanie wierzchołków odbywa się według najbardziej dopracowanych ciągłych kolorów *WL* z ostatniej warstwy konwolucji grafowej. Sortowanie to umożliwia sekwencyjne odczytywanie węzłów grafu przez sieci neuronowe i uczenie się stosownych modeli.

Kolejność wierzchołków oparta na kolorach *WL* z ostatniej warstwy jest ustalana poprzez sortowanie wierzchołków według ostatniego kanału w porządku malejącym. W przypadku, gdy dwa wierzchołki mają tę samą wartość w ostatnim kanale, rozstrzyganie odbywa się poprzez porównanie ich wartości w kolejnych kanałach.



Rysunek 1: Struktura *DGCNN*: Graf o dowolnej strukturze jest przetwarzany przez *warstwy konwolucji grafowej*, gdzie informacje są przekazywane między węzłami. Cechy wierzchołków są sortowane i grupowane *warstwą SortPooling*, a następnie przekazywane do struktur *CNN* w celu nauki modelu predykcyjnego.

Oprócz sortowania cech wierzchołków, warstwa *SortPooling* unifikuje również rozmiary tensorów wyjściowych, dostosowując je do zdefiniowanej przez użytkownika liczby k. Dzięki temu możliwe jest szkolenie sieci na grafach o różnej liczbie wierzchołków, które dzięki tej operacji uzyskują jednolity rozmiar.

Warstwa *SortPooling* pełni również funkcję mostu między *warstwami konwolucji grafowej* a *tradycyjnymi warstwami*, umożliwiając przekazywanie gradientów strat do poprzednich warstw, co jest kluczowe dla procesu uczenia się. Jest to znacząca przewaga nad metodami, które sortują wierzchołki w etapie preprocessingu, uniemożliwiając szkolenie parametrów przed sortowaniem.

3.3 Tradycyjne warstwy konwolucyjne

Po przeprowadzeniu operacji *SortPooling* otrzymywany jest tensor Z^{SP}, w którym każdy wiersz odpowiada wierzchołkowi, a każda kolumna kanałowi cechy. Następnie w ramach **tradycyjnych warstw konwolucyjnych**, tensor ten jest przekształcany w wektor. Następnie dodawana jest *jednowymiarowa warstwa konwolucyjna*, która sekwencyjnie stosuje filtry na deskryptorach cech wierzchołków. W dalszej kolejności, do modelu dodawane są *warstwy MaxPooling* oraz kolejne warstwy konwolucyjne, mające na celu naukę lokalnych wzorców na sekwencji węzłów. Na zakończenie procesu, do sieci dodawana jest *warstwa w pełni połączona*, po której następuje *warstwa Softmax*.

4 Sposób rozwiązania

W ramach projektu została wykorzystana implementacja *DGCNN* w PyTorch'u[2], udostępniona przez autorów artykułu. Repozytorium zawiera implementację sieci, zbiory danych rozszerzone ponad te wymienione w artykule oraz skrypty w MATLABie do transformacji grafów w formacie `.mat` oraz benchmarkowych grafowych datasetów udostępnionych przez Uniwersytet w Dortmundzie do formatu `.txt`.

Skrypt `run_DGCNN.sh` jest skryptem bash, który służy do uruchamiania modelu *DGCNN* na różnych zestawach danych. Skrypt ten został zmodyfikowany w celu optymalizacji i ułatwienia analizy wyników. Zamiast przyjmować argumenty wejściowe `fold` i `test_number`, skrypt ten teraz używa jedynego parametru `bsize`, który jest dostosowywany w zależności od wielkości zestawu danych, aby przyspieszyć wykonanie algorytmu. Dodatkowo, skrypt ten został zmodyfikowany, aby zapisywać ucięte wyjście do plików w folderze `outputs`, co umożliwia późniejszą analizę wyników w notebookach.

