

UNIwersytet WSB Merito w Poznaniu

Wydział Zamiejscowy w Chorzowie

**Anna Vezdenetska**

**Metody tworzenia grafów cząsteczek za pomocą  
technologii sztucznych sieci neuronowych: studium  
przypadku**

**Praca magisterska**

**Kierownik naukowy:**

**dr inż. Lesław Pawlaczyk**

**Kierunek: Informatyka**

**Specjalność: Programista Python**

**Numer albumu: 107938**

CHORZÓW 2024

# SPIS TREŚCI

<b>SPIS TREŚCI.....</b>	<b>1</b>
<b>WSTĘP.....</b>	<b>3</b>
<b>1 CEL I ZAKRES PRACY.....</b>	<b>5</b>
1. 1 Zakres pracy:.....	6
<b>2 CZĘŚĆ TEORETYCZNA.....</b>	<b>7</b>
2. 1 Sztuczne sieci neuronowe.....	7
2. 2 Budowa neuronu.....	7
2. 3 Budowa sieci neuronowej.....	9
<b>3 PRZEGLĄD WYKORZYSTANYCH TECHNIK I TECHNOLOGII.</b>	<b>11</b>
3. 1 Autoenkoder i jego budowa.....	12
3. 2 Wariacyjny autoenkoder (VAE).....	13
3. 3 Generatywna Sieć Przeciwstawna Wassersteina (WGAN).....	14
3. 4 WGAN z karą gradientową (WGAN-GP).....	15
3. 5 Relacyjna graficzna sieć splotowa (R-GCN).....	16
<b>4 PRZEGLĄD PRACY “DRUG MOLECULE GENERATION WITH VAE”.....</b>	<b>18</b>
4. 1 Często spotykane cząsteczki:.....	18
4. 2 Powtórzenie eksperymentu.....	22
4. 3 Regulacja zmiennej dropout rate w enkoderze.....	23
4. 3. 1 Wnioski podrozdziału:.....	27
4. 4 Regulacja zmiennej dropout rate w dekodery.....	29
4. 4. 1 Wnioski podrozdziału:.....	32
4. 5 Zmiana optymalizatora.....	34
4. 5. 1 Wnioski podrozdziału:.....	38
4. 6 Regulacja zmiennej batch size.....	38
4. 6. 1 Wnioski podrozdziału:.....	42
4. 7 Regulacja zmiennej batch normalization.....	44
4. 7. 1 Wnioski podrozdziału:.....	46
4. 8 Rozbudowa warstw.....	47
4. 8. 1 Rozbudowa warstwy w pełni połączonej enkodera.....	47
4. 8. 2 Rozbudowa warstw dekodera.....	49
4. 8. 3 Wnioski podrozdziału:.....	50
4. 9 Podsumowanie przeglądanej pracy.....	51

<b>5 PRZEGLĄD PRACY “WGAN-GP WITH R-GCN FOR THE GENERATION OF SMALL MOLECULAR GRAPHS”</b>	<b>53</b>
5. 1 Powtórzenie eksperymentu	54
5. 2 Wpływ zmiennej dropout_rate	55
5. 2. 1 Zmienna dropout_rate dla generatora	55
5. 2. 2 Zmienna dropout_rate dla dyskryminatora	61
5. 2. 3 Wnioski podrozdziału 5. 2:	65
5. 3 Wpływ optymalizatora	67
5. 3. 1 Analiza wyników	67
5. 3. 2 Wnioski podrozdziału:	68
5. 4 Regulacja zmiennej batch size	71
5. 4. 1 Wnioski podrozdziału:	73
5. 5 Wprowadzenie filtra unikalności	73
5. 5. 1 Eksperymenty z filtrem unikalności	74
5. 5. 1. 1 Trenowanie sieci o podstawowej architekturze	75
5. 5. 1. 2 Trenowanie sieci o wybranej wartości zmiennej dropout_rate	76
5. 5. 1. 3 Trenowanie sieci o wybranych optymalizatorach	77
5. 5. 1. 4 Trenowanie sieci o wybranym wsadzie	79
5. 5. 2 Wnioski podrozdziału:	80
5. 6 Podsumowanie przeglądanej pracy	80
<b>ZAKOŃCZENIE</b>	<b>80</b>
6. 1 Porównanie przeglądanych prac	81
6. 2 Porównanie otrzymanych wyników	82
<b>BIBLIOGRAFIA:</b>	<b>84</b>
Pozycje zwarte:	84
Artykuły w czasopismach:	84
Strony internetowe:	86
Materiały niepublikowane:	86
<b>SPIS RYSUNKÓW</b>	<b>87</b>
<b>SPIS TABEL</b>	<b>89</b>

# WSTĘP

Celem tej pracy jest przeprowadzenie studium przypadku dotyczącego metod tworzenia grafów cząsteczek za pomocą technologii sztucznych sieci neuronowych. W celu przeprowadzenia owego studium przypadku dokonano przeglądu prac, gdzie za pomocą zastosowania modeli z sieciami głębokiego uczenia zostały wygenerowane graficzne przedstawienia molekuł. Wybrane do przeglądu prace, to “*Drug Molecule Generation with VAE*” [35] oraz “*WGAN-GP with R-GCN for the generation of small molecular graphs*” [36]. Obecna praca jest podzielona na dwie główne części, teoretyczną i praktyczną. Praca zawiera sześć rozdziałów.

Pierwszy rozdział objaśnia bardziej szczegółowo cel i zakres wykonanej pracy. Drugi rozdział przybliży takie kluczowe pojęcia dla zrozumienia tematu pracy jak, czym jest sztuczna sieć neuronowa, budowa neuronu jako fundamentalnej jednostki sieci, a także opis architektury sieci neuronowej.

Trzeci rozdział został poświęcony opisaniu technik oraz technologii wykorzystanych w pracach, które stanowiły podstawę przeprowadzonych eksperymentów opisanych w rozdziale czwartym i piątym. Rozpatrzono takie techniki jak autoenkoder oraz wariacyjny autoenkoder (*VAE*). Ostatnia technika wariacyjnego autoenkodera została wykorzystana w przeglądanej pracy, o tytule “*Drug Molecule Generation with VAE*”. Także przybliżono takie technologie jak: Generatywna Sieć Przeciwna Wassersteina (*WGAN*), sieci *WGAN* z karą gradientową (*WGAN-GP*), Relacyjna graficzna sieć splotowa (*R-GCN*). Technologie *WGAN-GP* oraz *R-GCN* zostały wykorzystane w innej pracy, o tytule: “*WGAN-GP with R-GCN for the generation of small molecular graphs*”, której przegląd poświęcono piąty rozdział owej pracy.

W przypadku pierwszej pracy, “*Drug Molecule Generation with VAE*” skupiono się na eksperymentach poświęconym regulacji takich zmiennych jak: *dropout rate*, optymalizatora, *batch size*, *batch normalization*. Oprócz tych eksperymentów, został także przeprowadzony eksperyment poświęcony rozbudowie warstw dwóch modeli tej techniki: enkodera i dekodera. W przypadku drugiej pracy, “*WGAN-GP with R-GCN for the generation of small molecular graphs*”, powtórzono eksperyment poświęcony regulacji takich zmiennych jak: *dropout rate*, optymalizatora, *batch size*. Przegląd

ostatniej pracy pozwolił także wprowadzić filtr unikalności, który pozwolił generować unikalne cząsteczki w trakcie każdego trenowania sieci.

W zakończeniu podsumowano wyniki wszystkich badań oraz porównano przeglądane prace.

Celem przeglądu wyżej wspomnianych prac jest sprawdzenie skuteczności działania badanych sieci neuronowych pod wpływem wprowadzenia zmian do różnych hiperparametrów, takich jak *dropout rate*, optymalizator, *batch size*, *batch normalization*. Także dany przegląd prac obejmuje rozbudowę warstw enkodera i dekodera. Wykonane badania, mają na celu odnalezienie sposobu tworzenia unikalnych struktur chemicznych, które mogą zostać wykorzystane w przemyśle chemicznym lub farmakologicznym.

Warto także zaznaczyć że dana praca, skupia się wyłącznie na generowaniu unikalnych cząsteczek, lecz niektóre cząsteczki, wygenerowane przez badane sieci neuronowe mogą być niestabilne, lub niemożliwe. Jako przykład można opisać cząsteczkę, w której atom wodoru (H) posiada dwa wiązania. Taka cząsteczka, jest niezgodna z regułami chemicznymi, gdyż atom wodoru składa się z jednego protonu i jednego elektronu, co oznacza że może tworzyć tylko jedno wiązanie [2]. Taką błędną strukturę chemiczną można użyć jako "negatywny przykład", aby ocenić jakość modelu, gdyż model powinien funkcjonować z zachowaniem zasad chemicznych.

# 1 CEL I ZAKRES PRACY

Celem wykonanej pracy dyplomowej było uruchomienie algorytmów, przedstawionych w różnych artykułach naukowych, gdzie została wykorzystana technologia generatywnej sieci neuronowej dla utworzenia grafów unikalnych molekuł. Powtórne przeprowadzenie badań pozwala zweryfikować poprawność zastosowanych metod i algorytmów, a także ocenić jakość osiągniętych wyników. Dodatkowo, powtórzenie eksperymentów może pomóc w identyfikacji potencjalnych problemów lub błędów, które mogą wpłynąć na wyniki eksperymentów, oraz wskazanie potencjalnych obszarów, w których można dalej rozwijać dane podejście.

Otrzymane wyniki powtórzenia eksperymentów mogą mieć istotne znaczenie dla dalszych badań w danej dziedzinie i mogą wpłynąć na sposób, w jaki badacze wykorzystują dane metody i algorytmy w swoich badaniach. Powtórzenie eksperymentów stanowi ważny element metodyki naukowej, który pozwala na potwierdzenie i weryfikację wyników, a także na rozwój nauki poprzez udoskonalanie i dalsze badania danego zagadnienia.

Oprócz powtórzenia wyników, które zostały przedstawione w wykorzystanych pracach, zostały dodatkowo przeprowadzone eksperymenty, w których zmodyfikowane zostały wybrane hyperparametry sieci neuronowej. Głównie były to następujące zmienne: *dropout rate*, *batch size*, *batch normalization*, a także wybranie optymalizatora. Dane badania pozwolą zrozumieć optymalne wartości wybranych zmiennych, które nie zakłócają działania wybranego algorytmu, oraz zweryfikować wpływ wprowadzonych zmian na wynik końcowy.

Nauka wybranych sieci odbywa się w oparciu dwóch wybranych modeli generatywnych: VAE (z ang. *Variational Autoencoder*) oraz WGAN-GP (z ang. *Wasserstein Generative Adversarial Network z Gradient Penalty*).

VAE to model generatywny oparty na wariacyjnym autoenkoderze, jego struktura jest opisana w dalszej części danej pracy w drugim podrozdziale trzeciego rozdziału. Autoenkoder wariacyjny to probabilistyczny model generatywny, składający się z dwóch głównych części: enkodera i dekodera. Celem autoenkodera wariacyjnego jest generowanie danych poprzez formułowanie prawdopodobieństwa w ukrytej przestrzeni  $h$ . Enkoder przekształca zmienną wejściową w ukrytą przestrzeń, reprezentującą parametry o zmiennym rozkładzie. Dekoder natomiast odwrotnie -

generuje dane, przekształcając informacje z przestrzeni ukrytej z powrotem do przestrzeni wejściowej [4].

Inny model generatywny to WGAN-GP, jest to ulepszoną wersją sieci neuronowej typu GAN. Budowa i szczegółowy opis tej technologii znajdują się w podrozdziale czwartym rozdziału trzeciego. W tym modelu, algorytm dąży do zminimalizowania odległości *Wassersteina*<sup>1</sup> między rozkładem generowanym, a rzeczywistych danych. Autorzy, którzy opracowali dane podejście w opublikowanej przez nich pracy dwa tysiące siedemnastego roku, dążyły do poprawienia stabilności uczenia się sieci neuronowej [10].

Celem przeglądu wyżej wspomnianych prac jest poszukiwanie optymalnej architektury sieci neuronów, która w trakcie każdego trenowania, będzie generowała unikalne graficzne przedstawienia molekuł. Zbadanie wpływu hyperparametrów pozwala zrozumieć ich wpływ na trenowany model. Z kolei zoptymalizowany model, pozwala uzyskać bardziej stabilne i wiarygodne wyniki.

## 1. 1 Zakres pracy:

- przegląd literatury zgodnej z tematyką pracy dyplomowej,
- uruchomienie sztucznych sieci neuronowych, powołanych do tworzenia grafów wielkocząsteczkowych,
  - przeprowadzenie badań na wybranych fragmentach kodu wyżej wymienionej sieci oraz opis otrzymanych wyników,
  - przeprowadzenie samodzielnych badań w obrębie danych sieci i opis wyników, otrzymanych po modyfikacji konkretnych zmiennych,
  - sporządzenie wniosków na podstawie przeprowadzonych badań i zdobytej wiedzy.

---

<sup>1</sup> Odległość Wassersteina (znana inaczej jako *earth mover's distance*) między dwoma rozkładami prawdopodobieństwa intuicyjnie odwzorowuje ilość pracy, którą należy wykonać w celu przekształcenia jednego rozkładu w drugi [37].

## 2 CZĘŚĆ TEORETYCZNA

Obecny rozdział poświęcony został ogólnym zagadnieniom dotyczącym sieci neuronowych ANN (z języka angielskiego *Artificial Neural Networks*). Zostaną przedstawione trzy kluczowe zagadnienia związane z sztucznymi sieciami neuronowymi. Pierwszy z nich dotyczy samej koncepcji sztucznych sieci neuronowych, następnie omówiona zostanie budowa pojedynczego neuronu, który jest podstawową jednostką w tego typu sieciach. Na końcu omówimy ogólną strukturę i budowę sztucznej sieci neuronowej [4].

### 2.1 Sztuczne sieci neuronowe

Sztuczne sieci neuronowe, są to specyficzne rodzaje algorytmów uczenia maszynowego. Dane algorytmy uczenia maszynowego wyróżniają się budową, gdyż zostały skonstruowane tak aby w pewnym stopniu naśladować funkcje podobne do biologicznej sieci neuronowej. Struktura sztucznej sieci, jak i sieci biologicznej składa się z neuronów, których połączenia tworzą taką sieć [7].

Podstawową jednostką każdej sieci neuronowej jest tak zwany neuron. Dany element sieci jest powołany do przetworzenia informacji, w pewnym stopniu wzorowany na funkcjonowaniu biologicznej komórki nerwowej. Lecz jeśli porównywać ich budowę, to budowa sztucznego neuronu, okazują się bardziej uproszczona, w porównaniu do wersji biologicznej [4]. Biologiczne neurony przekazują sygnały elektryczne (potencjały czynnościowe) z dendrytów do zakończeń aksonu przez korpus aksonu. W taki sposób sygnały elektryczne są przesyłane przez synapsę z jednego neuronu do drugiego. Ludzki mózg ma około 100 miliardów neuronów [1 , 9]. Porównując taki poziom złożoności, jest oczywiste że jest dość trudno jego naśladować za pomocą istniejących komputerów [1].

### 2.2 Budowa neuronu

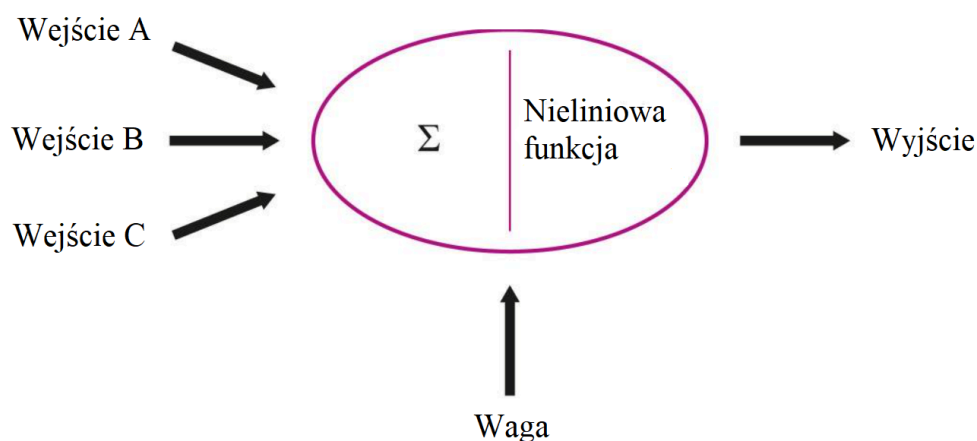
Budowa sztucznego neuronu jest bardziej uproszczona w porównaniu do biologicznych neuronów, jak już wspomniano wcześniej. Natomiast, nazwa taka tych



jednostek przyjęła się, ze względu na to że dana struktura posiada cechy i funkcje, które można zobaczyć także w modelu biologicznym [14]. W strukturze neuronu odnaleźć można wiele wejść oraz jedno wyjście, jest to cecha wspólna jak dla neuronów biologicznych, tak i dla neuronów sztucznych. Kolejną cechą wspólną jest komunikacja między neuronami, gdyż dzięki tym połączeniom, tworzy się sieć neuronowa. Sposób komunikacji natomiast w neuronach różni się, gdyż w sztucznych neuronach komunikacja następuje przez komplet wag, których wartości decydują o zachowaniu neuronu [19].

Porównując komunikację między neuronami biologicznymi, a sztucznymi, warto wspomnieć że neurony biologiczne otrzymują wiele sygnałów wejściowych od neuronów presynaptycznych [1]. Neurony w sztucznych sieciach neuronowych, w tak zwanych węzłach, także otrzymują kilka sygnałów, następnie otrzymane sygnały są dodawane i przetwarzane za pomocą pewną nieliniowej funkcji sumy jego wejść [4].

Otrzymana wartość przetworzona po wykonaniu wyżej wymienionych czynności, staje się wartością, którą otrzymamy na wyjściu. Poniżej przedstawiony rysunek pokazuje opisaną budowę neuronu. Ważne także zaznaczyć, że wartość wyjściowa neuronu zostanie wygenerowana, tylko wtedy, gdy suma wejść A, B i C przekroczy próg i zadziała nieliniowa funkcja [14].



Rysunek 1. Budowa neuronu

*Źródło: Opracowanie własne A. Vezdenetska na podstawie obrazu opublikowanego na stronie [38]*

Wszystkie sztuczne neurony posiadają kilka kluczowych elementów w swojej budowie, jak to zostało przedstawione na rysunku pierwszym. Są to sygnał wejściowy, waga, funkcja aktywacji oraz sygnał wyjściowy. Proces działania neuronu rozpoczyna

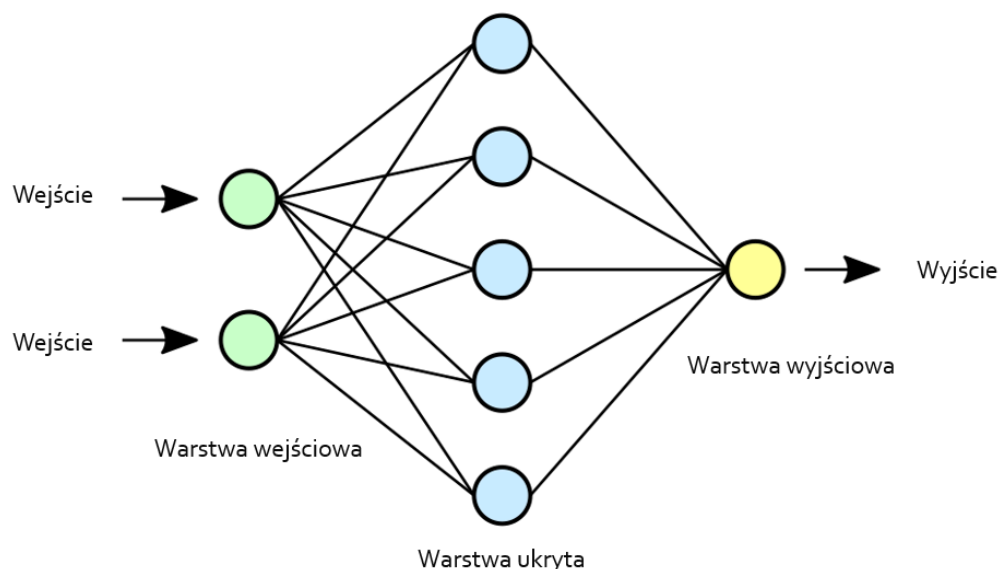
się od przemnożenia sygnału wejściowego przez przydzieloną mu wagę. Następnie te przemnożone wartości są sumowane. Kolejnym krokiem jest przetworzenie tej sumy przez nieliniową funkcję aktywacji. Takie przetworzenie danych decyduje o tym, czy neuron zostanie aktywowany, co z kolei ma wpływ na przekazywanie informacji od tego neuronu do kolejnych elementów sieci, ponieważ neuron musi osiągnąć pewien próg aktywacji. Jeśli sygnał nie przekroczy tego progu, informacja nie zostanie przekazana dalej [39].

Wagi, które pozwalają komunikować się neuronom w sieci neuronowej, najczęściej nie mogą mieć stałych wartości i mogą ulegać zmianom w procesie uczenia sieci. Wagi określają siłę połączeń między neuronami i decydują, jak duży wpływ ma sygnał wejściowy na neuron i jak mocno neuron przekazuje sygnał dalej [6].