Poniżej znajdują się szczegółowe informacje na temat argumentów wejściowych i ustawień skryptu:

1. `DATA`: Nazwa zestawu danych do użycia. Domyślnie jest to "MUTAG", ale można go zmienić na jeden z następujących zestawów danych: MUTAG, ENZYMES, NCI1, NCI109, DD, PTC, PROTEINS, COLLAB, IMDBBINARY, IMDBMULTI.

2. `bsize`: Rozmiar batcha do użycia podczas uczenia. Jest dostosowywany w zależności od wielkości zestawu danych, aby przyspieszyć wykonanie algorytmu.

Skrypt ten zawiera również wiele ustawień ogólnych, takich jak `gm`, `gpu_or_cpu`, `GPU`, `CONV_SIZE`, `sortpooling_k`, `FP_LEN`, `n_hidden`, `dropout`, oraz przeniesionych z parametrów `fold` i `test_number`, które są parametrami modelu *DGCNN* i ustawieniami środowiska wykonania.

W zależności od wybranego zestawu danych, skrypt ten dostosowuje również specyficzne dla zestawu danych ustawienia, takie jak `num_epochs` i `learning_rate`.

Domyślnie `fold` jest ustawiony na 0, co powoduje, że skrypt ten wykonuje 10-krotną walidację krzyżową na wybranym zestawie danych. W przypadku ustawienia innej wartości, skrypt uruchamia model *DGCNN* na wybranym zestawie danych z określonym foldem jako danymi testowymi.

Po zakończeniu procesu uczenia, skrypt wyświetla wyniki dokładności dla wybranego zestawu danych oraz oblicza średnią dokładność i odchylenie standardowe z ostatnich 10 wyników dokładności. Wyniki te są zapisywane zarówno na konsoli, jak i do pliku w folderze `/outputs` dla późniejszej analizy. Dodatkowo, skrypt zapisuje również średnie straty i dokładność dla każdej epoki z każdego foldu, co umożliwia szczegółową analizę procesu uczenia.

Poniżej znajduje się krótki opis każdego z zestawów danych użytych w badaniach opisanych w artykule:

- **MUTAG**: Zestaw danych MUTAG składa się z 188 chemicznych związków, które są podzielone na dwie klasy w zależności od ich mutagenności dla bakterii Salmonella typhimurium. Zadaniem jest klasyfikacja binarna, czyli przewidywanie, czy dany związek chemiczny jest mutagenny dla bakterii Salmonella typhimurium na podstawie jego struktury chemicznej.
- **PTC**: Zestaw danych PTC składa się z 230 chemicznych związków, które są podzielone na dwie klasy w zależności od ich toksyczności dla gryzoni. Zadaniem jest klasyfikacja binarna, czyli przewidywanie toksyczności związku chemicznego na podstawie jego struktury chemicznej.
- **NCI1**: Jest to zestaw danych składający się z 4110 chemicznych związków, które są podzielone na dwie klasy w zależności od ich zdolności do hamowania wzrostu komórek nowotworowych. Zadaniem jest klasyfikacja binarna, czyli przewidywanie, czy dany związek chemiczny ma zdolność do hamowania wzrostu komórek nowotworowych na podstawie jego struktury chemicznej.
- **PROTEINS**: Zestaw danych PROTEINS składa się z 1113 białek, które są reprezentowane przez grafy. Każdy wierzchołek w grafie reprezentuje aminokwas, a krawędzie reprezentują różne typy sąsiedztwa między aminokwasami. Zadaniem jest przewidzenie, czy dane białko jest enzymem czy nie, na podstawie struktury grafu, co jest zadaniem klasyfikacji binarnej.
- **D&D**: Zestaw danych D&D składa się z 1178 białek, które są reprezentowane przez grafy. Każdy wierzchołek w grafie reprezentuje aminokwas, a krawędzie reprezentują różne typy sąsiedztwa między aminokwasami. Zadaniem jest klasyfikacja binarna, czyli przewidywanie, czy dane białko jest enzymem czy nie na podstawie jego struktury grafowej.
- **IMDBBINARY**: Ten zestaw składa się z sieci 1000 aktorów i aktorek z filmów na IMDb, reprezentujących dwa gatunki: Akcję i Romans. Zadaniem jest klasyfikacja binarna, czyli przewidywanie gatunku filmu na podstawie struktury sieci.
- **IMDBMULTI**: Ten zestaw zawiera sieci 1500 aktorów i aktorek z różnych filmów dostępnych na IMDb, które są zaliczane do trzech gatunków: Akcji, Romansu i Komedii. Celem jest klasyfikacja wieloklasowa, polegająca na przewidywaniu gatunku filmu na podstawie analizy struktury sieci.