W procesie nauki sieci neuronowej wagi ulegają modyfikacji. Zmiany te mają na celu wspomaganie procesu nauki sieci, aby ona mogła prawidłowo odpowiadać na zadane pytania lub rozpoznawać wzorce w danych. Zmiana wartości wag pozwala na dostosowanie sieci do pewnego konkretnego zadania, a cały proces jest realizowany poprzez iteracyjne przetwarzanie danych uczących, które są prezentowane sieci razem z pożądanymi wynikami. W trakcie nauki sieć dostosowuje wagi w celu minimalizacji błędów i osiągnięcia jak najlepszych wyników [5].

## **2. 3 Budowa sieci neuronowej**

Połączenia neuronów i sposób ich ułożenia, określają strukturę sieci neuronowej, zwanej również jej architekturą [9]. Zazwyczaj neurony są agregowane w warstwy, które pełnią różne funkcje w przetwarzaniu informacji. W większości przypadków neurony nie łączą się ze sobą w obrębie tej samej warstwy. Natomiast wszystkie połączenia neuronów jednej warstwy, zwykle przesyłają sygnał do pewnego neuronu w następnej warstwie [39]. Poniższy rysunek dwa przedstawia strukturę sieci neuronowej ze sprzężeniem do przodu, jak widać na rysunku, w takiej strukturze neurone, nie są połączone w obrębie tej samej warstwy.



Rysunek 2. Struktura sieci neuronowej ze sprzężeniem do przodu

*Źródło: Opracowanie własne A. Vezdenetska na podstawie obrazu [39]*

Różne warstwy mogą wykonywać różne transformacje na swoich wejściach. Sygnały wag przemieszczają się z pierwszej warstwy, którą nazywamy warstwą wejściową do ostatniej warstwy, tak zwana warstwa wyjściowa. ponieważ wszystkie połączenia przechodzą między neuronami z różnych warstw. W strukturze sieci, można wyróżnić trzy typy warstw, widoczne na rysunku dwa [9; 11].

Jak przedstawiono na powyższym rysunku, sieci neuronowe są zbudowane z wielu połączonych ze sobą neuronów, które przetwarzają informacje. Każdy neuron otrzymuje sygnały wejściowe, przetwarza je i przekazuje dalej. W procesie uczenia sieci neuronowej, wagi połączeń między neuronami są dostosowywane w celu uzyskania oczekiwanych wyników [6].

Każda warstwa posiada własną funkcję. Pierwsza warstwa to warstwa wejściowa, której zadaniem jest wprowadzenie sygnałów wejściowych do sieci. Ostatnia warstwa to warstwa wyjściowa, odpowiedzialna za generowanie sygnałów wyjściowych [39]. Wszystkie warstwy, znajdujące się pomiędzy pierwszą - wejściową, a ostatnią - wyjściową, nazywane są warstwami ukrytymi. Ich liczba ma wpływ na "inteligencję" sieci, co oznacza, że im więcej warstw ukrytych posiada sieć neuronowa, tym trudniejsze zadania może rozwiązać. Warto jednak wspomnieć, że w takiej sytuacji również trudniej jest taką sieć trenować. Istnieje także możliwość zdefiniowania sieci bez warstw ukrytych [7].

## 3 PRZEGLĄD WYKORZYSTANYCH TECHNIK I TECHNOLOGII

W danym rozdziale omówiono kluczowe techniki i technologie związane z wykonaną pracą badawczą. Każdy podrozdział został poświęcony określonej technice lub technologii. Poniżej znajduje się opis każdego z podrozdziałów, z których składa się dany rozdział:

3.1. Autoenkoder i jego budowa: Dany podrozdział poświęcono opisaniu podstawowych założeń i budowy autoenkoderów, z omówieniem głównych składowych tej sieci neuronowej: enkodera i dekodera. Dla lepszego zrozumienia zawiera także ilustracje struktury danego modelu.

3.2. Wariacyjny autoenkoder (VAE): Skupia się na modelu wariacyjnego autoenkodera (VAE), podrozdział ten prezentuje szczegółową analizę tej techniki, gdyż jest ona wykorzystane w jednej z prac badawczych. Zostaną przedstawione zasady działania VAE oraz sposób, w jaki wykorzystuje modelowanie probabilistyczne. Wyjaśnia, jak VAE różni się od klasycznego autoenkodera oraz jakie ma zalety w kontekście generatywnego modelowania danych.

3.3. Generatywna Sieć Przeciwna Wassersteina (WGAN): Opisuje ogólne założenia i budowa Generatywnej Sieci Przeciwna (GAN) z wykorzystaniem odległości Wassersteina. W podrozdziale, także omówiono różnice między klasyczną budową modeli GAN-ów i budową modeli WGAN. Także omówione zostaną zasady działania tego modelu.

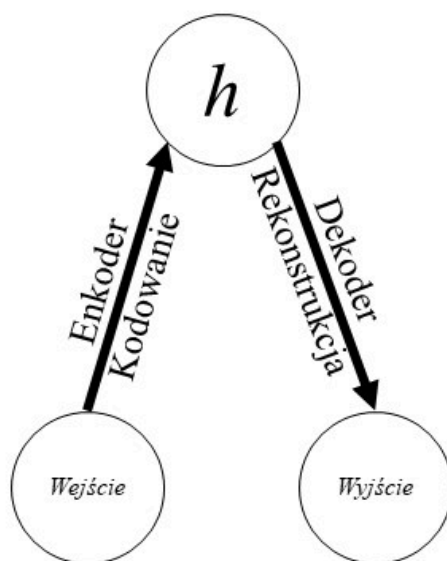
3.4. WGAN z karą gradientową (WGAN-GP): W tym podrozdziale przedstawiono opis ulepszanego modelu WGAN (z ang. *Gradient Penalty - GP*), w oparciu o karę gradientu. Wyjaśnia, jakie korzyści niesie za sobą dane ulepszenie na jakość generowanych danych.

3.5. Relacyjna graficzna sieć splotowa (R-GCN): Jest to ostatni podrozdział danego rozdziału. Tu będzie omówiona technika R-GCN (z ang. *Relational Graph Convolutional Network*), która jest istotna w kontekście analizy grafów molekuł. Zostaną przedstawione podstawy tej techniki oraz jej zastosowanie w kontekście generowania graficznych reprezentacji molekuł.

Każdy z podrozdziałów pozwala przybliżyć teorię o technikach, które zostały użyte w trakcie przeglądania prac badawczych, poświęconych do tworzenia graficznego przedstawienia molekuł.

### 3. 1 Autoenkoder i jego budowa

Autoenkoder jest rodzajem sieci neuronowej używanym w zadaniach uczenia nienadzorowanego. Podczas trenowania sieć podejmuje próbę oddzielenia istotnych cech od danych wejściowych i ich rekonstrukcji na wyjście. Ogólną strukturę autoenkodera, przedstawia poniższy rysunek trzy. Na poniższym rysunku trzecim, przedstawiono przykład działania autoenkodera. Przedstawiona budowa przedstawia to że autoenkoder składa się z dwóch głównych komponentów: enkodera i dekodera, które współpracują, aby przekształcić dane wejściowe poprzez warstwę ukrytą  $h$  do wyjścia [22].



Rysunek 3. Ogólna struktura autoenkodera.

*Źródło: Opracowanie własne A. Vezdenetska na podstawie schematu [4]*

Enkoder, odpowiada za mapowanie najważniejszych cech z danych wejściowych oraz przekazuje otrzymane dane na warstwę ukrytą  $h$ . Dany proces jest określany jako kodowanie. W wyniku działania kodera otrzymujemy ukrytą warstwę  $h$ , która jest bardziej kompaktowa niż dane wejściowe. Zmniejszenie rozmiaru tej warstwy wynika z faktu, że warstwa  $h$  zawiera jedynie mapę istotnych cech. Następnie dekodek, jest

próbuję odzyskać dane wejściowe korzystając z zakodowanych danych wejściowych, dany proces jest określany jako rekonstrukcja [4].

Celem autoenkodera jest minimalizowanie różnicy między danymi wejściowymi oraz tymi, które otrzymywane na wyjściu. Dane podejście pozwala na efektywne uczenie się ważnych cech danych. W trakcie nauki sieć stara się zoptymalizować wewnętrzną reprezentację  $h$  tak, aby jak najlepiej odzwierciedlała cechy danych wejściowych [9].

Podsumowując ogólną ideą procesu kodowania-dekodowania: enkoder poszukuje cech wspólnych i przekazuje zakodowaną informację do dekodera, który na podstawie otrzymanej mapy, próbuje odtworzyć pierwotne dane wejściowe. Taki mechanizm nie pozwala otrzymać próbek bardzo podobne do danych wejściowych, które nie są pełną kopią [22].

### 3. 2 Wariacyjny autoenkoder (VAE)

W pierwszej pracy, której przegląd został wykonany, został wykorzystany wariacyjny autoenkoder (z ang. *Variational encoder*). VAE umożliwia modelowanie probabilistyczne, co sprawia, że różni się od tradycyjnych autoenkoderów [21].

Jeśli porównamy tradycyjny model i wariacyjny, to modelowanie probabilistyczne jest główną cechą odróżniającą VAE. W tradycyjnym autoenkoderze, dwa modele enkoder i dekoderek uczą się dokładnie odwzorowywać dane z zestawu uczącego, gdzie enkoder koduje dane do przestrzeni ukrytej, a dekoderek rekonstruuje dane wejściowe, na podstawie zakodowanych cech [21]. Natomiast w przypadku VAE, enkoder również kompresuje dane do przestrzeni ukrytej, lecz kodowanie tych cech nie jest deterministyczne, jak w tradycyjnych autoenkoderach, ale probabilistyczne. Oznacza to, że enkoder nie zwraca jednej konkretnie określonej reprezentacji, ale rozkład prawdopodobieństwa reprezentacji. VAE uczy się, jakie są statystyczne cechy (takie jak wartości średnie i odchylenia standardowe) tych zmiennych losowych, aby móc generować nowe próbki, które będą podobne do danych ze zbioru treningowego, lecz nie będą ich kopiować [9].

W przypadku procesu kodowania w VAE, enkoder mapuje dane wejściowe do rozkładu prawdopodobieństwa w przestrzeni ukrytej, zamiast jednoznacznie określać konkretną reprezentację. Cel nauki danych modeli jest również inny, gdyż VAE dąży

zminimalizować odległość między rozkładem prawdopodobieństwa danych wejściowych, a rozkładem prawdopodobieństwa generowanych danych. Natomiast tradycyjny autoenkoder dąży do minimalizacji błędu przy rekonstrukcji [11].

Autoenkodery wariacyjne to probabilistyczne modele generatywne. Oznacza to, że VAE potrafią generować nowe próbki, które przypominają próbki ze zbioru treningowego, a proces generowania nowych próbek jest oparty na probabilistycznych właściwościach modelu. VAE podczas procesu nauki koduje dane wejściowe, tworząc rozkład prawdopodobieństwa w przestrzeni ukrytej, a następnie rekonstruuje dane poprzez losowe próbkowanie z tego rozkładu. Dany układ pozwala generować nowe próbki, które nie były obecne w zbiorze treningowym [21].

VAE znajdują zastosowanie w wielu dziedzinach, ze względu na ich zdolność do generowania nowych przykładów danych, reprezentacji przestrzeni cech oraz efektywnego uczenia reprezentacji danych. Jednak w przeglądanej pracy autoenkodera wariacyjnego użyto do tworzenia oryginalnych próbek molekuł [1].

### **3. 3 Generatywna Sieć Przeciwna Wassersteina (WGAN)**

Rozwinięcie koncepcji generatywnej sieci przeciwstawnej (z ang. *Generative Adversarial Networks*), która zostanie omówiona w tym podrozdziale, to generatywna sieć przeciwna Wassersteina (z ang. *Wasserstein Generative Adversarial Networks*). Jedną z głównych różnic między tymi metodami, leży w sposobie oceniania jakości generowanych danych. W metodzie GAN, jakość jest oceniana za pomocą funkcji straty, a w metodzie WGAN, jakość generowanych danych jest oceniana za pomocą odległości Wassersteina [10].

Proces trenowania GAN, został opisany przez twórców tego modelu, jako gra pomiędzy dwiema konkurującymi sieciami. Jedna sieć, to tak zwany generator, który tworzy dane podobne do rzeczywistych danych treningowych. Druga sieć, jest nazywana dyskriminatorem, która szacuje prawdopodobieństwo, tego że próbka pochodzi z danych uczących, a nie od generatora. Główny cel generatora to zwiększyć prawdopodobieństwo tego że dyskriminator, popełni błąd i przegra [24].

W tradycyjnych GAN-ach, funkcją straty służy wyznacznikiem, który pozwala rozpoznać jak bardzo wygenerowana próbka jest podobna do danych wejściowych.

Najczęściej używana funkcja straty jest logarytmiczną funkcją, która oblicza różnice między danymi wejściowymi, a wygenerowaną próbką. Użycie innej metody do obliczenia danej różnicy może powodować, takie problemy jak "zanikający gradient" (z ang. *vanishing gradient*) oraz "złamanie różnorodności" (z ang. *mode collapse*). Znikający gradient, utrudnia proces trenowania, dany problem występuje, kiedy małe gradienty w początkowych warstwach sieci neuronowej utrudniają skuteczną aktualizację wag. Złamanie różnorodności, natomiast, występuje kiedy generator produkuje jedynie ograniczony zestaw próbek, przez co poprawienie generowanych próbek, staje się trudniejsze [10].

Jednak, także warto zaznaczyć że funkcja logarytmiczna, także może powodować pojawienie się tego problemu. Wasserstein GAN zmienia podejście, wprowadzając odległość Wassersteina jako funkcję kosztu, w celu rozwiązania problemu "zanikającego gradientu" zostało opracowane ulepszenie, takie jak Wasserstein GAN, które także pozwoliło na bardziej stabilne i różnorodne generowanie próbek. Dane ulepszenie było możliwe, przez to że ta miara ma lepsze własności matematyczne oraz umożliwia bardziej stabilne i efektywne trenowanie modelu GAN [13].

Główna zasada działania WGAN jest opiera się na to, że generator dąży zminimalizować odległość Wassersteina, co oznacza generowanie próbek, które będą zbliżać się do danych wejściowych. Dla dyskriminatora celem jest określenie, jak blisko generowane dane są do danych z zestawu treningowego pod względem odległości Wassersteina. W trakcie trenowania sieci, takie podejście umożliwia doskonalenie generatora [10].

Podsumowując, WGAN wprowadza odległość Wassersteina jako miarę efektywności w treningu modelu GAN, pozwalając wyeliminować niektóre problemy związane z tradycyjnymi GAN-ami [10].

### **3. 4 WGAN z karą gradientową (WGAN-GP)**

Jak już omówiono w poprzednim podrozdziale, tradycyjne generatywne sieci przeciwstawne, mogą napotkać wiele różnych problemów, takich jak "złamanie różnorodności" oraz "zanikający gradient". Dane problemy mogą skutkować niestabilnością procesu nauki. Oprócz zastosowania odległości Wassersteina, stosowanej w celu rozwiązania powyżej wspomnianych problemów, omówione



zostanie rozwinięcie poprzedniego podejścia, gdzie została zastosowana kara gradientowa. Dane podejście jest znane jako generatywna sieć przeciwstawna Wassersteina z karą gradientu (z ang. *Wasserstein Generative Adversarial Network with Gradient Penalty*) [26].

Model WGAN-GP ma na celu osiągnięcie bardziej stabilnego i kontrolowanego procesu uczenia modelu generatywnego. Funkcja kary, stosowana w WGAN-GP, opiera się na odległość Wassersteina, co skutkuje otrzymaniem bardziej stabilnego procesu uczenia. Odporność generatora do pojawienia się zjawiska nazywanego "złamaniem różnorodności" (z ang. *mode collapse*) wzrasta, w związku z czym otrzymywane próbki stają się bardziej rozmaite. Implementacja kary do modelu WGAN, powoduje wprowadzenie dodatkowego wymogu na różnice pomiędzy wartościami funkcji generatora. Dzieje się tak ze względu na wprowadzenie kary dla gradientów dyskryminatora. Wprowadzenie danej kary powoduje to że dyskryminator, staje się bardziej ostrożny i dokładny w ocenie otrzymywanych próbek aby "uniknąć" kary. Wzrastające wymogi dyskryminatora, z kolei mają wpływ na ulepszenie próbek tworzonych w generatorze, gdyż on musi produkować próbki jak najbliższe do danych wejściowych, by one zostały zaakceptowane dyskryminatorem [13].

Podsumowując obecny podrozdział, zastosowanie kary dla gradientów dyskryminatora, pozwala podwyższyć jakość i różnorodność próbek kreowanych przez generator. Również dane działanie prowadzi do stabilizacji procesu nauki sieci neuronowej. Zastosowanie modelu WGAN-GP, pomaga w redukcji problemu "złamania różnorodności" [13].

### **3. 5 Relacyjna graficzna sieć spłotowa (R-GCN)**

Dany podrozdział jest poświęcony omówieniu rozszerzeniu techniki GCN (z ang. *Graph Convolutional Networks*), stosowanej do analizy danych w formie grafów. Relacyjne Grafowe Sieci Konwolucyjne (z ang. *Relational Graph Convolutional Networks*), dają możliwość efektywnie analizować graficzne przedstawienia, gdzie występują tak zwane "relacje" między węzłami. Takie relacje określają powiązania jak i zależności, które występują między węzłami [31]. Przykładowo, w przypadku tworzenia grafów molekularnych, każdy atom cząsteczki, to węzeł, a połączenia między cząsteczkami to relacje [23].

Jeśli porównywać GCN i R-GCN, to główna różnica między nimi leży w sposobie uwzględniania relacji między węzłami w grafie. Zwykły model GCN zakłada że każdy węzeł ma stałą wagę reprezentowaną jako macierz. Oznacza to, że sieć traktuje wszystkie relacje między węzłami jednakowo. Także dany model nie posiada mechanizmu do wyrażania różnic w relacjach między węzłami, ponieważ zakłada, że wszystkie krawędzie są jednorodne [31].

Natomiast model R-GCN dynamicznie przypisuje wartości wag w zależności od typu relacji między węzłami, to pozwala na elastyczniejsze modelowanie relacji biorąc pod uwagę różnice w relacjach. Takie podejście modelu pozwala na bardziej precyzyjne odzwierciedlenie struktury graficznych reprezentacji. Dany model można wykorzystać dla generowania molekuł z uwzględnieniem struktury molekularnej oraz różnych typów wiązań [31].

Jako podsumowanie warto zaznaczyć że relacyjna graficzna sieć splotowa to technika, która umożliwia efektywnie analizować i przetwarzać graficzne przedstawienia z uwzględnieniem relacji między węzłami. W związku z tym, ten model może być używany do dokładnego modelowania, uwzględniając zróżnicowanie w rodzajach wiązań chemicznych [32].

## **4 PRZEGLĄD PRACY “DRUG MOLECULE GENERATION WITH VAE”**

W tym rozdziale omówiona zostanie praca badawcza o nazwie "Drug Molecule Generation with VAE". Jak nazwa pracy wskazuje, głównym celem omawianej pracy było stworzenie modelu, który mógłby generować nowe, potencjalnie skuteczne cząsteczki, biorąc pod uwagę ich struktury chemiczne i właściwości z wykorzystaniem sieci neuronowych, w szczególności zmodyfikowanej architektury wariacyjnego autoenkodera (VAE) [35].













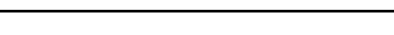
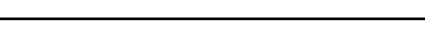
Model został wytrenowany na zbiorze danych zawierającym tysiące cząsteczek, a następnie był testowany na zbiorze danych, który nie był wykorzystywany podczas treningu. W pracy wykorzystano różne techniki i algorytmy do poprawy jakości generowanych cząsteczek, takie jak zastosowanie funkcji straty opartej na podobieństwie strukturalnym między generowanymi a rzeczywistymi cząsteczkami oraz modyfikacja architektury VAE [35].

### **4. 1 Często spotykane cząsteczki:**

Ze względu na to że w dalszej części pracy będą przedstawiane wyniki trenowania sieci, a także będą analizowane wyniki, na podstawie różnorodności molekuł produkowanych przez sieć neuronową w tej krótkiej części umieszczono opis, jak rozumieć wynik.

Poniżej przedstawiona tabela przedstawia proste związki chemiczne, tak zwane alkany. Dane cząsteczki często pojawiają się w wynikach trenowania sieci neuronowej wziętej z pracy o tytule: “DRUG MOLECULE GENERATION WITH VAE”.





Tabela 1. Lista wybranych przykładów alkanów

Nazwa węglowodoru	Model cząsteczki	Wzór sumaryczny
Metan	$\text{CH}_4$	$\text{CH}_4$
Etan		$\text{C}_2\text{H}_6$
Propan		$\text{C}_3\text{H}_8$
Butan		$\text{C}_4\text{H}_{10}$
Pentan		$\text{C}_5\text{H}_{12}$
Heksan		$\text{C}_6\text{H}_{14}$
Heptan		$\text{C}_7\text{H}_{16}$
Oktan		$\text{C}_8\text{H}_{18}$
Nonan		$\text{C}_9\text{H}_{20}$
Dekan		$\text{C}_{10}\text{H}_{22}$
Undekan		$\text{C}_{11}\text{H}_{24}$
Dodekan		$\text{C}_{12}\text{H}_{26}$
Tridekan		$\text{C}_{13}\text{H}_{28}$
Tetradekan		$\text{C}_{14}\text{H}_{30}$
Pentadekan		$\text{C}_{15}\text{H}_{32}$

Źródło: Opracowanie własne A. Vezdenetska na podstawie źródła[2]

Oprócz wyżej wspomnianych węglowodorów nasyconych, alkanów, często można spotkać również cząsteczki nienasyconych węglowodorów alkenów. Zasadnicza różnica między tymi związkami, polega na tym że alkany można zapisać ogólnym wzorem:  $C_nH_{2n+2}$ , gdzie:  $n$  to liczba atomów węgla w cząsteczce, a w tym czasie gdy alkeny można zapisać wzorem:  $C_nH_{2n}$ . Jak można wnioskować z opisu powyższych wzorów, widzimy że alkeny posiadają mniejszą ilość atomów wodoru (H), dzieje się tak ponieważ w alkenach występują podwójne wiązania między atomami węgla. W poniższej tabeli przedstawione zostały wybrane cząsteczki alkenów spotykanych w wynikach trenowanie sieci neuronowej [2].




Tabela 2. Lista wybranych przykładów alkenów

Nazwa węglowodoru	Model cząsteczki	Wzór sumaryczny
Eten		$C_2H_4$
Propen		$C_3H_6$
Buten		$C_4H_8$
Penten		$C_5H_{10}$

Źródło: Opracowanie własne A. Vezdenetska na podstawie źródła[2]

Także wśród wyników, często można spotkać alkiny. Ogólny wzór alkinów to  $C_nH_{2n-2}$ . W alkinach, między dwiema pewnymi atomami węgla występuje potrójne wiązanie. Oprócz wyżej wspomnianych węglowodorów nasyconych, alkanów, często można spotkać również cząsteczki nienasyconych węglowodorów alkenów. Zasadnicza różnica między tymi związkami





Tabela 3. Lista wybranych przykładów alkinów

Nazwa węglowodoru	Model cząsteczki	Wzór sumaryczny
Etyn		$C_2H_2$
Propyn		$C_3H_4$
Butyn		$C_4H_6$

Źródło: Opracowanie własne A. Vezdenetska na podstawie źródła [2]

Oprócz węglowodorów często można spotkać powtarzające się cząsteczki alkoholi. W poniższej tabeli cztery, przedstawiono, często spotykane struktury w wynikach. Wśród wzorów sumarycznych dla alkoholi, można również wydzielić pewny ogólny wzór  $C_nH_{2n+1}OH$  [2].

Tabela 4. Lista wybranych przykładów alkoholi

Nazwa alkoholu	Model cząsteczki	Wzór sumaryczny
Metanol		$CH_3OH$
Etanol		$C_2H_5OH$
Propanol		$C_3H_7OH$
Pentanol		$C_5H_{11}OH$

Źródło: Opracowanie własne A. Vezdenetska na podstawie źródła [2]

Wszystkie molekuly wymienione w tym podrozdziale, nie możemy uważać za unikalnie zbudowane cząsteczki. W tym rozdziale najczęściej mierzono procentowy współczynnik otrzymania unikalnych molekuł w porównaniu do najczęściej spotykanych cząsteczek.

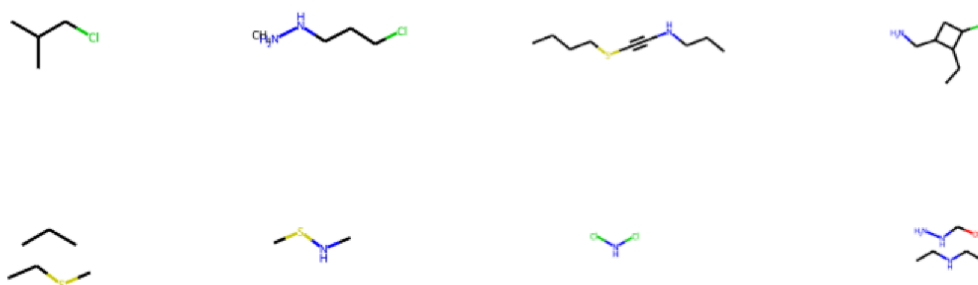
## 4. 2 Powtórzenie eksperymentu.

W pierwszym podejściu uruchomiono daną sieć neuronową, bez wprowadzania do niej jakichkolwiek zmian. Dane otrzymane po uruchomieniu sieci z pierwotnym kodem, pozwalają zrozumieć skuteczność wytrenowanego modelu podczas tworzenia molekuł, a także ocenić czas poświęcony na wytrenowanie modelu.

Podczas pierwszego uruchomienia modelu, odnotowano że wytrenowanej jednej epoki trwa ponad 4 minuty, maksymalnie trening trwał do 5 minut. Co oznacza że dla wytrenowania 10 epok, potrzebny czas ponad 40 minut. Otrzymane próbki były dosyć podobne do próbek przedstawionych w notatniku załączonym do pracy “DRUG MOLECULE GENERATION WITH VAE” na platformie keras. Warto zaznaczyć że procentowa różnica unikalnych molekuł, czyli takich, które nie powtarzały by wzorce innych molekuł, jest rażąco niższa w porównaniu do ilości molekuł które cały czas się powtarzały, a były to głównie długie łańcuchy węglowe.

Podczas zwykłego treningu sieci, udało się uzyskać przykłady około sto sześćdziesiąt graficznych przedstawień molekuł. Większość otrzymanych molekuł przedstawiała z siebie jedynie długie łańcuchy węglowodorowe, w których występował tylko łańcuch główny [32].

Poniżej przedstawiono unikalne molekuły, czyli takie które wyróżniały się spośród wygenerowanych długich łańcuchów węglowych, uzyskanych podczas treningu. Po przeliczeniu uzyskanych molekuł ogólnie w odniesieniu do otrzymanych molekuł unikalnych, można stwierdzić, że system ten jest mało wydajny, gdyż procentowa ilość otrzymanych unikalnych molekuł, stanowi około szesnastu procent.



Rysunek 4. Unikalne molekuły, zebrane z kilku epok

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [35]

W celu sprawdzenia czy jest możliwość poprawy danego wyniku, poddane analizie zostały wszystkie otrzymane molekuly, otrzymane w czasie treningu dziesięciu epok, przy użyciu standardowego kodu zapożyczonego ze strony keras. Analiza otrzymanych wyników pozwoliła wyjaśnić że najczęściej procentowa ilość uzyskanych unikalnych molekuł w czasie trenowania epoki wynosiła w okolicach ośmiu, maksymalnie dziewięciu procent.

### 4.3 Regulacja zmiennej *dropout rate* w enkoderze

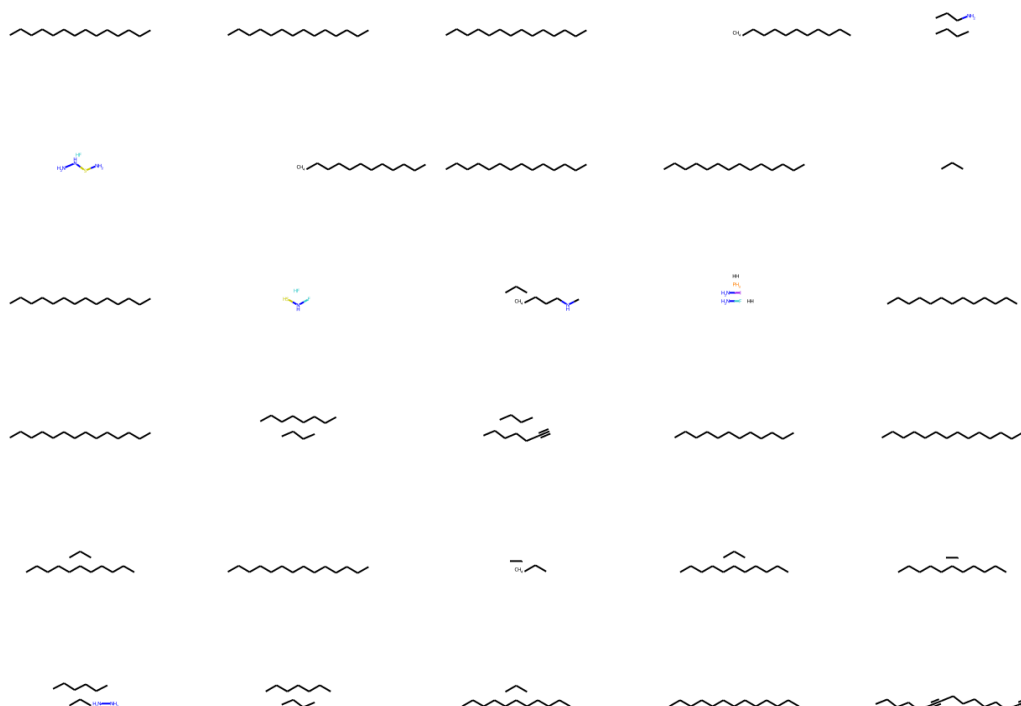
W pracy "Drug Molecule Generation with VAE" autorzy stosują dropout w enkoderze jako regularyzację, aby zapobiec wystąpienia zjawiska *overfittingu*. Problem z przetrenowaniem sieci często pojawia się, gdy model zbyt dobrze dopasowuje się do danych treningowych, co prowadzi do utraty zdolności do generalizacji na nowych danych [8].

Podczas uczenia VAE, sieć neuronowa jest trenowana na danych treningowych poprzez minimalizację różnicy między rozkładem latentnym (czyli rozkładem zmiennych ukrytych), który jest wynikiem enkodera, a rozkładem danych oryginalnych, które są rekonstruowane przez dekodera. Kluczowym celem uczenia VAE jest nauczenie modelu generować próbki podobne do wejściowych danych treningowych, tak żeby te próbki mogły stać się nowym przykładem, które są probabilistycznie zgodne z rozkładem latentnym [11].

Kiedy VAE zostanie zbyt mocno dopasowany do danych treningowych, może to spowodować *overfitting* sieci neuronowej. Model zapamiętuje szczegóły i szumy w danych treningowych, które nie mają istotnego znaczenia dla ogólnego rozkładu, co prowadzi do przesadnego dopasowania. Takie dopasowanie może powodować, że model będzie miał trudności z generalizacją na nowe dane, które różnią się od danych treningowych [22; 11].

Funkcja *dropout* działa przez losowe usuwanie niektórych jednostek (neuronów) z sieci neuronowej podczas treningu. Im mniej jednostek zostanie usuniętych, tym więcej jednostek będzie mogło „spiskować” między sobą, aby ostatecznie dopasować zestaw treningowy. W kodzie podstawowym, zmienna *dropout\_rate* w enkoderze została ustawiona na 0, co oznacza, że żadna z jednostek nie zostanie usunięta, a zatem żadna regularyzacja nie będzie stosowana [21].



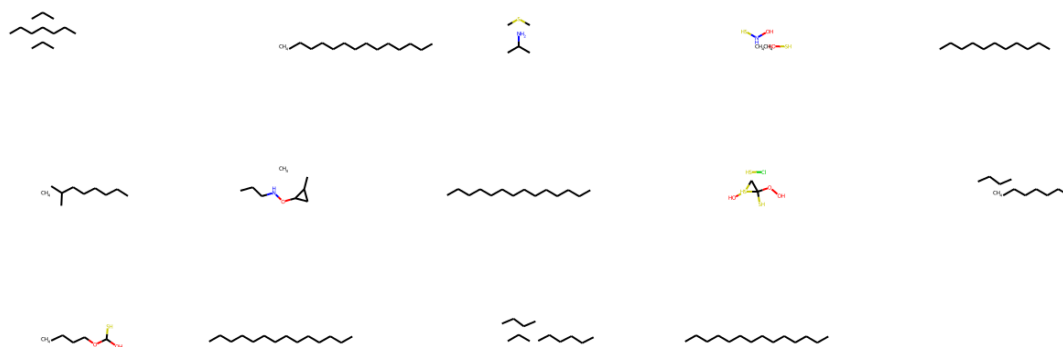


Rysunek 5. Wynik dla wartości zmiennej `dropout_rate` 0.0

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [35]

Rysunek pięć, przedstawia wynik otrzymany w jednej z epok przy ustawieniach zmiennej `dropout_rate=0.0`, gdzie enkoder nie był poddany regulacji dropout. To oznacza, że enkoder miał większą liczbę aktywnych jednostek, a zatem może uczyć się z większą precyzją, ale dane ustawienie może również spowodować *overfitting* sieci neuronowej. Analiza otrzymanego wyniku wskazuje na to że zostało otrzymano około dwudziestu trzech procent unikalnych molekuł w ciągu szkolenia jednej epoki.

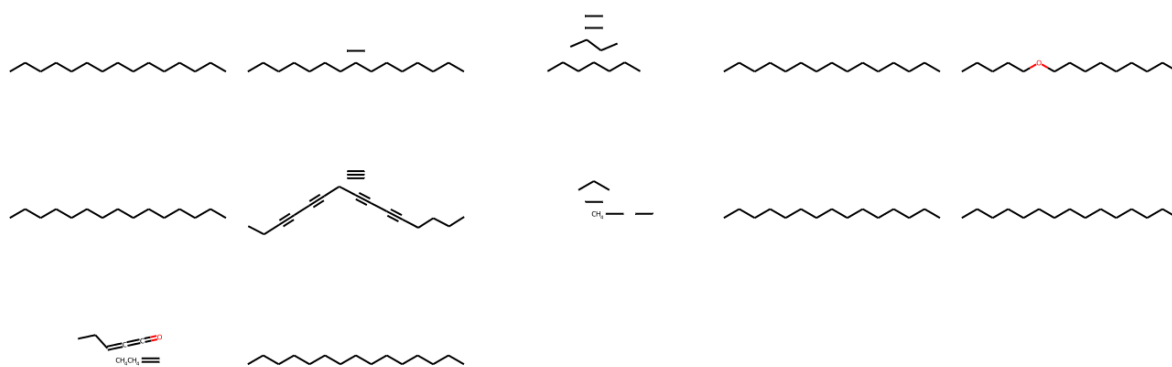
Ustawienie danej zmiennej `dropout_rate` na zero niekoniecznie musi prowadzić do *overfittingu*, ponieważ dane zjawisko jest zależne od wielu czynników. Lecz w celu badawczym zdecydowano podwyższyć dany parametr. Na rysunkach poniżej zostały zaprezentowane wybrane epoki uzyskane podczas treningu różnych wartości tej zmiennej.



Rysunek 6. Wynik dla wartości zmiennej *dropout\_rate* 0.1

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [35]

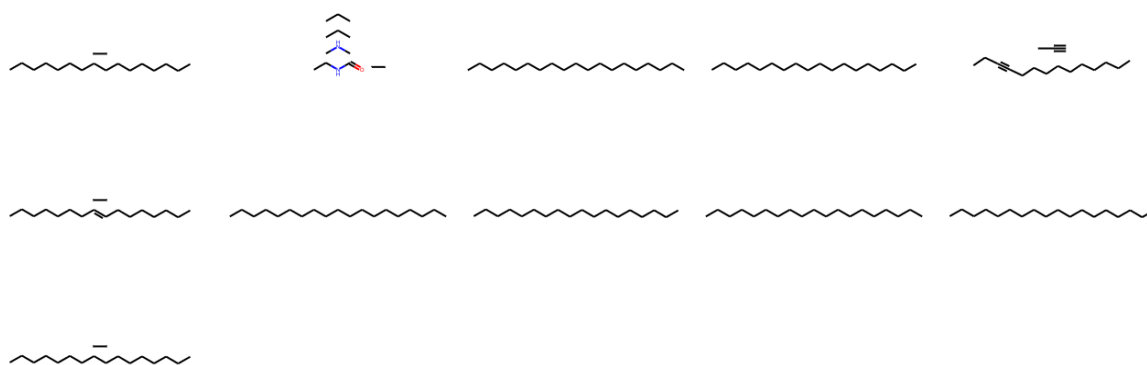
Podczas uzyskania wyników po zwiększeniu wartości *dropout rate* do zero przecinek jeden, prawie we wszystkich epokach zauważono przyrost wygenerowanych unikalnych molekuł, większości przykładów procent unikalnych molekuł do wszystkich pozostałych. Przykładowo powyższy rysunek ilustruje dany przyrost unikalnych molekuł generowanych podczas kształcenia jednej epoki udało się otrzymać czterdzieści trzy procent unikalnych cząsteczek.



Rysunek 7. Wynik dla wartości zmiennej *dropout\_rate* 0.2

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [35]

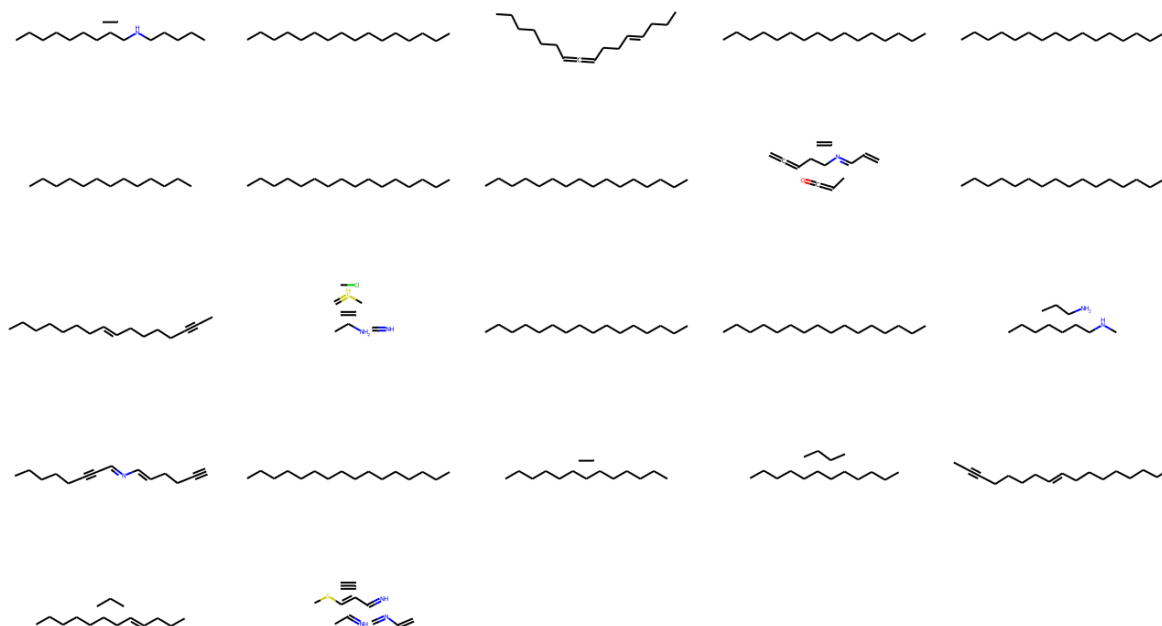
Dalsze zwiększenie wartości zmiennej *dropout rate* do zero przecinek dwa, niestety nie przyniosło pożądanego skutku. Procentowa ilość unikalnych molekuł w tym trenowaniu waha się w okolicy dwudziestu procent. Powyższy rysunek siedem, przedstawia wynik, w którym ćwierć otrzymanych cząsteczek są unikalne.



Rysunek 8. Wynik dla wartości zmiennej `dropout_rate` 0.3

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [35]

Następne podwyższenie wartości zmiennej *dropout rate* do zero przecinek trzy, nie przyniosło pożądanych skutków. Analiza wyniku otrzymanej w jednej z epok, podczas trenowania sieci z ustawieniem wartości zmiennej *dropout rate* zero przecinek trzy wskazują na to że podczas trenowania otrzymano około dziewięciu procent unikalnych molekuł. Podczas danego treningu sieci w innych epokach wynik wahał się pomiędzy dziesięć, a dwadzieścia procent.

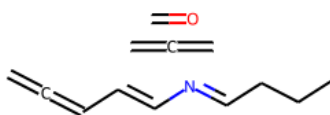


Rysunek 9. Wynik dla wartości zmiennej `dropout_rate` 0.4

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [35]

Ostatnia wartość, którą udało się sprawdzić, było podwyższenie wartości zmiennej *dropout rate* do zero przecinek cztery, powyższy rysunek dziewięć, przedstawia wynik otrzymany podczas szkolenia jednej z dziesięciu epok. Analiza powyższego rysunku wskazuje na to, że w trenowaniu tej epoki, udało się otrzymać czterdzieści procent unikalnych molekuł.

Ze względu na to, że próba wyszkolenie dziesięciu epok, przy wartości zmiennej *dropout rate* zero przecinek pięć okazała się niemożliwa, gdyż sieć została przeciążona taką wartością. Trenowanie udało się przeprowadzić tylko dla pięciu epok, z daną wartością zmiennej, w związku z powyższym udało się otrzymać wynik przedstawiony na poniższym rysunku.



Rysunek 10. Wynik dla wartości zmiennej *dropout\_rate* 0.5

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [35]

Tak jak przedstawia powyższy rysunek dziesięć, jest to wynik otrzymany podczas szkolenia jednej z pięciu epok. W ciągu szkolenia jednej epoki otrzymywano zwykle wzór jednej lub dwóch unikalnych molekuł. Mimo to, że podczas ostatniego eksperymentu podczas szkolenia każdej epoki była możliwość otrzymać jedną unikalną molekułę, takie szkolenie jest zbyt czasochłonne, zwłaszcza jeśli nie koniecznie uda się wygenerować molekułę o pożądanym właściwościach.