Numeryczne statystyki zbiorów zostały przedstawione w Tabeli 1.

	MUTAG	**PTC**	**NCI1**	**PROTEINS**	**D&D**	**IMDB-B**	**IMDB-M**
Wierzchołki (maks.)	28	109	111	620	5748	136	89
Wierzchołki (śr.)	17.93	25.56	29.87	39.06	284.32	19.77	13.00
Grafy	188	344	4110	1113	1178	1000	1500

Tabela 1: Statystyki użytych zbiorów: maksymalna i średnia liczba wierzchołków oraz liczba grafów.

5 Przeprowadzone badania

W ramach badań, model został przetrenowany na siedmiu różnych, wcześniej wspomnianych zestawach danych. Dla każdego z nich przeprowadziliśmy 10-krotną walidację krzyżową (10-fold cross-validation), która jest techniką statystyczną używaną do oceny zdolności modelu do generalizacji na nieznane dane. Polega ona na podziale zestawu danych na 10 podzbiorów, a następnie trenowaniu modelu na dziewięciu z nich i testowaniu go na pozostałym. Proces ten jest powtarzany dziesięciokrotnie, co pozwala na uzyskanie bardziej stabilnej oceny wydajności modelu.

Aby uruchomić poszczególne walidacje krzyżowe, używana jest komenda `./run_DGCNN.sh $DATASET_NAME $bsize`, gdzie `$DATASET_NAME` to nazwa zestawu danych, na którym chcemy przeprowadzić walidację, a `$bsize` to rozmiar batcha. Ta komenda pozwala na automatyczne przeprowadzenie procesu walidacji krzyżowej dla danego zestawu danych.

Poniżej znajdują się komendy dla każdego z zestawów danych z odpowiednim rozmiarem batcha:

- **MUTAG**: `./run_DGCNN.sh MUTAG 16`
- **NCI1**: `./run_DGCNN.sh NCI1 100`
- **D&D**: `./run_DGCNN.sh DD 100`
- **PTC**: `./run_DGCNN.sh PTC 16`
- **PROTEINS**: `./run_DGCNN.sh PROTEINS 32`

- **IMDBBINARY**: `./run_DGCNN.sh IMDBBINARY 32`
- **IMDBMULTI**: `./run_DGCNN.sh IMDBMULTI 64`

Rozmiar batcha (`$bsize`) został dostosowany dla każdego zestawu danych w celu optymalizacji wydajności algorytmu. Kod oraz wyniki eksperymentów są dostępne na repozytorium[3].

6 Uzyskane wyniki

Wyniki z przeprowadzonych eksperymentów zostały przedstawione w Tabeli 2. Zbiory NCI1, D&D i PTC były uczone na 200 epokach, zbiory MUTAG oraz IMDBBINARY na 300, zbiór IMDBMULTI na 500, a zbiór PROTEINS na 100. Wszystkie zbiory były uczone z parametrem learning_rate równym 0.0001, poza zbiorami DD i PROTEINS, dla których ten parametr przyjął wartość 0.00001. Dodatkowo dla zbiorów IMDBBINARY i IMDBMULTI zmieniono wartość parametru sortpooling_k z domyślnej wartości 0.6 na wartość 0.9.