#### 4. 3. 1 Wnioski podrozdziału:

Dane badanie wykazało, że brak regulacji wartości zmiennej *dropout rate* pierwotnego modelu doprowadził do *overfittingu* sieci neuronowej. Poniższa tabela pięć wyświetla dokładną ilość procent unikalnych cząsteczek, otrzymanych w trakcie trenowania wybranej architektury sieci.

Tabela 5. Wyniki testów wykonanych dla podrozdziału 4. 2

<i>Dropout rate</i>	<i>dropout</i> 0.0	<i>dropout</i> 0.1	<i>dropout</i> 0.2	<i>dropout</i> 0.3	<i>dropout</i> 0.4	<i>dropout</i> 0.5
Procent unikalnych molekuł	23%	43%	25%	10%	36%	100%

Źródło: Opracowanie własne A. Vezdenetska na podstawie otrzymanych wyników

Analizując wyniki zawarte w tabeli, optymalną wartością dla zmiennej *dropout rate* wyniosła zero przecinek jeden. Gdyż przy tej wartości zmiennej *dropout rate* udało się osiągnąć najwyższy procent unikalnych molekuł.

Analizując problem nadmiernego dopasowania, można stwierdzić że najlepszy wynik otrzymamy, przy wyłączeniu dziesięciu procent neuronów w trakcie trenowania sieci neuronowej. Najgorszy wynik otrzymano przy wartości zmiennej *dropout\_rate* gdzie wyłączone zostało trzydzieści procent neuronów.

Warto zaznaczyć że tabela, także zawiera informację o wartości zmiennych *dropout\_rate* równej zero przecinek cztery oraz zero przecinek pięć. Ilość unikalnych molekuł przy architekturze zero przecinek cztery jest wyższa niż przy zero przecinek trzy. A przy wartości zmiennej zero przecinek pięć otrzymano były wygenerowane tylko unikalne molekuły. Wyniki te są mylące, z powodu tego że sieć została przeciążona przy tych wartościach i zaczęła dopasowywać generowane dane do szumu, a nie do danych z zestawu uczącego [25].

Także w czasie prowadzenia danego badania odnotowano duży wzrost zużycia pamięci operacyjnej, po wprowadzeniu nowej wartości dla zmiennej *dropout\_rate* znajdującej się w enkoderze, co czasem skutkowało awarią sieci neuronowej i była konieczność rozpoczęcia treningu na nowo. Wszystkie wyniki, otrzymane w trakcie trenowania, można będzie odnaleźć w odpowiednim repozytorium na github pod poniższym linkiem:

## 4. 4 Regulacja zmiennej *dropout rate* w dekodерze

Podobnie jak w poprzednim podrozdziale, zmienna *dropout rate* została zastosowana w dekodерze, jako regulację również w celu zapobiegania *overfittingu* trenowanej sieci.

Dana zmienna pozwala losowo usunąć niektóre neurony z sieci neuronowej podczas treningu [15]. W podstawowym kodzie dana zmienna ma wartość zero przecinek dwa, to oznacza że każdy neuron sieci może zostać usunięty podczas trenowania z dwudziesto procentowym prawdopodobieństwem [1]. Tak długo, jak wartość danej zmiennej nie osiągnie zera, oznacza to, że wybrana sieć podlega wspomnianej regulacji, co implikuje mniejszą liczbę aktywnych jednostek w dekodерze podczas treningu. Takie wyłączenie neuronów może pomóc w zapobieganiu *overfittingu* i zwiększyć zdolność sieci do generalizacji modelu.

W celach badawczych, także jak i w poprzednim eksperymencie z enkoderem, zmianie ulegała tylko zmienna znajdująca się w dekodерze, natomiast w enkoderze ta zmienna *dropout\_rate* pozostała taka sama jak i w kodzie podstawowym czyli równa zero.

Ze względu na to że wyniki powinny być jednorodne oraz spójne, wykonano próbę uruchomienia sieci neuronowej, gdzie wartości *dropout\_rate* jest równe zero, czyli, innymi słowami zmienna *dropout\_rate* nie ulega regularyzacji ani w enkoderze ani w dekodерze. Brak wyżej wspomnianych regulacji prowadził do znacznego przeuczenia sieci, na poniższym rysunku, przedstawiono wynik otrzymany podczas trenowania jednej z dziesięciu epok.

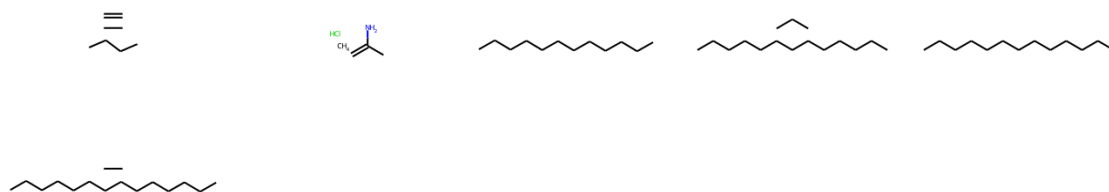


Rysunek 11. Część wyniku dla wartości zmiennej dekodera *dropout\_rate* 0.0

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [35]

Otrzymane graficzne przedstawienia molekuł otrzymanych w czasie danego szkolenia sieci neuronowej, zostały poddane analizie. W wyniku braku regulacji za pomocą zmiennej *dropout\_rate* w trakcie trenowania sieci udało się uzyskać około szesnaście procent unikalnych molekuł.

W następnym uruchomieniu badanej sieci neuronowej, wartości zmiennej *dropout\_rate* nadano wartość zero przecinek jeden, w związku z czym uzyskano trzydzieści sześć procent unikalnych molekuł w wyniku trenowania dziesięciu epok. Przykład wyniku otrzymanego w trakcie treningu jednej z epok, znajdują się na poniższym rysunku jedynastym. Porównując dany wynik z poprzednim, otrzymanym przy braku regulacji za pomocą badanej zmiennej, widoczny jest procentowy przyrost ilości unikalnych molekuł na dwanaście procent.

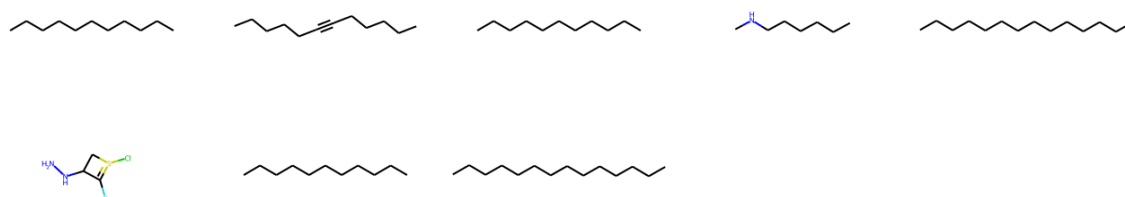


Rysunek 12. Wynik dla wartości zmiennej dekodera *dropout\_rate* 0.1

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [35]

Następna badana wartość zmiennej *dropout rate* była zero przecinek dwa. Dana wartość, była wartością podstawową, od której rozpoczął się eksperyment dla wartości zmiennej *dropout\_rate* w poprzednim podrozdziale. Powtórne uzyskanie tego samego wyniku, w odniesieniu do procentu unikalnych molekuł wygenerowanych przez sieć neuronową, w wyniku trenowania dziesięciu epok pozwoli mieć klarowny wgląd do analizowanych danych.

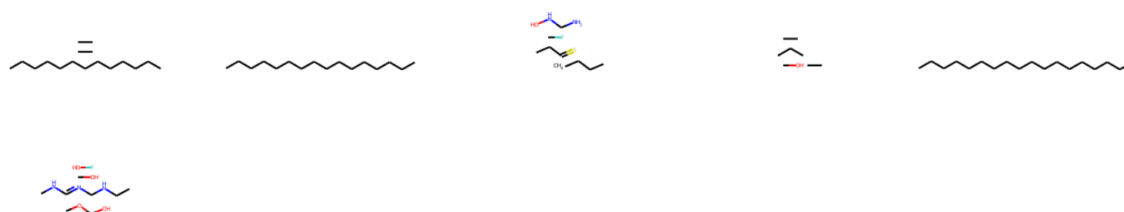
Na poniżej umieszczonym rysunku trzynaście widziane schematy molekuł otrzymanych podczas szkolenia jednej epoki. Analiza wyników otrzymanych w trakcie szkolenia dziesięciu epok sieci neuronowej wskazują na to że podczas trenowania udało się wygenerować około siedemnastu procent różnorodnych cząsteczek.



Rysunek 13. Wynik dla wartości zmiennej dekodera  $dropout\_rate$  0.2

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej[35]

Dalsze zwiększenie wartości zmiennej  $dropout\_rate$  do zero przecinek trzy, pozwoliło uzyskać około dwudziestu procent unikalnych molekuł w wyniku trenowania dziesięciu epok. Na poniżej umieszczonym rysunku czternaście, przedstawiono schematy molekuł otrzymanych w trakcie szkolenia jednej epoki, podczas nauki sieci neuronowej.

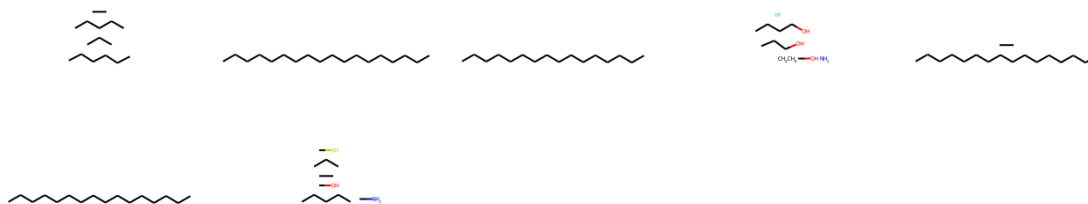


Rysunek 14. Wynik dla wartości zmiennej dekodera  $dropout\_rate$  0.3

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [35]

Kolejna wartość zmiennej  $dropout\_rate$  - zero przecinek cztery, taka wartość oznaczała że podczas nauki czterdzieści procent neuronów, mogą być losowo wyłączone, niestety tak wielka ilość pominiętych neuronów, doprowadziła do przeuczenia się sieci w wyniku podczas tego treningu, udało się otrzymać tylko około pięciu procent unikalnych molekuł. Poniższy rysunek piętnaście przedstawia graficzną reprezentację cząsteczek otrzymanych podczas szkolenia jednej z dziesięciu epok.

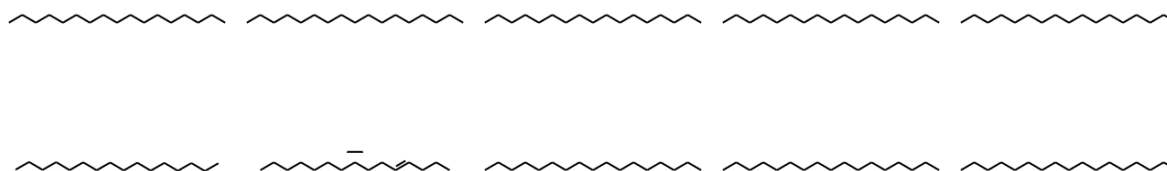




Rysunek 15. Wynik dla wartości zmiennej dekodera dropout\_rate 0.4

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [35]

Wartość zmiennej *dropout\_rate* zero przecinek pięć także została zweryfikowana, lecz tak jak i z poprzednią badaną wartością, nie przyniosło to pożądanych skutków, wyłączenie połowy neuronów z procesu nauki stało przyczyną takich wyników.



Rysunek 16. Wynik dla wartości zmiennej dekodera dropout\_rate 0.5

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [35]

Poniższy rysunek szesnaście przedstawia wynik trenowania jednej epoki. Sieć generuję głównie długie łańcuchy węglowe, w których występują tylko łańcuch główny. Procentowa ilość otrzymanych unikalnych molekuł w wyniku danego trenowania stanowi około jednego procentu.

#### 4. 4. 1 Wnioski podrozdziału:

Biorąc pod uwagę zmiany w procentowej ilości otrzymanych unikalnych molekuł, można twierdzić że najlepszy wynik był otrzymany, kiedy odłączono dziesięć procent losowych neuronów. Gdyż podczas tego trenowania sieci neuronowej udało się pozyskać ponad trzydzieści procent różnorodnych cząsteczek w wyniku trenowania dziesięciu epok.

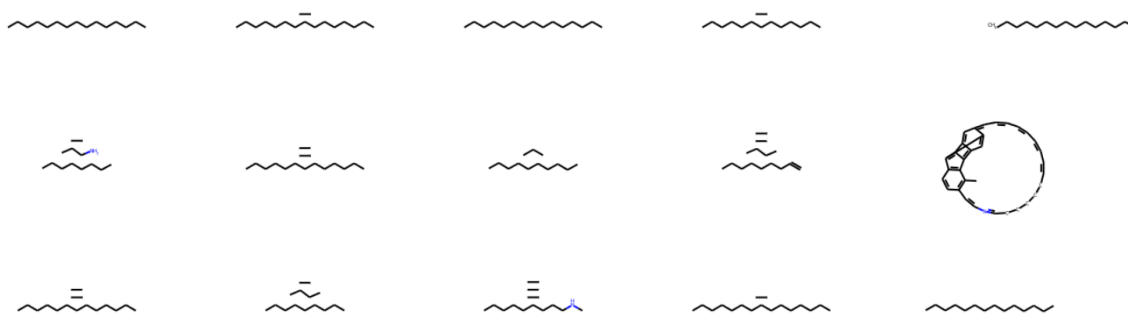
Tabela 6. Wyniki testów wykonanych dla podrozdziału 4. 3

<i>Dropout rate</i>	<i>dropout</i> 0.0	<i>dropout</i> 0.1	<i>dropout</i> 0.2	<i>dropout</i> 0.3	<i>dropout</i> 0.4	<i>dropout</i> 0.5
Procent unikalnych molekuł	16%	36%	17%	21%	5%	1%

Źródło: Opracowanie własne A. Vezdenetska na podstawie otrzymanych wyników

W tabeli pięć zostały zebrane wyniki otrzymane w tym podrozdziale. Analizując przedstawione dane, można stwierdzić że optymalna wartość zmiennej *dropout\_rate* to zero przecinek jeden. Przy tym warto zauważyć że przy wartości zmiennej *dropout\_rate* zero przecinek trzy, pojawia się rzekomy wzrost ilości różnorodnych cząsteczek, jest to sytuacja podobna tej, która pojawiła się w enkoderze przy wartości zero przecinek cztery, kiedy generowane dane, zaczęły dopasowywać się do szumu. Ale jak świadczą dalsze wyniki, wyłączenie jeszcze większej ilości neuronów dekodera powoduje to że sieć przestaje się uczyć.

Dla podsumowania wyników obu podrozdziałów poświęconych weryfikacji optymalnej wartości hyperparametru *dropout\_rate* wytrenowano sieć neuronową za pomocą odnalezionych wartości optymalnych. Czyli dla enkodera, tak i dla dekodera, losowo zostaną wyłączone dziesięć procent neuronów.



Rysunek 17. Część wyniku, eksperymentu 4. 4. 1

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [35]

Rysunek siedemnasty przedstawia wynik, otrzymany podczas trenowania dziesięciu epok, gdzie były zastosowane parametry o najlepszych wartościach zmiennej

*dropout\_rate* dla enkodera 0.1 i dekodera 0.1, co miało zwiększyć oczekiwaną procentową ilość unikalnych molekuł, otrzymanych w trakcie trenowania sieci neuronowej.

Lecz niestety wynik okazał się odwrotnym, dane ustawienia wartości, dają najlepsze wyniki kiedy działają osobno. Podczas tej próby udało się otrzymać około pięćdziesiąt procent unikalnych molekuł. W związku z tym w dalszej części powtórnych badań wykorzystane zostały dane ustawienia dla zmiennej *dropout\_rate*. Wszystkie wyniki, otrzymane w trakcie trenowania, można będzie odnaleźć w odpowiednim repozytorium na github pod poniższym linkiem:

<https://github.com/anksunamoona/Praca-Dyplomowa/tree/main/PRZEGLĄD%20RACY%20“DRUG%20MOLECULE%20GENERATION%20WITH%20VAE”/4.3.%20Dekoder%20dropout>

## 4. 5 Zmiana optymalizatora

Adam to optymalizator, który wykorzystuje badana sieć neuronowa. Algorytm ten został przedstawiony w pracy “*ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION*” opublikowanej w dwa tysiące czternastym roku [20]. Adam to algorytm optymalizacji stosowany do ulepszania procesu głębokiego uczenia, na przykład w sieciach neuronowych. Adam działa na podstawie tak zwanej stochastycznej funkcji celu, czyli takiej, która może się zmieniać w trakcie procesu uczenia. Te zmiany są obliczane gradientowo, co pomaga określić, w którym kierunku i z jaką siłą powinny być aktualizowane wagi w trakcie trenowania sieci neuronowej, co z kolei przekłada się na otrzymanie lepszych wyników, ponieważ sieć uczy się lepiej wykonywać swoje zadanie [20].

Imię „Adam” wywodzi się od wyrażenia „momenty adaptacyjne”, co sugeruje, że algorytm uwzględnia adaptacyjne oszacowania momentów pierwszego i drugiego rzędu w celu optymalizacji procesu uczenia maszynowego, takiego jak trening sieci neuronowych. Tak jak opisano powyżej Adam jest tak zaprojektowany, aby uwzględniać różne aspekty danych i błędy systematyczne, a także dostosowywać tempo uczenia w odpowiedni sposób, aby znaleźć optymalne rozwiązania w trakcie uczenia sieci neuronowej [4]. Ze względu na to że taka metoda jest prosta do wdrożenia, wydajna obliczeniowo, ma niewielkie wymagania dotyczące pamięci, jest

niezmienna w przypadku przeskalowania gradientów po przekątnej i dobrze nadaje się do problemów dużych pod względem danych i/lub parametrów, sprawiło to że stała się dość popularna i często wykorzystywana w dziedzinie uczenia maszynowego, zwłaszcza w przypadku złożonych i wymagających zadań treningowych [20].

W pracy naukowej, gdzie po raz pierwszy był przedstawiony algorytm Adam, został on porównany z takim algorytmem jak SGD z pędem, a także nawiązuje podobieństwo między tymi algorytmami, nawiązując do SGD jako wydajnej oraz skutecznej metody optymalizacji, która odegrała kluczową rolę w sukcesach uczenia maszynowego, zwłaszcza w dziedzinie głębokiego uczenia [20]. Algorytm stochastycznego zniżania gradientowego, znany także pod nazwą SGD, SGD z pędem lub SGD z momentum, jest skutecznym rozwiązaniem problemu przetwarzania dużych zbiorów danych treningowych, co pozwala na trening głębokich modeli uczenia maszynowego w bardziej wydajny sposób [30].

Algorytm optymalizacji, który został wykorzystany w celu zwiększenia wydajności sieci neuronowej w dalszej części badań to SGD. Wybrany algorytm działa na zasadzie wykorzystania małych losowo wybranych próbek danych treningowych w każdym kroku, co pozwala na przyspieszenie treningu. Na każdym kroku uczenia sieci neuronowej obliczany jest gradient, czyli kierunek, w którym wartości wag w modelu powinny się zmieniać, na podstawie pobranej małej próbki danych. Następnie wagi modelu są aktualizowane, aby lepiej pasować do tych danych, a szybkość uczenia się kontroluje, jak duże są te aktualizacje wag. Takie aktualizacje w trakcie treningu sprawiają że dany algorytm można wykorzystać, aby trenować modele na dużych zbiorach danych, takich jak sieci neuronowe. Także z powodu tego że gradient nie musi być obliczany dla całego zbioru, sprawia że dany model jest znacznie bardziej efektywnie obliczeniowo [30].

Niestety podczas próby trenowania sieci neuronowej za pomocą danego optymalizatora, nie udało się uzyskać żadnych modeli, co wskazuje na to że dany optymalizator nie pasował do trenowania wybranej sieci neuronowej.

Następnym wybranym algorytmem do porównania w badanej sieci neuronowej był Adagrad. Nazwa tego algorytmu pochodzi od adaptacyjnego gradientu, co wskazuje na to, że na każdym kroku uczenia sieci neuronowej obliczany jest gradient, czyli kierunek, w którym wartości wag w modelu powinny się zmieniać [12]. Jednak optymalizator Adagrad działa wolniej w porównaniu do optymalizatorów, takich jak Adam i SGD [20]. Główny wpływ na szybkość trenowania modelu ma adaptacyjność

kroków nauki, która jest jedną z kluczowych cech algorytmu Adagrad. Oznacza to, że dla każdego parametru modelu, algorytm dostosowuje tempo uczenia się na podstawie historii gradientów tego parametru. Parametry, które mają duże gradienty, otrzymują mniejsze aktualizacje, podczas gdy parametry z małymi gradientami otrzymują większe aktualizacje. Dana adaptacyjność kroków nauki, pomaga zniwelować problem z tak zwanym “wybuchającym gradientem” w głębokich sieciach neuronowych. Wybuchający gradient, wiąże się z tym że w kolejnych iteracjach wartości wag mogą rosnać lub maleć znacznie szybciej, niż jest to pożądane. Więc takie zjawisko może prowadzić do niestabilności procesu uczenia się i może uniemożliwić zbieżność modelu do optymalnego rozwiązania [12].

Niestety zarówno, jak i podczas próby trenowania sieci neuronowej za pomocą optymalizatora SGD, podczas trenowania sieci z użyciem optymalizatora Adagrad próba uzyskania nowych modeli skończyła się niepomyślnie, co wskazują na to że dany optymalizator nie pasował do trenowania wybranej sieci neuronowej.

Następnym algorytmem optymalizacji, który został użyty by sprawdzić czy uda się za pomocą danego optymalizatora uzyskać lepsze wyniki, a także by sprawdzić czy będzie on pasował do badanej sieci neuronowej, był wybrany algorytm Adadelta. Dany algorytm umożliwia dostosowywanie tempa nauki modelu w czasie rzeczywistym, w zależności od miary postępów podczas nauki. Także model ten dopasowuje się samodzielnie, w związku z tym nie ma potrzeby ręcznego poprawiania parametrów, takich jak tempo uczenia się. Dodatkowo, algorytm jest odporny na trudności, które mogą wystąpić podczas trenowania modeli w uczeniu maszynowym, na przykład takie jak zakłócone dane wejściowe (nieuporządkowane). Także wybrany algorytm ma wysoką efektywność obliczeniową, gdyż korzysta tylko z gradientów, czyli prostej informacji na temat tego, jak bardzo model się zmienia [34].

Niestety podczas próby trenowania sieci neuronowej za pomocą optymalizatora Adadelta, nie udało się uzyskać żadnych modeli, co wskazują na to że dany optymalizator nie pasował do trenowania wybranej sieci neuronowej.

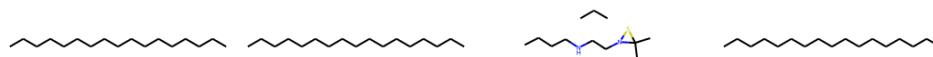
Ostatni optymalizator który został wykorzystany w celach badawczych był Nadam. Nazwa danego algorytmu pochodzi z języka angielskiego, a dokładnie “*Nesterov-accelerated Adaptive Moment Estimation*”. Nadam jest modyfikacją popularnego optymalizatora Adam, o którym była mowa na początku podrozdziału, poświęconemu regulacji zmiennej dropout rate w dekodzie. Optymalizator Nadam wprowadza poprawki do działania optymalizatora Adam, a dokładniej wykorzystuje

technikę przyspieszony gradient Niestierowa (z ang. *Nesterov Accelerated Gradient* - NAG) w celu poprawy jego stabilności i skuteczności.

Algorytm optymalizacji Nadam używany w maszynowym uczeniu do dostosowywania wag sieci neuronowych podczas treningu. W skrócie opisując jego działalność rozpoczyna się od inicjalizacji wag modelu i dwóch zbiorów parametrów: momentów (z ang. *moments*)  $m_t$  oraz momentów adaptacyjnych  $n_t$ , które początkowo są ustawione na zero. Podczas każdej iteracji treningowej obliczany jest gradient funkcji straty względem wag modelu [28].

Algorytm używa momentów ( $m_t$ ) do śledzenia kierunku, w którym wagi modelu się zmieniają. Moment adaptacyjny ( $n_t$ ) jest wykorzystywany do dostosowania współczynnika uczenia dla każdego wagowego parametru, a także zapobiegania niestabilności w trakcie uczenia. Oznacza to, że przed obliczeniem gradientu, moment adaptacyjny jest aktualizowany w kierunku, w którym mogą znajdować się wagi następnej iteracji. Na podstawie  $m_t$  i  $n_t$ , algorytm nadaje nowe wartości wagom modelu, uwzględniając obliczone gradienty, momenty i momenty adaptacyjne. Następnie obliczenie gradientu oraz aktualizacja  $m_t$  i  $n_t$ , a także aktualizacja wag, są powtarzane przez określoną liczbę epok treningowych lub do momentu osiągnięcia zadanego kryterium zakończenia treningu. Wszystkie wyżej wymienione kroki mają za zadanie zapewnić skuteczność i stabilność algorytmu Nadam w czasie treningu głębokich sieci neuronowych [40].

Poniżej przedstawiony rysunek osiemnaście przedstawia wynik otrzymany podczas szkolenia jednej z dziesięciu epok, na wybranych parametrach, a mianowicie przy wartościach zmiennej *dropout\_rate* 0.1 dla enkodera oraz 0.3 dla dekodera w trakcie korzystania z optymalizatora Nadam.



Rysunek 18. Wynik dla optymalizatora Nadam

Źródło: Opracowanie własne A.Vezdenetska na podstawie sieci neuronowej[35]

Analiza otrzymanych schematów w trakcie szkolenia sieci neuronowej na wskazanych parametrach ujawnia, że dany optymalizator nie jest najlepszym wyborem

dla badanej sieci, gdyż procentowa ilość unikalnych molekuł stanowi około piętnaście procent.

#### 4. 5. 1 Wnioski podrozdziału:

Podstawowy optymalizator Adam, który został wykorzystany w kodzie źródłowym jest najbardziej dopasowany do wykonania zadania danej sieci neuronowej. Porównując dany optymalizator do pozostałych, wykorzystanych podczas badań. Wszystkie wyniki, otrzymane w trakcie trenowania z innymi optymalizatorami, można będzie odnaleźć w odpowiednim repozytorium na github pod poniższym linkiem:

<https://github.com/anksunamoona/Praca-Dyplomowa/tree/main/PRZEGLĄD%20PACY%20“DRUG%20MOLECULE%20GENERATION%20WITH%20VAE”/4.4.%20Optimizers>

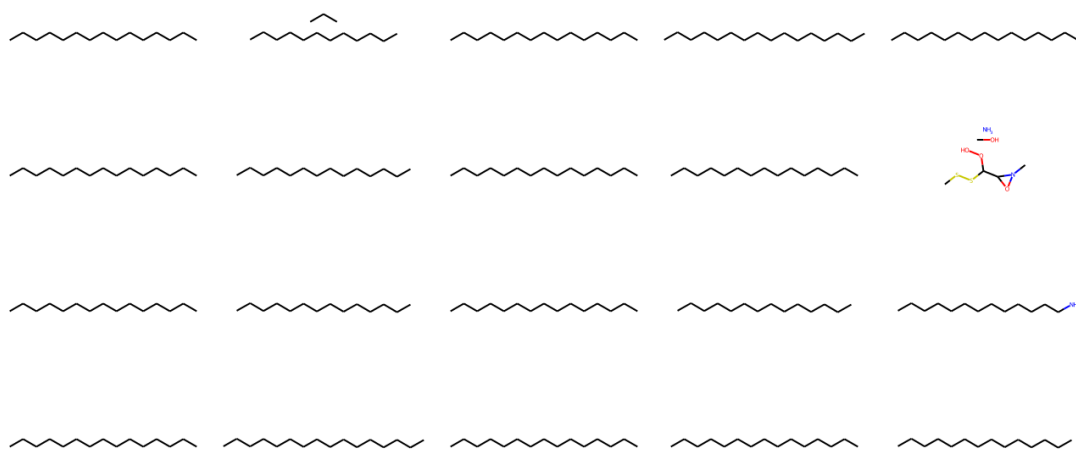
#### 4. 6 Regulacja zmiennej *batch size*

Zmienna *batch size* określa rozmiar wsadu danych treningowych. Dostosowanie tej wartości pozwala zmieniać rozmiar mini-partii danych (wsadu), co może mieć wpływ na tempo uczenia się i zużycie pamięci sieci neuronowej. Wybór odpowiedniego rozmiaru wsadu (*batch size*) ma istotne znaczenie podczas szkolenia sieci i może mieć wpływ na dokładność oszacowania gradientu. Większe znaczenie zmiennej (*batch size*), często zapewnia dokładniejsze oszacowanie gradientu, co z kolei przekłada się na skrócenie czasu treningu, ale zbyt wysoka wartość może również skutkować zwiększeniem zużycia pamięci. Przy tym większa ilość wsadu może prowadzić do znacznego skokowego zachowania funkcji kosztu i trudności w zbieżności modelu, także zbyt mały rozmiar wsadu może prowadzić do niestabilnego treningu [3].

W tym eksperymencie początkową wartością w kodzie podstawowym sieci neuronowej była wartość zmiennej *batch size* 100. W trakcie badań wartość ta ulegała zmianom. Ze względu na to że dana zmienna może przyjąć dowolną wartość, najpierw próbowano zmieniać tę wartość o 100, lecz warto zaznaczyć że w literaturze naukowej było zaznaczone że najlepiej model trenujący odreagowuje na znaczenia wartości potęgi liczby dwa. Z tego powodu zbadano następujące wartości zmiennej: 32, 64, 128, 256, 512.

Opis otrzymanych wyników będzie opisywany od wartości pięćset dwanaście, pierwsze trzy epoki były wygenerowane bez problemu, gdyż były w nich generowane liczbowe wartości strat, lecz w trakcie treningu czwartej epoki wystąpił problem z procesem szkoleniowym, gdyż strata przyjmując wartość „nan” (nie liczbową wartość). Przez napotkanie owego problemu w trakcie szkolenia, nie udało się wygenerować wyniku wartego prezentowania się.

Analiza otrzymanych wyników podczas szkolenia badanego modelu wykazała że procentowa ilość uzyskanych unikalnych molekuł w czasie trenowania epoki wynosiła w okolicach szesnastu procent.

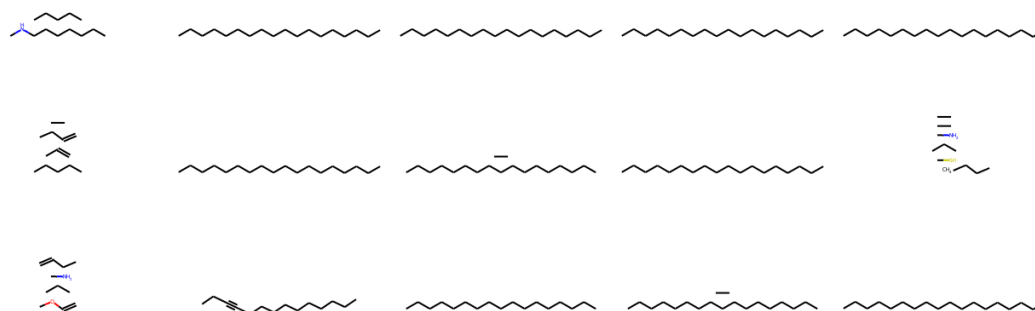


Rysunek 19. Wynik przy wartości zmiennej batch size 256

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [35]

Kolejna wartość zmiennej *batch size*, która została objęta treningiem była wartość sto dwadzieścia osiem. Poniżej przedstawiony rysunek osiemnaście przedstawia fragment wyniku szkolenia jednej z epok otrzymanych w trakcie szkolenia sieci za pomocą danej wartości parametru *batch size*. Analiza otrzymanych wyników podczas szkolenia badanego modelu wskazują że w czasie trenowania otrzymano około pięciu procent różnorodnych cząsteczek.

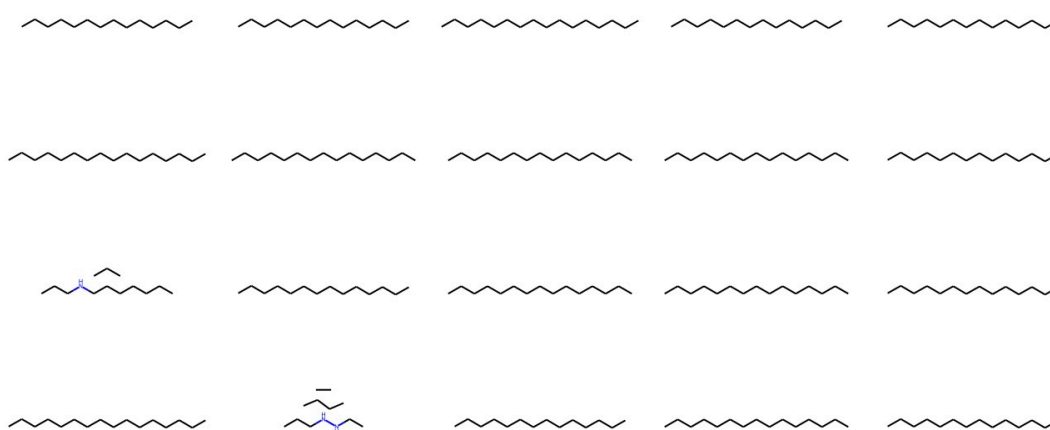




Rysunek 20. Wynik przy wartości zmiennej batch size 128

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [35]

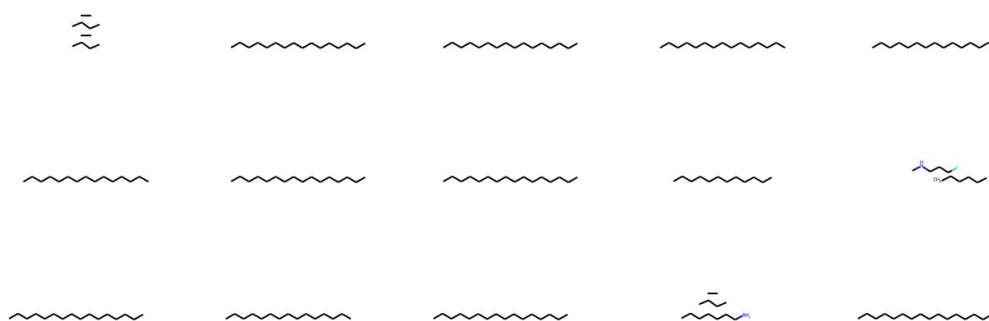
Następna wartość którą przyjęła zmienna *batch size* to sześćdziesiąt cztery. Poniższy rysunek dwadzieścia jeden przedstawia fragment wyniku otrzymanego podczas szkolenia jednej z dziesięciu epok, z użyciem wyżej wspomnianej wartości zmiennej. Analiza otrzymanych wyników podczas szkolenia modelu z wartością zmiennej sześćdziesiąt cztery, wykazuje że w czasie trenowania każdej epoki procentowa ilość uzyskanych unikalnych cząsteczek wynosi około trzech procent. Porównując otrzymany wynik, z poprzednimi, można klarownie stwierdzić że owa wartość zmiennej *batch size*, miała wpływ na pogorszenie wyników szkolenia sieci neuronowej, w związku z powyższym, można wnioskować że dana wartość nie nadaje się do trenowania tej sieci neuronowej.



Rysunek 21. Wynik przy wartości zmiennej batch size 64

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [35]

Ostatnia wartość zmiennej *batch size*, która została objęta treningiem była wartość trzydzieści dwa, a poniższy rysunek dwadzieścia dwa przedstawia fragment wyniku otrzymanego podczas szkolenia jednej z dziesięciu epok. Analiza otrzymanych wyników podczas szkolenia badanego modelu wykazuje że w czasie trenowania każdej epoki procentowa ilość uzyskanych unikalnych molekuł stanowi około dwóch procent.



Rysunek 22. Wynik przy wartości zmiennej *batch size* *batch size* 32

Źródło: Opracowanie własne A.Vezdenetska na podstawie sieci neuronowej[35]

Dana wartość zmiennej *batch size*, miała wpływ na pogorszenie wyników szkolenia sieci neuronowej, w związku z powyższym, można wnioskować że dana wartość nie nadaje się do trenowania owej sieci neuronowej.

#### 4. 6. 1 Wnioski podrozdziału:

Zwiększenie wsadu, miało zapewnić bardziej stabilne i szybkie uczenie się sieci. Wsad o największej badanej wartości w tym eksperymencie równy pięćset dwanaście, jednak dan wielkość wsadu nie pozwoliła uzyskać wyniku. Powodem ku temu było to że po ukończeniu szkolenia trzech epok strata przyjęła nie liczbowa wartość - „nan”. Przez napotkanie owego problemu w trakcie szkolenia, nie udało się wygenerować schematów molekuł.

Dalsze obniżenia wartości zmiennej *batch size* wyglądały obiecująco, zwłaszcza wynik otrzymany przy testowanej wartości sto dwadzieścia osiem, gdyż wtedy w niektórych epokach procentowa ilość uzyskanych unikalnych molekuł sięgała trzydziestu procent. Taki wynik świadczył o tym że dana wartość znajduje się bliżej wartości optymalnej dla tego parametru w porównaniu do poprzedniego wyniku. Lecz następne obniżenie wielkości wsadu przełożyło się nie tylko na spowolnienie szybkości nauki sieci neuronowej, ale także doprowadziły do przeciążenia sieci i napotkanie problemu przeuczenia się sieci neuronowej.

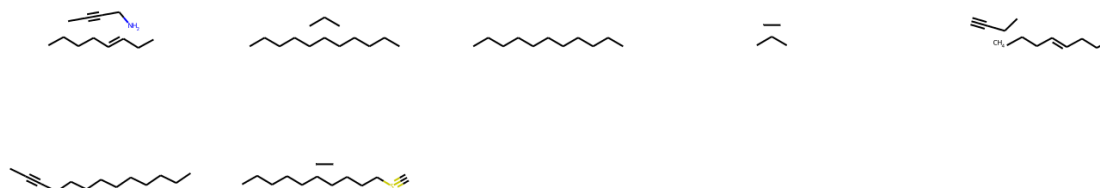
Porównując otrzymane wyniki, przedstawione w poniższej tabeli można klarownie stwierdzić że wartość dwieście pięćdziesiąt sześć, była najbliższej do pożądanego wyniku.

Tabela 7. Wyniki otrzymane w trakcie weryfikacji zmiennej batch size

<i>Batch size</i>	32	64	128	256
Procent unikalnych molekuł	2%	3%	5%	16%

Źródło: Opracowanie własne A. Vezdenetska na podstawie otrzymanych wyników

Ze względu na to że dla testowanych wartości wynik procentowy otrzymywanych unikalnych molekuł był niski, powtórzono trenowanie z wartością zmiennej *batch size* równej stu. Przy tej wartości, udało się otrzymać, jak do tej pory największą ilość różnorodnych cząsteczek.



Rysunek 23. Wynik przy wartości zmiennej batch size 100

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [35]

Procentowa ilość uzyskanych unikalnych molekuł przy *batch size* równe 100 była powyżej trzydziestu procent. Wszystkie wyniki, otrzymane w trakcie trenowania, można będzie odnaleźć w odpowiednim repozytorium na github pod poniższym linkiem:

<https://github.com/anksunamoona/Praca-Dyplomowa/tree/main/PRZEGLĄD%20PRAKTYCZNY%20“DRUG%20MOLECULE%20GENERATION%20WITH%20VAE”/4.5.%20Batch%20size>

## 4. 7 Regulacja zmiennej *batch normalization*

W celu poprawienia wyników do badanej architektury sieci neuronowej dodano parametr odpowiedzialny za normalizację wsadu (z ang. *batch normalization*), stosowany zazwyczaj w warstwach w pełni połączonych<sup>2</sup>. Taka warstwa występuje w enkoderze, w związku z tym technika ta była zastosowana w pierwszej kolejności dla enkodera.

Technika ta pozwala na utrzymanie stałego rozkładu danych na wejściu do każdej warstwy, gdyż bezpośrednio dostosowuje dane wejściowe każdej warstwy w oparciu o mini-partię (wsad) z danych treningowych. Normalizacja pomaga w stabilizacji rozkładu danych, gdyż utrzymuje wartość średniej, tak by ona była bliska zera, natomiast odchylenie standardowe było bliskie jeden. To sprawia że sieć jest bardziej stabilna, oraz mniej podatna na zanikanie gradientu lub eksplozję gradientu podczas uczenia [3]. Normalizacja wsadu nie pozwala gradientowi nawiązywać operacji, które zwiększałyby odchylenie standardowe lub średnią danej jednostki [16].

Regulacja zmiennej odbywa się przez przepisanie wartości *momentum* w zakresie od 0 do 1, która określa, jak duży wpływ ma historyczna średnia i wariancja (uzyskana z poprzednich mini-partii) na obliczanie średniej i wariancji dla aktualnej mini-partii. Wartość gdy *momentum* jest bliskie 1, większy nacisk kładziony jest na historyczne statystyki, a gdy jest bliskie 0, większy nacisk jest skierowany na bieżące statystyki mini-partii [39].

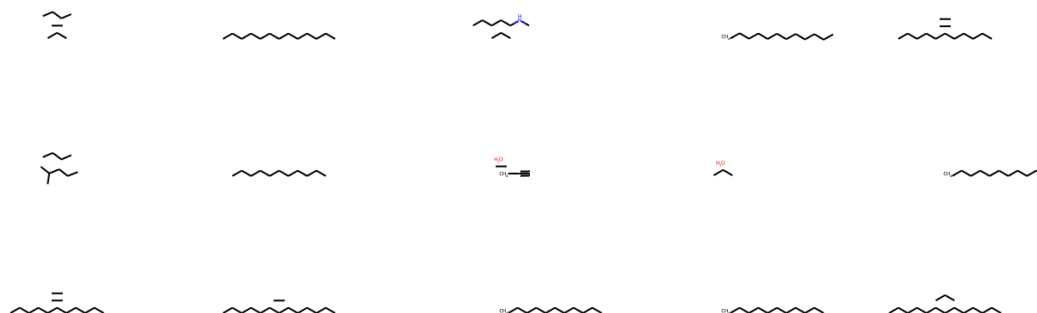
W dalszej części zostaną przedstawione rysunki otrzymanych modeli molekuł w zależności od wykorzystanego *momentum*, zweryfikowano kilka wartości od 0,9 do 0,3, a także wartość 0,1, lecz wynik z trenowania 0,1 nie zostanie przedstawiony na rysunku, gdyż średnio unikalne molekuły pozyskane podczas szkolenia stanowią około dwóch procent od całości. Wykorzystanie takiej wartości oznacza że obliczenia średnich i wariancji wykonane w poprzednich epokach mają wpływ około dziesięciu procent, podczas gdy pozostałe dziewięćdziesiąt procent wpływu pochodzi od bieżącej mini-partii.