Dataset	MUTAG	PTC	NCI1	PROTEINS	D&D	IMDB-B	IMDB-M
Zbiór treningowy	89.69±1.75	79.41±3.39	75.15±0.88	72.96±0.78	77.29±0.35	73.13±0.41	52.89±8.63
Zbiór testowy	84.44±6.94	56.47±5.55	72.34±2.38	73.15±4.83	76.92±3.52	68.70±6.30	49.25±3.58

Tabela 2: Uzyskana skuteczność na zbiorze treningowym i testowym

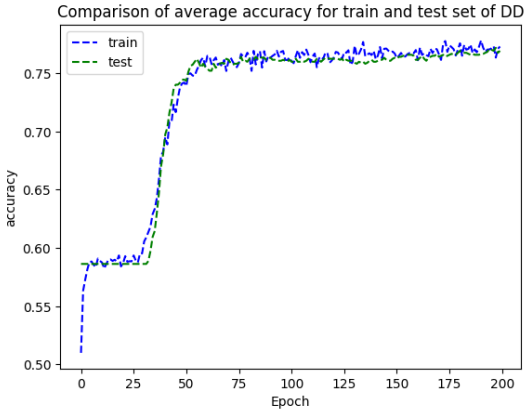
W przypadku zbiorów MUTAG i PTC, sieć zdawała się osiągać najlepszy metryki w okolicy 100-nej epoki, po której zaczynała przeuczać się na danych treningowych [Rysunek 2].

(a) dla zbioru MUTAG

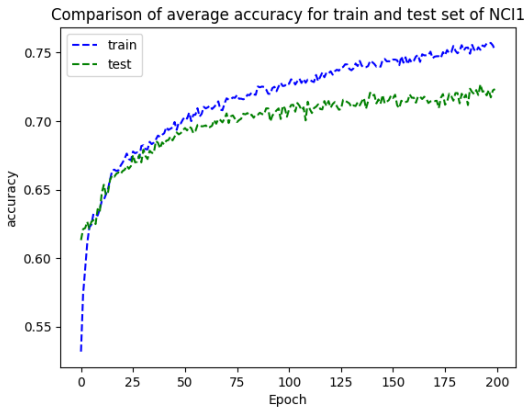
(b) dla zbioru PTC

Rysunek 2: Średnia wartość straty

W przypadku pozostałych zbiorów, proces uczenia wydaje się być "książkowy"[Rysunek 3 i 4] z drobnym odstępstwem w przypadku zbioru PROTEINS, dla którego sieć niby blokuje się w pewnym optimum około 15 epok na stałej wartości dokładności, mimo ciągłej minimalizacji straty zarówna na zbiorze treningowym i testowym [Rysunek 5].