Następna analizowana wartość zmiennej wynosi 0,3. Poniższy rysunek przedstawia część wyniku otrzymanego w trakcie nauki sieci neuronowej. W tym przypadku ze

---

<sup>2</sup> Warstwa w pełni połączona (z ang. *dense layer*) to warstwa, w której każdy neuron jest połączony z każdym neuronem w poprzedniej warstwie[4].

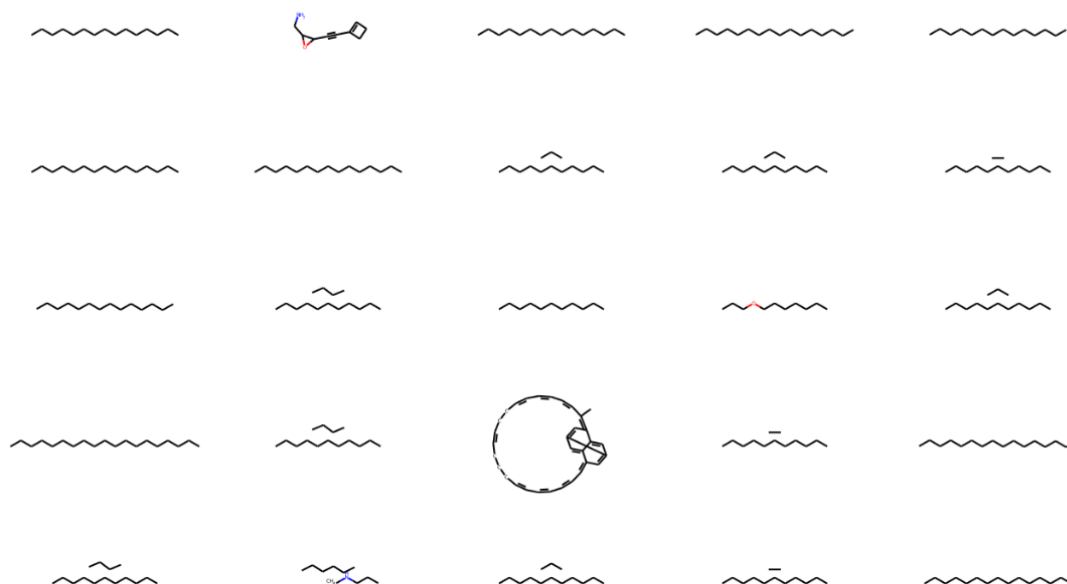
zwiększeniem wartości, poprawił się również wynik. Siedemdziesiąt procent wpływu na naukę w tym przypadku pochodzi od aktualnej mini-partii, a ilość unikalnych molekuł w tym wyniku wynosi około ośmiu procent.



Rysunek 24. Część wyniku, momentum 0,3

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [35]

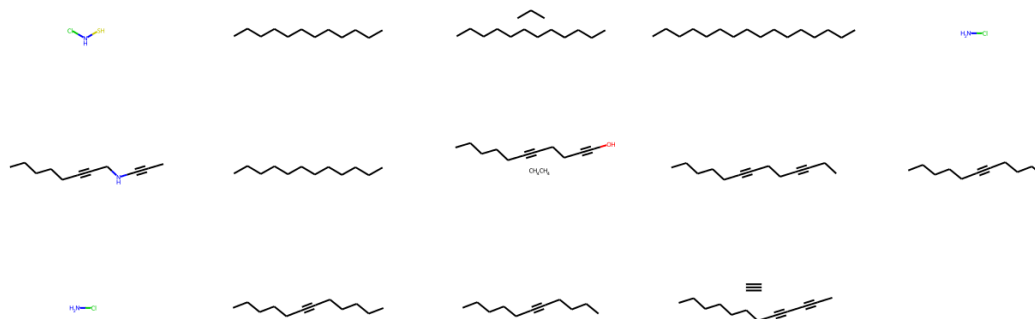
Następna weryfikowana wartość zmiennej jest 0,6, otrzymaną część wyniku przedstawiono poniżej na rysunku dwadzieścia pięć. W tym przypadku wynik pogorszył się, gdyż ilość unikalnych molekuł w tym wyniku wynosi około sześciu procent.



Rysunek 25. Część wyniku, momentum 0,6

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [35]

W dalszej części przeanalizowano wynik otrzymany podczas trenowania sieci o wartości *momentum* równej 0,9. Wpływ wsadu jest w tym wyniku jest najmniejszy, stanowi on około dziesięciu procent. Natomiast obliczenia średnich i wariancji wykonane w poprzednich epokach mają wpływ około dziewięćdziesięciu procent.



Rysunek 26. Część wyniku, *momentum* 0,9

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [35]

Trenowanie sieci o danych parametrach, pozwoliło otrzymać lepszy wynik w porównaniu, do poprzednich analizowanych. Dana sieć wygenerowała około piętnastu procent różnorodnych przedstawień graficznych.

#### 4. 7. 1 Wnioski podrozdziału:

Analiza otrzymanych wyników wskazuje na to że im mniejszy wpływ mają aktualne dane z wsadu, tym lepszy wynik otrzymamy podczas trenowania wybranej sieci neuronowej. Oprócz wyżej wymienionych wartości zmiennej *momentum* sprawdzono także inne wartości, lecz najlepszy uzyskany wynik jest przy wartości *momentum* 0,9. Otrzymane wyniki sprawdzane dla różnych wartości kroków, są udostępnione również w repozytorium na github pod poniższym linkiem:

<https://github.com/anksunamoona/Praca-Dyplomowa/tree/main/PRZEGLĄD%20PRACY%20DRUG%20MOLECULE%20GENERATION%20WITH%20VAE>/4.6.%20Batch%20normalization%20encoder

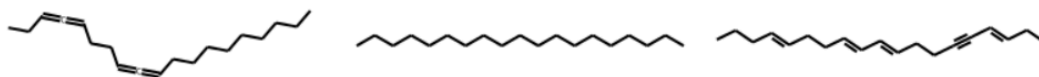
## 4. 8 Rozbudowa warstw

W obecnej sieci neuronowej w podstawowej architekturze sieci uwzględniono dwie warstwy w pełni połączone. W celu weryfikacji jak zmiany w architekturze warstw wpłyną na wynik końcowy został ten eksperyment. Zmienione zostały nie tylko ilość warstw, ale także ilość neuronów.

### 4. 8. 1 Rozbudowa warstwy w pełni połączonej enkodera

Najpierw w warstwie w pełni połączonej zmieniono ilość neuronów, lecz najlepszy wynik był otrzymany w podstawowym kodzie, gdzie w warstwie było pięćset dwanaście neuronów. Sprawdzono ilość dwieście pięćdziesiąt sześć neuronów oraz tysiąc dwadzieścia cztery neurony. W tym przypadku, najlepszy wynik pozostał dla podstawowej sieci o ilości pięciuset dwunastu neuronów

Następnie rozbudowano warstwę w pełni połączoną tak, aby składała się z dwóch warstw. Sieć została wytrenowana dla dziewięciu architektur, w których warstwy składały się z dwustu pięćdziesięciu sześciu neuronów; pięćset dwunastu neuronów; tysiąc dwudziestu czterech neuronów.



Rysunek 27. Wynik, 1024 i 256 neuronów.

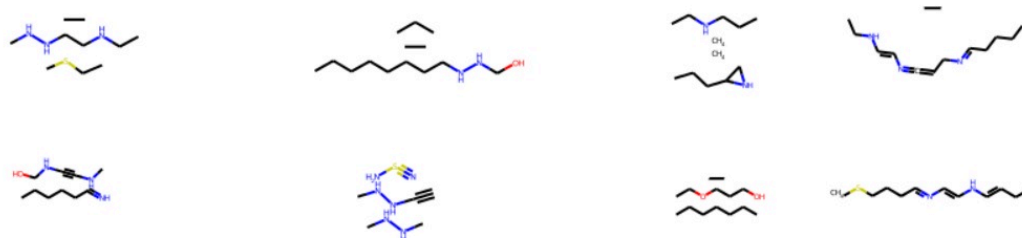
*Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [35]*

Jeśli analizować otrzymane wyniki, to najlepsze były spotykane w architekturze, która składała się z warstw gdzie było tysiąc dwadzieścia cztery i dwieście pięćdziesiąt sześć neuronów, gdyż ilość unikalnych molekuł uzyskanych w trakcie trenowania stanowi około sześćdziesiąt procent. Powyższy rysunek dwadzieścia siedem, prezentuje otrzymany wynik, otrzymany w trakcie trenowania jednej z dziesięciu epok.

Kolejne interesujące wyniki otrzymano przy trenowaniu sieci, gdzie warstwa w pełni połączona składała się z dwóch warstw po dwieście pięćdziesiąt sześć neuronów



w każdej warstwie. Przy trenowaniu tej architektury otrzymano pięćdziesiąt procent unikalnych molekuł.

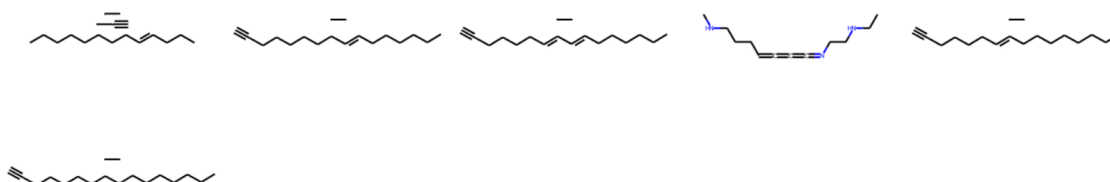


Rysunek 28. Wynik, 1024 i 1024 neuronów.

*Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [35]*

Architektura, gdzie warstwy składały się z tysiąca dwudziestu czterech neuronów, ilość unikalnych molekuł wahała się między dziesięć a pięćdziesiąt procent, powyższy rysunek dwadzieścia sześć, prezentuje wybrane unikalne molekuły otrzymane w trakcie trenowania. Na dane molekuły zwraca się szczególną uwagę, gdyż taki wynik świadczy o tym że jakościowo dany trening dał lepszy wynik, pomimo tego że procentowy wynik jest niższy.

Kolejno rozbudowano warstwę tak, aby składała się z trzech warstw z wykorzystaniem poprzednio wymienionej liczby jednostek. Liczbę jednostek regulowano w celu weryfikacji otrzymywanych wyników, było przetrenowane w sumie dwadzieścia siedem architektur.



Rysunek 29. Wynik, 1024, 512, 256 neuronów.

*Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [35]*

Najlepszy uzyskany wynik był otrzymany przy architekturze, gdzie warstwy kolejno składały się z tysiąc dwudziestu czterech, pięciuset dwunastu oraz dwustu pięćdziesięciu sześciu neuronów. Powyższy rysunek przedstawia wynik trenowania

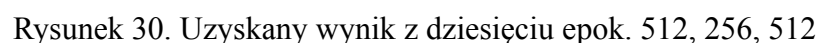
jednej z dziesięciu epok. Otrzymany wynik, po wartości unikalnych molekuł sięgnął siedemdziesięciu procent. Były także dwie inne architektury warte uwagi, lecz w ich przypadku udało się maksymalnie otrzymać połowę unikalnych molekuł z otrzymanej puli w trakcie trenowania. Wszystkie wyniki, otrzymane w trakcie trenowania sieci dotyczącej rozbudowy warstwy w pełni połączonej enkodera, można będzie odnaleźć w odpowiednim repozytorium na github pod poniższym linkiem: <https://github.com/anksunamoona/Praca-Dyplomowa/tree/main/PRZEGLĄD%20PRACY%20%22DRUG%20MOLECULE%20GENERATION%20WITH%20VAE%22/4.7.%20Rozbudowa%20warstw/encoder>

#### **4. 8. 2 Rozbudowa warstw dekodera**

W kodzie podstawowym dekodery składa się z trzech warstw w pełni połączonych, gdzie liczba neuronów w kolejnych warstwach wynosi: sto dwadzieścia osiem, dwieście pięćdziesiąt sześć i pięćset dwanaście. W dalszej części tego rozdziału zmieniono ilość neuronów, które testowano wcześniej dla enkodera. W dalszej części testów, zarówno enkoder oraz dekodery składają się z trzech warstw w pełni połączonych. Wybrana architektura enkodera, składa się z następujących warstw, które kolejno zawierają tysiąc dwadzieścia cztery neuronów, pięćset dwanaście oraz dwieście pięćdziesiąt sześć neuronów. a ilość neuronów dla dekodera zmienia się w tym zakresie.

Tak jak i w przypadku enkodera dla trzech w pełni połączonych warstw zweryfikowano dwadzieścia siedem architektur, gdzie głównie zmieniano ilość neuronów występujących w każdej warstwie. Ilość neuronów w warstwach zmieniała się tak, aby warstwy składały się z dwustu pięćdziesięciu sześciu neuronów; pięćset dwunastu neuronów; tysiąc dwudziestu czterech neuronów.

Niestety większość wyników nie udało się uzyskać gdyż sieć podczas trenowania w funkcji straty pojawiał się nie liczbowy wynik "nan", co oznacza że wartość funkcji wykracza poza granice liczb zmiennoprzecinkowych.



Najlepszy wynik w tym badaniu uzyskano z trenowania sieci o architekturze, w której ilość neuronów w warstwach dekodera wyniosły kolejno pięćset dwanaście, dwieście pięćdziesiąt sześć i pięćset dwanaście. Powyższy rysunek trzydzieści przedstawia otrzymane schematy molekuł otrzymanych z dziesięciu epok. Jak przedstawia rysunek każda molekula jest unikalna, lecz również można zauważyć już opisane związki chemiczne. Wszystkie wyniki, otrzymane w trakcie trenowania sieci, z rozbudowaną warstwą dekodera można zostały przedstawione w odpowiednim repozytorium na github pod poniższym linkiem:

<https://github.com/anksunamoona/Praca-Dyplomowa/tree/main/PRZEGLĄD%20PRA CY%20“DRUG%20MOLECULE%20GENERATION%20WITH%20VAE”/4.7.%20R ozbudowa%20warstw/decoder>

### 4. 8. 3 Wnioski podrozdziału:

Rozbudowa sieci przyczyniła się do uzyskania wyniku, gdzie wszystkie cząsteczki powstałe w trakcie trenowania sieci były unikalne. Jednak otrzymany rezultat może okazać się mylący, gdyż sieć została przeciążona ilością warstw, a także ilością neuronów w niej zawartych. Z kolei niepowtarzalność otrzymanych molekuł świadczy o tym że sieć zaczęła dopasowywać generowane dane do szumu, a nie do danych z zestawu uczącego [25]. Pomimo to, wszystkie otrzymywane molekuły w dalszym ciągu są prawdopodobne i mogą istnieć, gdyż zgodne z regułą oktetu, węgiel w

cząsteczkach organicznych tworzy zazwyczaj cztery wiązania, a dana reguła nie została naruszona w wygenerowanych cząsteczkach [18]. W związku z powyższym zachowaniem reguł, można uważać że otrzymane wyniki, nawet od przeciążonej sieci można uważać za pozytywne.

Jednakże warto zauważyć, że otrzymywane cząsteczki w trakcie szkolenia sieci nie zawsze będą w stu procentach unikatowe, gdyż mogą pojawić się powtarzające się związki chemiczne. Także nie ma gwarancji tego że w powstałych cząsteczkach reguła okrętu nie zostanie złamana, więc warto przyglądać się do otrzymanego wyniku. W celu sprawdzenia tego twierdzenia, trenowanie tej sieci powtórzono kilka razy. Czasem w trakcie trenowania sieci wśród dziesięciu powstałych cząsteczek, dziewięć było unikalnych, gdy w trakcie innych trenowań można było pozyskać dziesięć unikalnych molekuł, z dziesięciu wygenerowanych. W związku z otrzymanymi wynikami, można zrobić wniosek, o tym że w wyniku trenowania sieci o takiej architekturze, można otrzymać ponad dziewięćdziesiąt procent unikalnych molekuł wartych uwagi.

## 4. 9 Podsumowanie przeglądanej pracy

W niniejszym rozdziale zostały omówione badania przeprowadzone nad siecią neuronową przedstawioną w artykule zatytułowanym "*Drug Molecule Generation with VAE*". Przedstawione eksperymenty miały na celu uzyskanie architektury sieci, której zadaniem było generowanie maksymalnej ilości unikalnych grafów cząsteczek. poprawienie uzyskanych wyników przeprowadzono kilka eksperymentów z różnymi architekturami badanej sieci. Niektóre z eksperymentów nie przyniosły początkowo zadowalających rezultatów, ale kontynuowanie ich doprowadziło do uzyskania satysfakcjonujących wyników.

Podczas tych eksperymentów wprowadzone zostały różne modyfikacje, które miały na celu poprawę początkowych wyników. Niektóre z tych modyfikacji, które miały pozytywny wpływ na rezultaty, zostały zachowane w kodzie. Jednak nie wszystkie modyfikacje, które początkowo wydawały się prowadzić do optymalnych wyników, zostały uwzględnione w dalszych eksperymentach.

Kluczowym aspektem tych badań było podejście eksperymentalne i cykliczne, które pozwoliło na weryfikację różnych pomysłów i strategii. To podejście pozwoliło

również na uniknięcie przetrenowania modelu poprzez ciągłe optymalizowanie i dostosowywanie architektury oraz hiperparametrów modelu.

Otrzymanie satysfakcjonujących jakościowo wyników w generowaniu związków chemicznych było głównym celem tego badania. Podejście eksperymentalne pozwoliło na osiągnięcie tego celu, a testowanie i modyfikacje pozwoliły na doskonalenie modelu, optymalizację wyników aby uzyskać pożądane struktury chemiczne.

## 5 PRZEGLĄD PRACY “*WGAN-GP WITH R-GCN FOR THE GENERATION OF SMALL MOLECULAR GRAPHS*”

W tym rozdziale omówiona zostanie praca badawcza o nazwie "*WGAN-GP with R-GCN for the generation of small molecular graphs*". Jak nazwa pracy wskazuje, głównym celem omawianej pracy było stworzenie modelu, który mógłby generować małe cząsteczki molekuł [36].

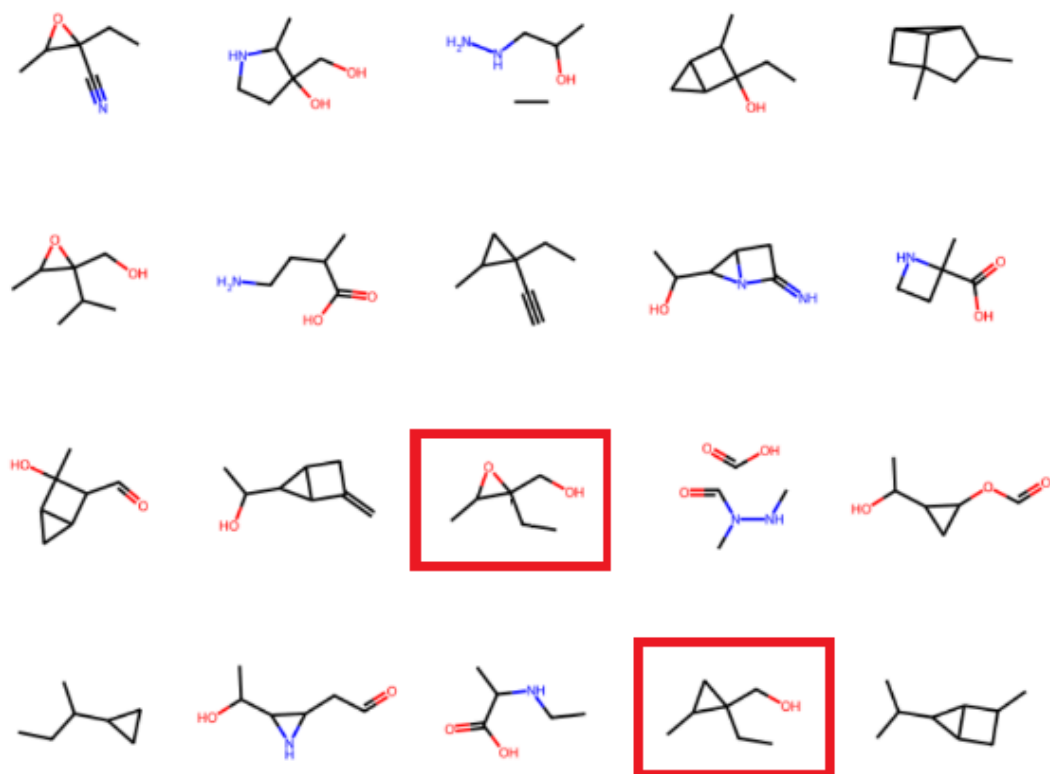
Model został wytrenowany na zbiorze danych mechaniki kwantowej (*QM9*) uzyskany z *MoleculeNet*. Pomimo tego że zestaw danych zawiera wiele kolumn funkcji i etykiet, w danej pracy skupiono się tylko na kolumnie SMILES [36]. SMILES (z ang. *Simplified Molecular Input Line Entry System*) wyraża strukturę danej cząsteczki w postaci ciągu znaków ASCII. Taka forma zapisu jest czytelna dla ludzi, a także może być przetwarzana przez komputery [33]. Zbiór danych *QM9*, gdzie maksymalna liczba atomów ciężkich (innych niż wodór) występujących w cząsteczce wynosi tylko dziewięć [27].

Podejście zastosowane w przeglądanej pracy, jest podejściem hybrydowym, gdyż łączy ze sobą dwie różne techniki. WGAN-GP, dąży do stworzenia jak największej ilości rozmaitych próbek [13]. Natomiast R-GCN to model sieci neuronowej, która została opracowana aby umożliwić pracę z danymi w postaci grafów, podejście relacyjne, pozwala lepiej uwzględniać powiązania między atomami [31].

Podejście zastosowane w tej przeglądanej pracy sprawia to, że model ma zdolność do generowania nowych małych grafów molekularnych, które są zarówno zgodne z danymi treningowymi, jak i nowatorskie. R-GCN dostarcza głębokie zrozumienie struktury molekularnej, a WGAN-GP poprawia jakość i różnorodność wygenerowanych danych.

## 5. 1 Powtórzenie eksperymentu.

W pierwszym podejściu uruchomiono sieć neuronową opisaną w pracy "*WGAN-GP with R-GCN for the generation of small molecular graphs*" bez wprowadzania do niej jakichkolwiek zmian. Dane otrzymane po uruchomieniu sieci z pierwotnym kodem, pozwalają ocenić skuteczność modelu, czas trenowania, a także jakość generowanych cząsteczek molekuł. Jest to kluczowe w celu zrozumienia wyjściowej wydajności modelu i stworzenia punktu odniesienia do ewentualnych późniejszych modyfikacji i doskonalenia.



Rysunek 31. Powtórzenie eksperymentu.

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [36]

Analizując powyższy rysunek trzydziesty pierwszy otrzymany w wyniku uruchomienia sieci neuronowej, widzimy że prawie wszystkie otrzymane związki chemiczne są unikalne. Co świadczy o tym że dany model dobrze radzi sobie z postawionym problemem. W trakcie szkolenia otrzymano tylko dwie takie same molekuly, co także wskazuje na to że daną sieć można jeszcze udoskonalić. W dalszej

części przeglądania tej pracy zostaną zmienione hiper parametry, w celu weryfikacji, jaki wpływ na końcowy wynik będą miały te lub inne modyfikacje. Także warto zaznaczyć że podstawowy kod również będzie powtarzany, w celu porównania otrzymywanych wyników. Dane powtarzania jednak, prawie zawsze pozwalają otrzymać w pełni satysfakcjonujący wynik, gdzie wszystkie otrzymane molekuły są w pełni unikalne.

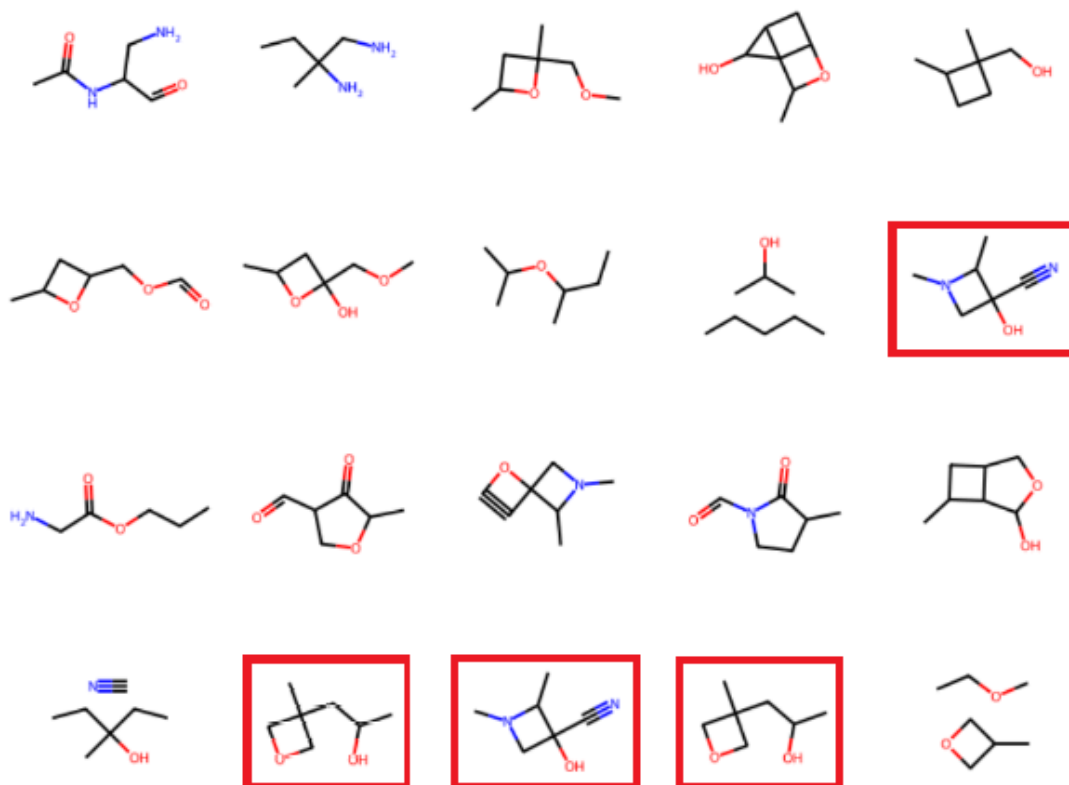
## **5. 2 Wpływ zmiennej *dropout\_rate***

Tak samo jak i dla poprzedniej sieci, dla danej sieci sprawdzony zostanie wpływ na końcowy wynik zmiennej *dropout\_rate*. Dany Parametr występuje zarówno w generatorze, jak i w dyskryminatorze badanej sieci neuronowej. W dalszej części przedstawiono i omówiono otrzymane wyniki.

### **5. 2. 1 Zmienna *dropout\_rate* dla generatora**

Jak już omówiono wcześniej dany hiperparametr w podrozdziale drugim rozdziału czwartego, zmienna *dropout\_rate* kontroluje stopień "wyłączania" losowo wybranych neuronów w trakcie trenowania sieci. W dalszej części tego podrozdziału przedstawiono wyniki otrzymane w trakcie trenowania sieci ze zmianą tego parametru dla generatora, natomiast dla dyskryminatora, wartość zmiennej *dropout\_rate* nie ulega edycji i pozostaje równa początkowej wartości zero przecinek dwa.

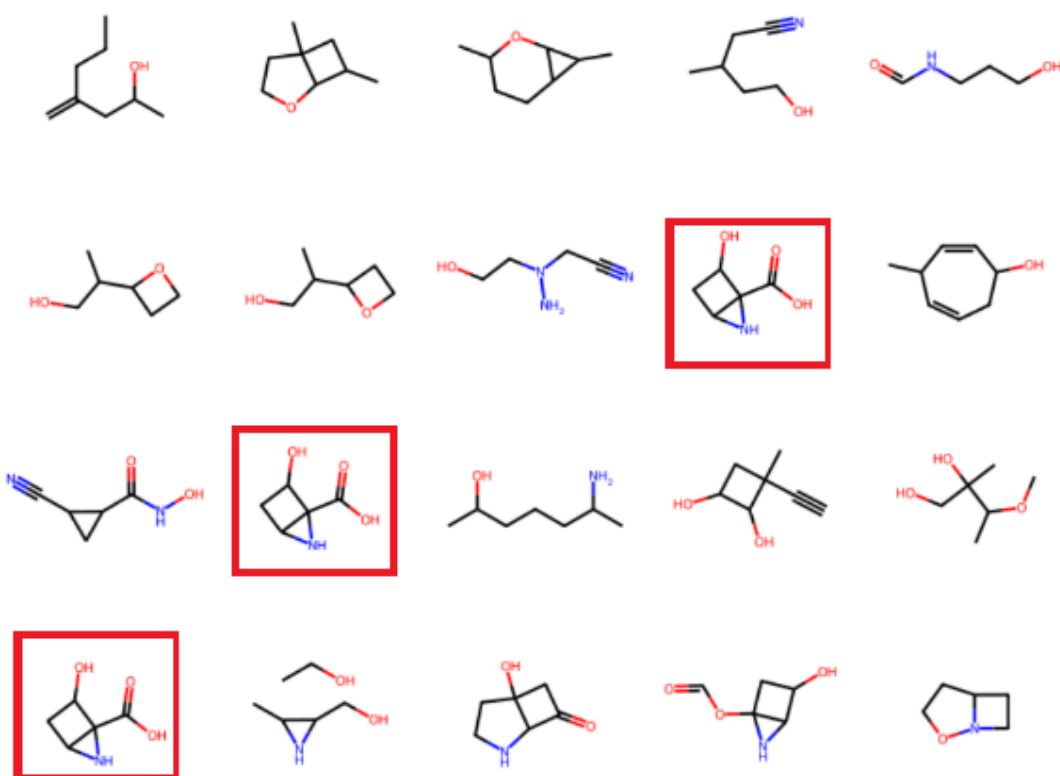




Rysunek 32. Wynik *dropout\_rate* generatora równy 0.0.

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [36]

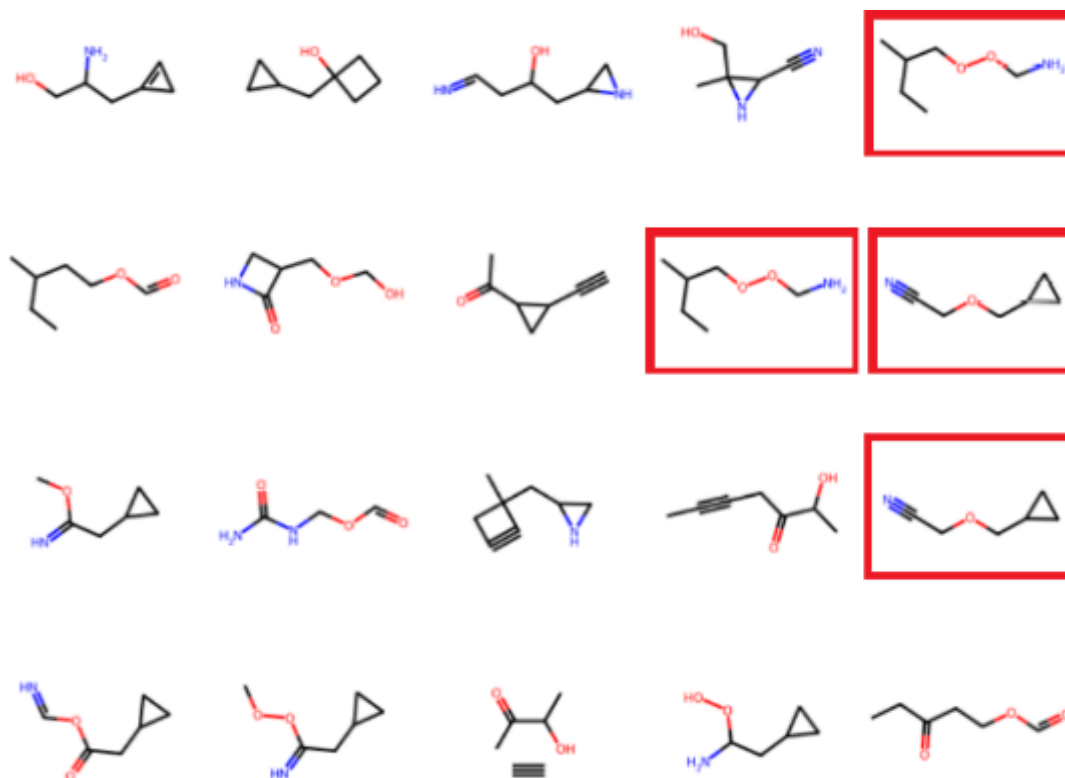
Powyższy rysunek, przedstawia zbiór wytworzonych związków chemicznych otrzymany w trakcie trenowania sieci, gdzie zmienna *dropout\_rate* miała wartość 0.0 dla generatora. Analizując otrzymany rysunek widzimy cztery powtarzające się molekuly, co oznacza że brak wyłączenia neuronów dla generatora sprzyjało *overfittingowi* sieci.



Rysunek 33. Wynik *dropout\_rate* generatora równy 0.2.

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [36]

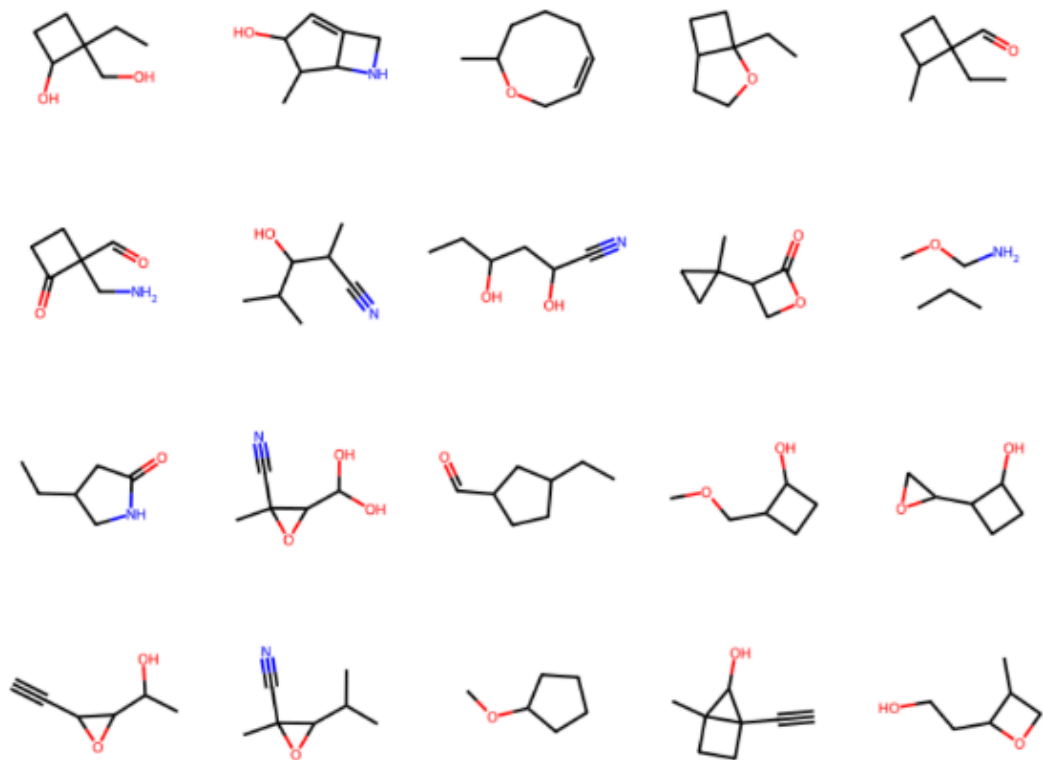
Rysunek trzydzieści trzy, przedstawia wynik trenowania sieci, w której wartość zmiennej *dropout\_rate* jest równa 0,2 zarówno dla generatora jak i dyskryminatora. Dany model powtarza model podstawowy, początkowego kodu, który był wzięty dla przeprowadzenia eksperymentów. Analiza danego wyniku, jednak wskazują na to że w danym trenowaniu pojawiły się trzy identyczne molekuly, co oznacza że dana wartość zmiennej *dropout\_rate* nie jest jednak wartością optymalną dla dalszych badań.



Rysunek 34. Wynik *dropout\_rate* generatora równy 0.4.

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [36]

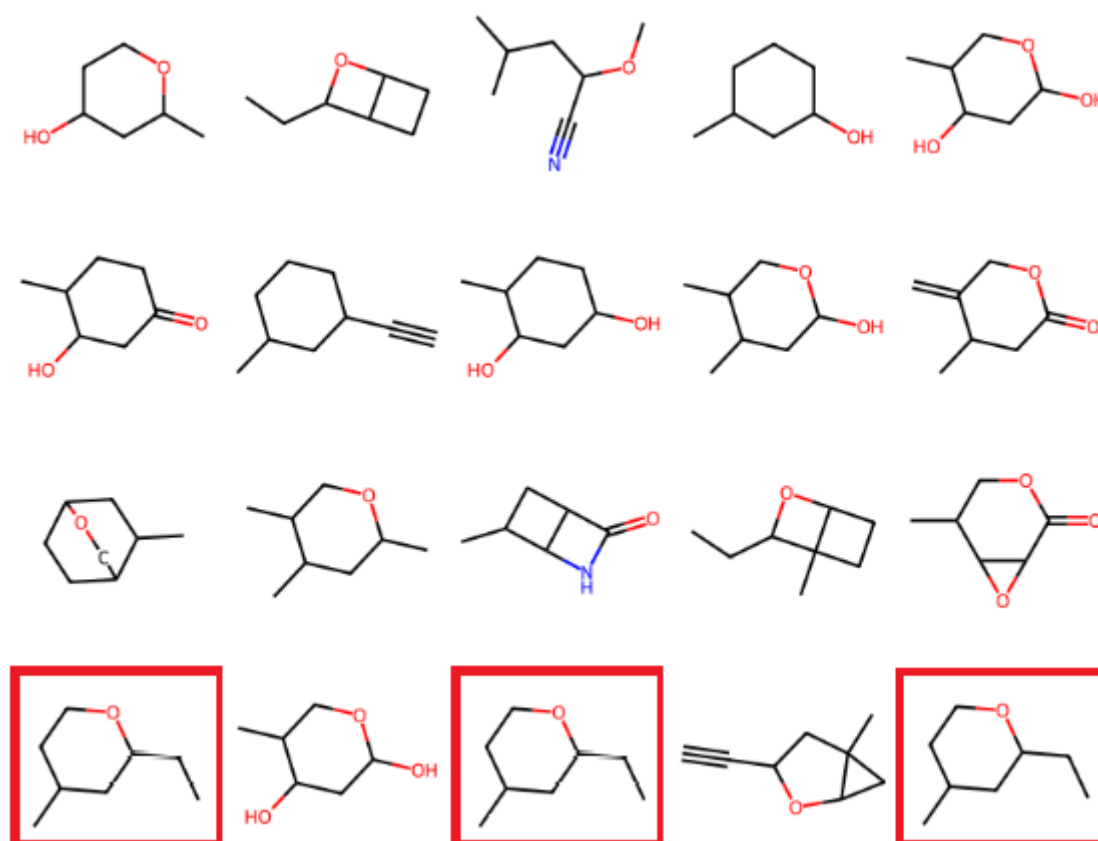
Powyższy rysunek trzydzieści cztery, przedstawia wynik trenowania sieci, w której wartość zmiennej *dropout rate* stanowi zero przecinek cztery dla generatora. Analiza powyższego rysunku, wskazuje że zostały wygenerowane cztery takie same molekuly, co oznacza że dana wartość zmiennej *dropout\_rate* nie jest optymalna.



Rysunek 35. Wynik *dropout\_rate* generatora równy 0.6.

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [36]

Kolejny analizowany rysunek trzydziesty piąty, gdzie zmienna *dropout rate* dla generatora przyjmuje wartość zero przecinek sześć. Na powyższym rysunku nie udało się odnaleźć identycznych molekuł, co wskazuje na to że dana wartość może być optymalną wartością zmiennej dla generatora. W dalszej części badań w niektórych eksperymentach będzie użyta dana wartość zmiennej w celu uzyskania najbardziej urozmaiconych wyników.



Rysunek 36. Wynik *dropout\_rate* generatora równy 0.8.

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [36]

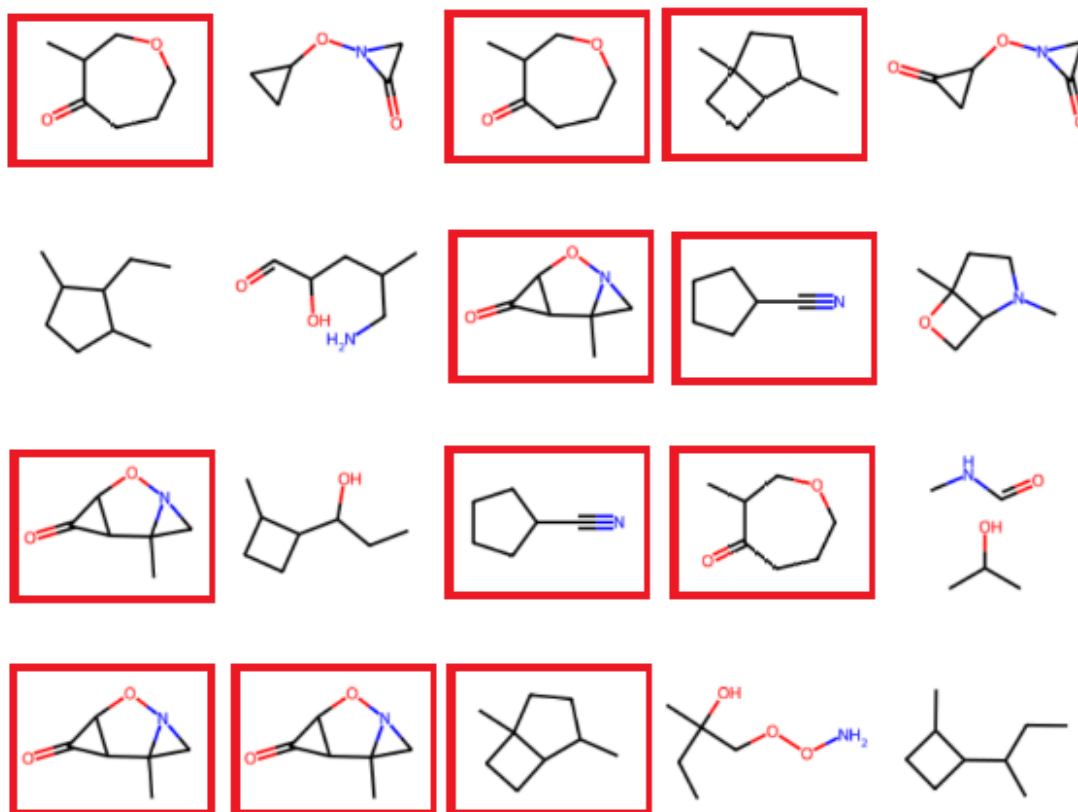
Następny rysunek, który został otrzymany w wyniku trenowania sieci, gdzie zmienna generatora *dropout\_rate* przyjmuje wartość zero przecinek osiem. Jak przedstawia powyższy rysunek trzydzieści sześć, pojawiają się trzy identyczne molekuly, a także wszystkie pozostałe molekuly tworzą się wokół jednego wzorca. Taki skutek wskazują na to że dana sieć neuronowa została przetrenowana. Natomiast wartość zmiennej *dropout\_rate*, gdzie wyłączono zostało osiemdziesiąt procent neuronów generatora w trakcie trenowania sieci, nie pozwala otrzymać pożądanego wyniku.