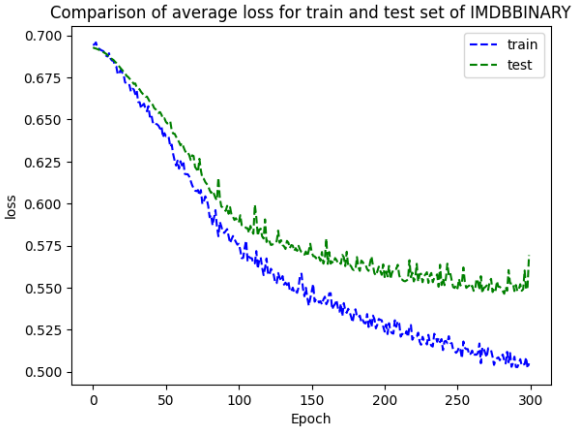


(a) dla zbioru D&D

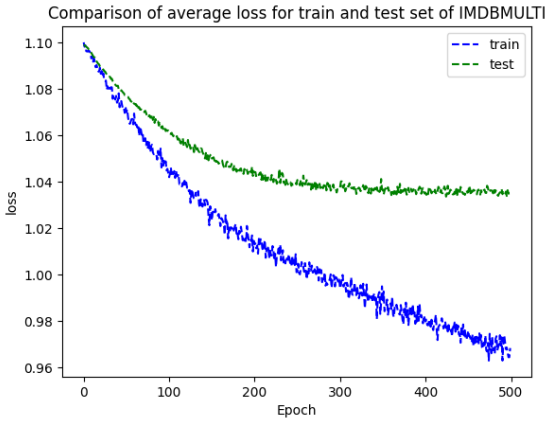


(b) dla zbioru NCI1

Rysunek 3: Średnia wartość dokładności

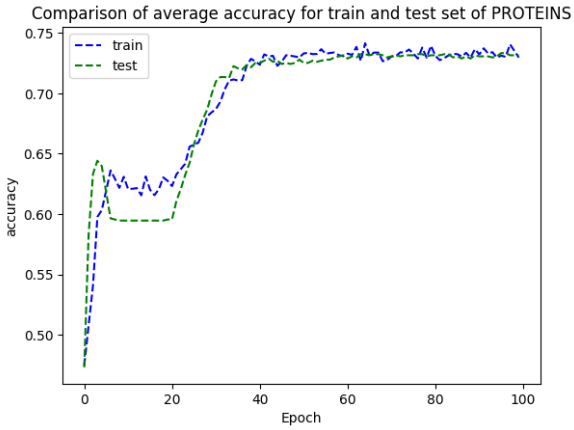


(a) dla zbioru IMDBBINARY

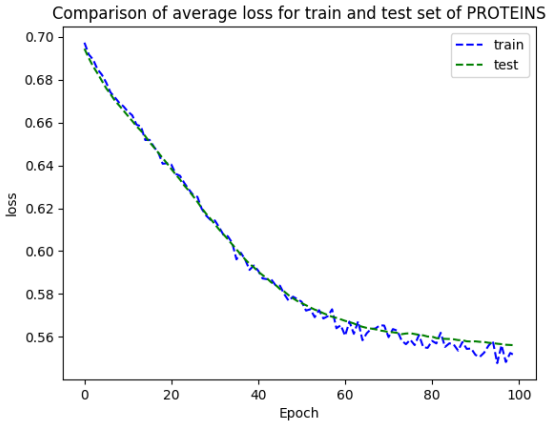


(b) dla zbioru IMDBMULTI

Rysunek 4: Średnia wartość fukncji kosztu



(a) skuteczność



(b) strata

Rysunek 5: Średnie wartości skuteczności i straty dla zbioru PROTEINS

Należy zaznaczyć, że w niniejszej sekcji zostały przedstawione tylko niektóre z wykresów wygenerowanych podczas analizy wyników. Pozostałe wykresy, w tym wykresy średniej dokładności i średniej straty dla każdego zestawu danych, a także wizualizacje dokładności i straty w trakcie treningu dla poszczególnych foldów (tj. tych z walidacji krzyżowej), można znaleźć w folderze `/notebooks`.

7 Porównanie wyników

7.1 Porównanie z wynikami z artykułu

Porównanie z wynikami z artykułu zostało przedstawione w Tabeli 3. Otrzymane wyniki nie pokrywają się z nimi idealnie, jednak można zauważyć powtarzający się trend. Uzyskane przez nas wyniki różnią się średnio o 2 punkty procentowe na niekorzyść autorów. Taki stan rzeczy usprawiedliwiamy faktem, że użyliśmy różnych wielkości batchy dla różnych datasetów, w celu przyśpieszenia procesu uczenia, które różniły się od domyślnych parametrów wywołania skryptu, dla których wielkość batchy była ustawiona na wartość 1. Wyjątkiem jest dataset IMDBMULTI, gdzie anomalnie udało nam się uzyskać wyniki lepsze niż te prezentowane w artykule.

Wyniki	**MUTAG**	**PTC**	**NCI1**	**PROTEINS**	**D&D**	**IMDB-B**	**IMDB-M**
uzyskane	84.44±6.94	56.47±5.55	72.34±2.38	73.15±4.83	76.92±3.52	68.70±6.30	**49.24±3.58**
z artykułu	**85.83±1.66**	**58.59±2.47**	**74.44±0.47**	**75.54±0.94**	**79.37±0.94**	**70.03±0.86**	47.83±0.85

Tabela 3: Porównanie uzyskanych wyników z wynikami z artykułu

7.2 Porównanie uzyskanych wyników z wynikami jąder grafowych

W Tabeli 4 wyniki uzyskane przy pomocy DGCNN są porównywane z czterema różnymi jądrami grafów: jądrem graphlet (GK), jądrem random walk (RW), jądrem propagacji (PK) i jądrem poddrzewa Weisfeiler-Lehman (WL).

Dataset	**MUTAG**	**PTC**	**NCI1**	**PROTEINS**	**D&D**
DGCNN	**84.44±6.94**	56.47±5.55	72.34±2.38	73.15±0.05	76.92±0.04
GK	81.39±1.74	55.65±0.46	62.49±0.27	71.39±0.31	74.38±0.69
RW	79.17±2.07	55.91±0.32	>3 days	59.57±0.09	>3 days
PK	76.00±2.69	**59.50±2.44**	82.54±0.47	73.68±0.68	78.25±0.51
WL	84.11±1.91	57.97±2.49	**84.46±0.45**	**74.68±0.49**	**78.34±0.62**

Tabela 4: Porównanie uzyskanych wyników z wynikami jąder grafowych

Przyjrzyjmy się teraz bliżej poszczególnym jądrom grafów:

- **Graphlet Kernel (GK)**: GK polega na porównywaniu małych podgrafów, zwanych graphletami. W kontekście DGCNN, GK osiąga niższą dokładność na wszystkich zbiorach w porównaniu do DGCNN. To sugeruje, że DGCNN jest bardziej skuteczny w wykorzystaniu informacji zawartych w badanych zbiorach danych.
- **Random Walk Kernel (RW)**: RW polega na porównywaniu losowych ścieżek w grafach. Literatura nie dostarcza wyników jądra RW dla zbiorów NCI1 i D&D, co wynika ze zbyt długiego działania w przypadku tych datasetów. Jednak wyniki dla innych zbiorów sugerują, że to jądro grafowe jest gorsze w porównaniu do DGCNN.
- **Propagation Kernel (PK)**: PK polega na propagacji etykiet węzłów przez graf i porównywaniu wynikowych rozkładów. PK osiąga wyższą dokładność na większości zbiorów w porównaniu do DGCNN, co sugeruje, że ta metoda może być bardzo skuteczna. Jednak DGCNN nadal osiąga konkurencyjne wynikii.
- **Weisfeiler-Lehman Subtree Kernel (WL)**: WL polega na iteracyjnym przypisywaniu nowych etykiet węzłom na podstawie ich sąsiedztwa, a następnie porównywaniu rozkładów etykiet. W porównaniu do DGCNN, WL osiąga wyższą, lecz nadal porównywalną dokładność na wszystkich zbiorach danych.

Podsumowując, DGCNN wykazuje się wysoką skutecznością w porównaniu do innych metod opartych na jądrach grafów. Co więcej, DGCNN jest trenowany za pomocą SGD, co pozwala uniknąć co najmniej kwadratowej złożoności w odniesieniu do liczby grafów wymaganych dla jąder grafów. Dlatego oczekuje się, że DGCNN będzie miał znaczną przewagę przy zastosowaniu do zbiorów danych na skalę przemysłową. Dodatkowo, wyniki uzyskane przez autorów artykułu poprzez lepsze dopasowanie parametrów i użycie 1 próbki na batch są lepsze od wyników uzyskanych w ramach badania. Te wyniki przewyższają prawie wszystkie wyniki uzyskane przez metody oparte na jądrach grafów, co potwierdza skuteczność i efektywność DGCNN.

7.3 Porównanie z wynikami innych architektur

W Tabeli 5 wyniki uzyskane przy pomocy DGCNN są porównywane z czterema innymi architekturami sieci neuronowych: PSCN, DCNN, ECC i DGK.

Dataset	**NCI1**	**PROTEINS**	**D&D**	**IMDB-B**	**IMDB-M**
DGCNN	72.34±2.38	73.15±0.05	**76.92±3.52**	68.70±6.30	**49.24±3.58**
PSCN	**76.34±1.68**	**75.00±2.51**	76.27±2.64	**71.00±2.29**	45.23±2.84
DCNN	56.61±1.04	61.29±1.60	58.09±0.53	49.06±1.37	33.49±1.42
ECC	76.82	-	72.54	-	-
DGK	62.48±0.25	71.68±0.50	-	66.96±0.56	44.55±0.52

Tabela 5: Porównanie uzyskanych wyników z wynikami innych architektur

Przyjrzyjmy się teraz bliżej poszczególnym architekturom:

- **PATCHY-SAN (PSCN)**: PSCN jest najbliższe DGCNN. W porównaniu do DGCNN, PSCN osiąga wyższą dokładność na większości zbiorów danych. To sugeruje, że PSCN może być bardzo skuteczne, ale DGCNN nadal osiąga konkurencyjne wyniki.
- **Diffusion-ConvNet (DCNN)**: DCNN wykorzystuje konwolucje grafowe do ekstrakcji cech podstruktur wieloskalowych. W porównaniu do DGCNN, DCNN osiąga znacznie niższą dokładność na wszystkich zbiorach danych, co sugeruje, że DGCNN jest bardziej skuteczny.

- **Edge-Conditioned Convolution (ECC)**: ECC to metoda, która rozwija koncept Neural Fingerprints poprzez dodanie hierarchii do procesu przypisywania etykiet węzłom na podstawie ich sąsiedztwa. ECC osiąga wyższą dokładność na zbiorze NCI1 w porównaniu do DGCNN, ale nie ma dostępnych wyników dla innych zbiorów. To sugeruje, że DGCNN jest bardziej skuteczny w wykorzystaniu informacji zawartych w badanych zbiorach danych.
- **Deep Graphlet Kernel (DGK)**: DGK uczy się podobieństw podstruktur za pomocą technik osadzania słów. W porównaniu do DGCNN, DGK osiąga niższą dokładność na wszystkich zbiorach danych, co sugeruje, że DGCNN jest bardziej skuteczny.

Podsumowując, DGCNN wykazuje się wysoką skutecznością w porównaniu do innych metod opartych na sieciach neuronowych. Dodatkowo, wyniki uzyskane przez autorów artykułu poprzez lepsze dopasowanie parametrów i użycie 1 próbki na batch są lepsze od wyników uzyskanych w ramach badania. Te wyniki przewyższają prawie wszystkie wyniki uzyskane przez metody oparte na innych architekturach sieci neuronowych, co potwierdza skuteczność i efektywność DGCNN.

8 Wnioski

Celem niniejszych badań było odtworzenie wyników autorów artykułu wprowadzającego sieci typu DGCNN oraz porównanie tych wyników do innych podejść rozwiązujących zadanie klasyfikacji grafów. Wykonane eksperymenty wskazują, że pomimo otrzymania metryk, które nie pokrywają się idealnie z oczekiwanymi, można uznać że skrypty oraz eksperymenty przeprowadzone przez autorów były rzetelne i umożliwiają one w dużym stopniu odtworzenie wyników z artykułu. Eksperymenty potwierdzają również wnioski autorów i wskazują że sieć DGCNN radzi sobie lepiej od klasycznych podejść wykorzystujących jądra grafowe czy od innych podejść, wykorzystujących metody głębokiego uczenia, wymienionych w artykule. Jest to spowodowane architekturą sieci, która przyjmuje bezpośrednio na wejściu strukturę grafu, bez potrzeby jej transformacji do tensorów. Dzięki takiemu zabiegowi cały proces nauki sieci, opierający się o wykorzystanie gradientu do minimalizacji funkcji celu, może obejmować wszystkie operacje, które są wykonywane na danych wejściowych.

Literatura

[1] Zhang, M., Cui, Z., Neumann, M., & Chen, Y. (2018). An End-to-End Deep Learning Architecture for Graph Classification. Proceedings of the AAAI Conference on Artificial Intelligence, 32(1). https://doi.org/10.1609/aaai.v32i1.11782

[2] Oryginalny kod źródłowy - https://github.com/muhanzhang/pytorch_DGCNN

[3] Kod oraz wyniki eksperymentów - https://github.com/maksanm/pytorch_DGCNN