Warto także zaznaczyć, to że sieć neuronowa tworzy pewne wzorce, do których dąży oraz rozbudowuje. Takie wzorce, stają się widoczne gdy poddajemy analizie wszystkie otrzymane rysunki w trakcie szkolenia sieci neuronowej.

### 5. 2. 2 Zmienna *dropout\_rate* dla dyskryminatora

W tym podrozdziale kontynuowano badanie wpływu zmiennej *dropout\_rate* dla dyskryminatora na otrzymywane wyniki. Dla tej zmiennej wybrano te same wartości, co były użyte dla generatora.

Poniższy rysunek trzydziesty siódmy przedstawia rezultat otrzymany w trakcie trenowania sieci, gdzie wszystkie neurony dyskryminatora brały udział w trenowaniu sieci. Otrzymany wynik, wskazują na przetrenowanie sieci, gdyż analiza rysunku wykazuje cztery wzory identycznych molekuł.



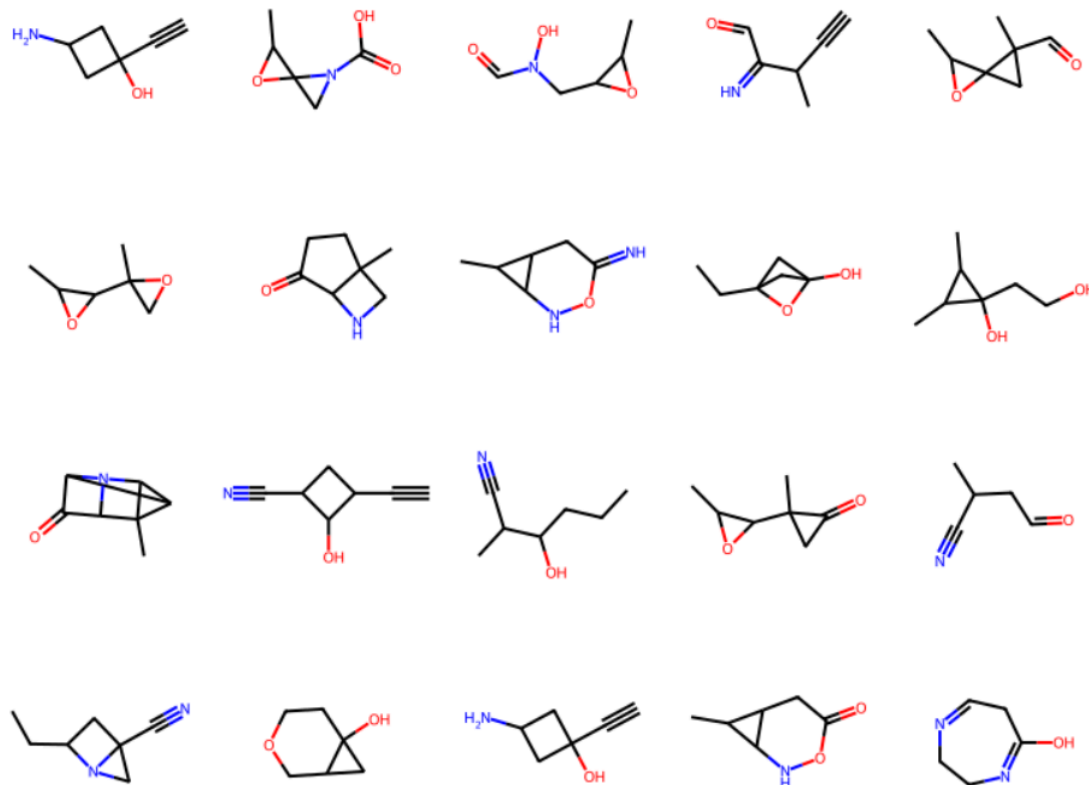
Rysunek 37. Wynik *dropout\_rate* dyskryminatora równy 0.0.

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [36]

Następny rysunek, który poddano analizie, to rysunek trzydzieści osiem, który reprezentuje wynik trenowania sieci gdzie w trakcie trenowania, zostało wyłączono dwadzieścia procent neuronów w każdej iteracji treningowej.

Poniższy rysunek nie posiada wzorów identycznych molekuł, lecz warto również podkreślić, że dany wynik, jest również reprezentacją wyniku pierwotnego kodu.

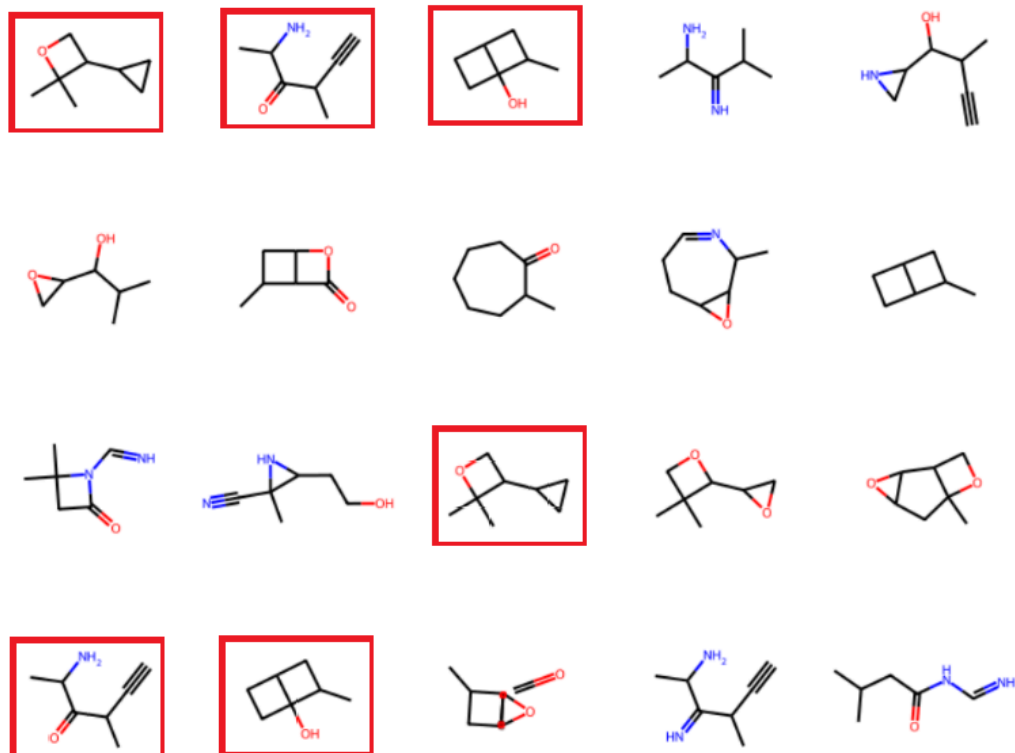
Oznacza to że wartość zmiennej *dropout\_rate* jest identyczna zarówno dla dyskryminatora, jak i generatora.



Rysunek 38. Wynik *dropout\_rate* dyskryminatora równy 0.2.

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [36]

Pomimo tego że w przedstawionym wyniku nie zostały ujawnione identyczne molekuły, można zauważyć pewne wzorce wokół których tworzy się model. Także warto wspomnieć o tym że poprzednie trenowania identycznej architektury sieci, prowadziły do pojawiania się wzorców, przynajmniej dwóch identycznych molekuł. Taki wynik wskazują na to że dana wartość zmiennej *dropout\_rate* nie jest wartością optymalną, ale czasem daje możliwość pozyskać zestaw unikalnych molekuł.



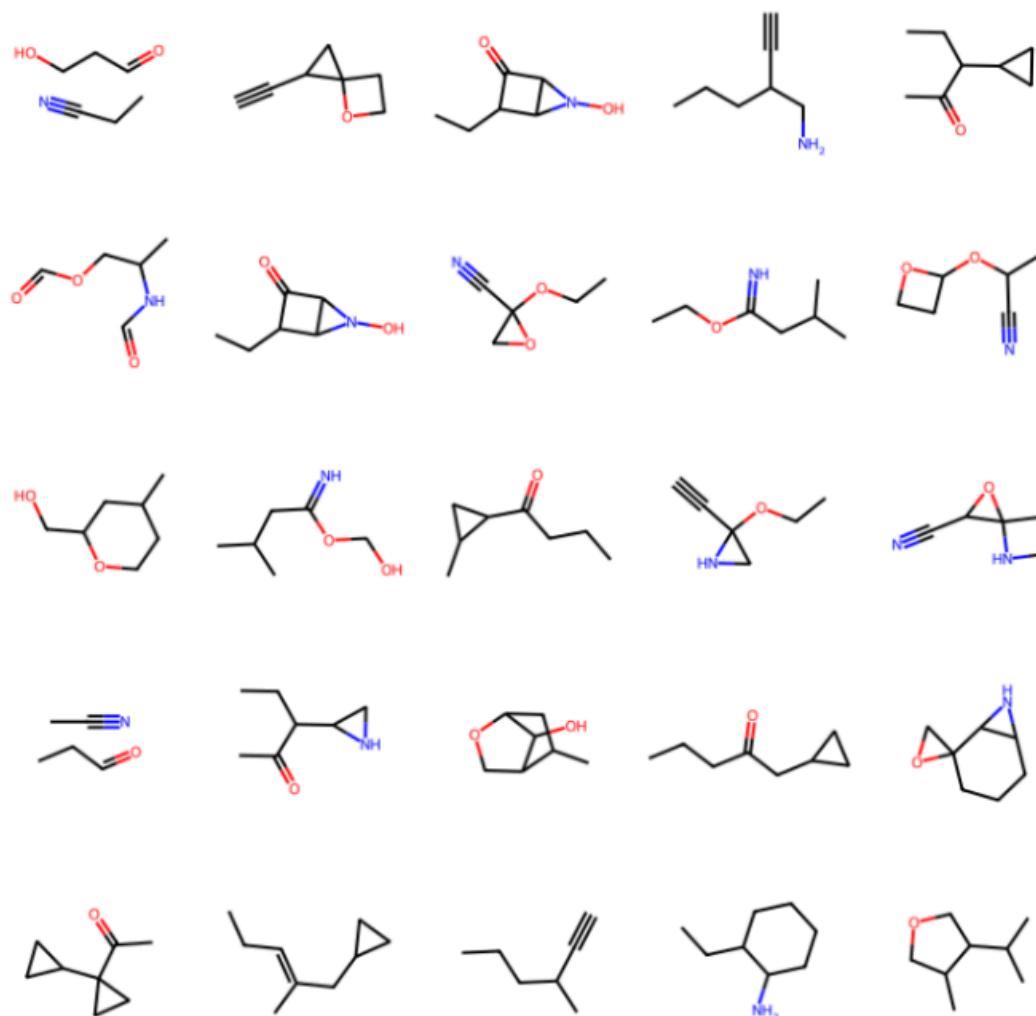
Rysunek 39. Wynik *dropout\_rate* dyskryminatora równy 0.4.

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [36]

Następny wynik, który zostanie omówiony, został otrzymany, kiedy w trakcie trenowania, zostało wyłączono czterdzieści procent neuronów w każdej iteracji treningowej. Analiza rysunku trzydzieści dziewięć wykazała że pojawiły się trzy wzory identycznych molekuł. Występowanie takich wzorów, klarownie wskazują na to że badana sieć neuronowa ulega zjawisku *overfittingu*, przez wyłączenie czterdziestu procent neuronów, w trakcie trenowania sieci.

Kolejny wynik, przedstawia rysunek czterdziesty, otrzymany w wyniku trenowania sieci, gdzie wyłączone zostały sześćdziesiąt procent neuronów w każdej iteracji treningowej. Analiza poniższego rysunku, nie wykazała występowania wzorów identycznych molekuł.

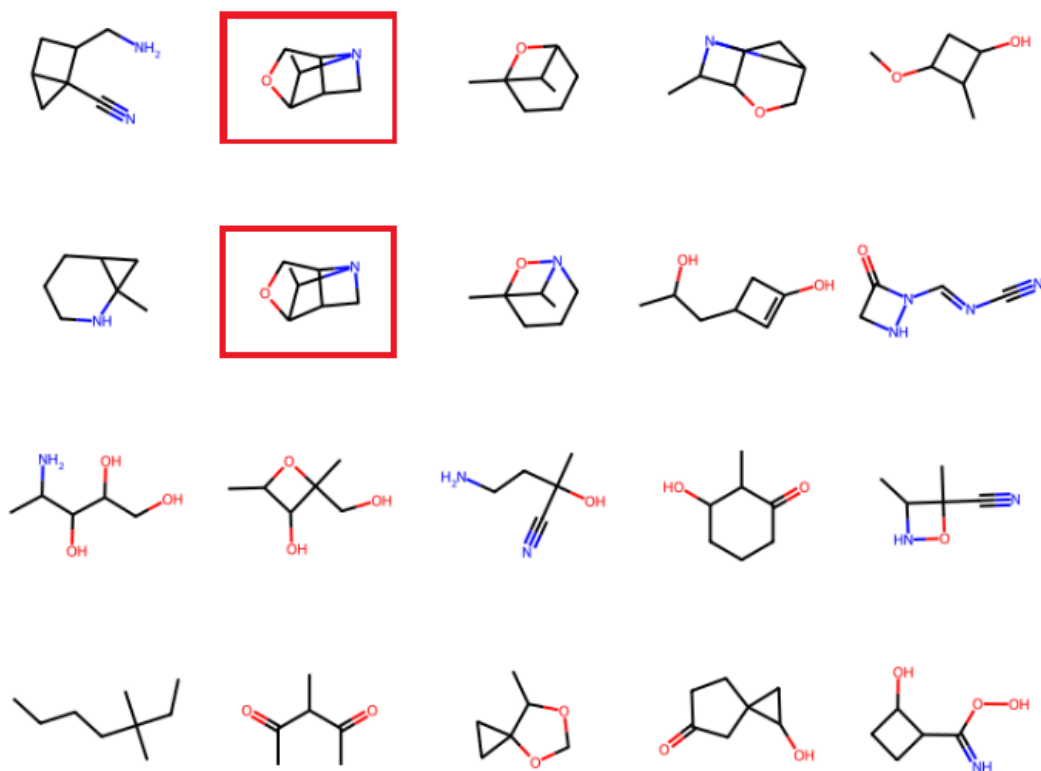




Rysunek 40. Wynik *dropout\_rate* dyskryminatora równy 0.6.

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [36]

Następny wynik, który zostanie poddany analizie, znajdujący się poniżej, rysunek czterdziesty pierwszy. W trakcie trenowania tej sieci, zostało usunięto osiemdziesiąt procent neuronów dyskryminatora w każdej iteracji treningowej. Wyłączenie tak znacznej ilości neuronów, również wywołało zjawisko przetrenowania sieci. Analiza poniższego rysunku wykazała występowanie dwóch identycznych molekuł.



Rysunek 41. Wynik *dropout\_rate* dyskryminatora równy 0.8.

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [36]

Analiza wszystkich powyżej wymienionych cząsteczek, pomaga wyjaśnić występowanie zjawiska przetrenowania sieci, a także pomaga w odnalezieniu możliwości uniknięcia tego problemu.

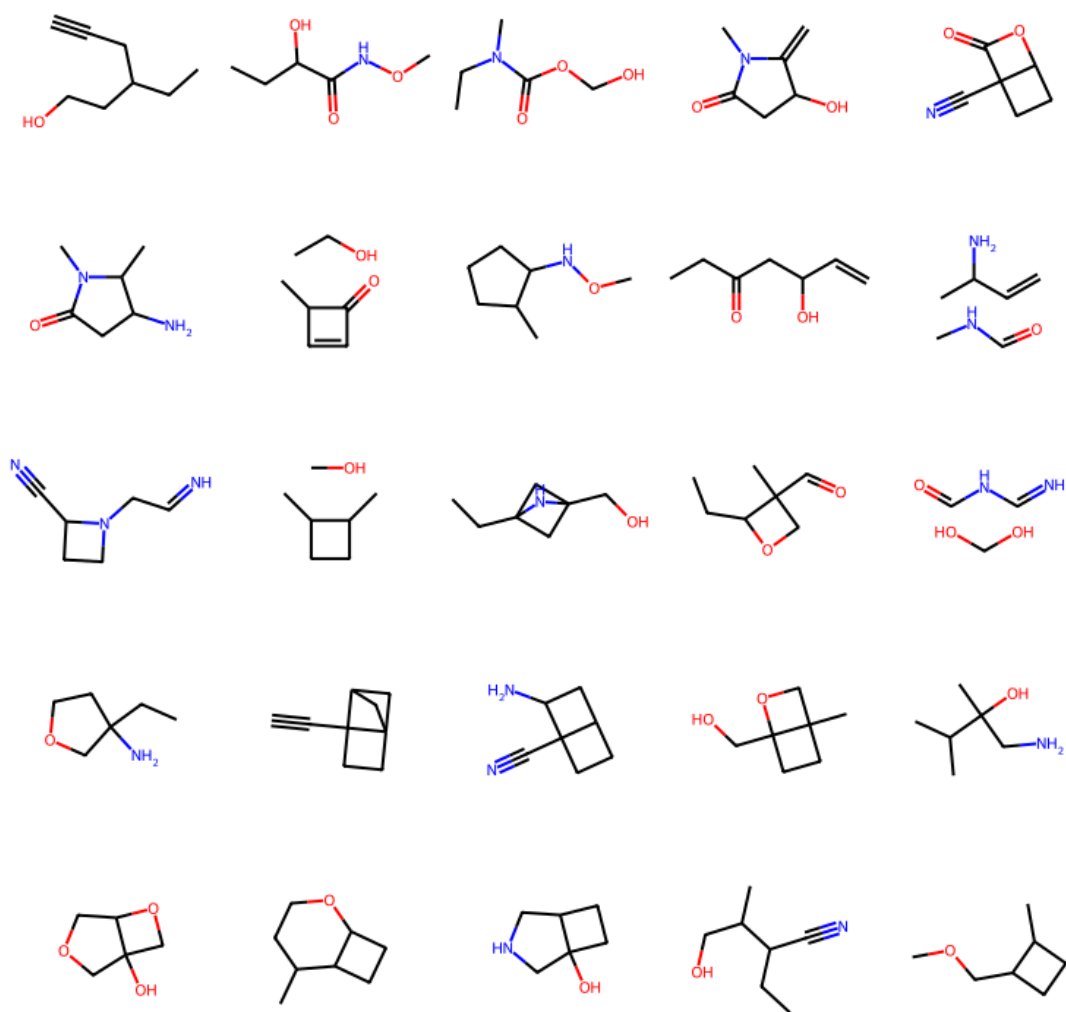
### 5. 2. 3 Wnioski podrozdziału 5. 2:

Podczas weryfikacji optymalnych wartości dla badanego hiperparametru, jedna wartość, zdaje się być wartością optymalną, zarówno dla generatora, jak i dyskryminatora. Wartość ta wiąże się z wyłączeniem sześćdziesięciu procent neuronów, nie prowadząc przy tym do przetrenowania sieci. Warto także wziąć pod uwagę to że dana wartość prowadzi do przeciążenia sieci, ale wykorzystanie technologii WGAN -GP razem z technologią R-GCN, pozwoliły otrzymać wiarygodne cząsteczki.

W celu potwierdzenia danej hipotezy wykonano pięciokrotne trenowanie sieci o zadanej wartości dla zmiennej *dropout\_rate* zarówno dla generatora i dyskryminatora.

Poniższy rysunek, przedstawia jeden z pięciu otrzymanych wyników trenowania sieci, gdzie zmienna *dropout\_rate* miała wartość 0.6 zarówno dla dyskryminatora, jak i generatora. Dany, a także inne wyniki zostały umieszczone w repozytorium na github pod poniższym linkiem

<https://github.com/anksunamoona/Praca-Dyplomowa/tree/main/PRZEGLĄD%20PRACY%20WGAN-GP%20with%20R-GCN%20for%20the%20generation%20of%20small%20molecular%20graphs>



Rysunek 42. *Dropout* 0,6 dla dekodera i enkodera.

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [36]

Analiza powyższego rysunku nie wykazała obecności identycznych molekuł, co oznacza że wykorzystane wartości danej zmiennej mogą rzeczywiście stać się wartością optymalną. Oprócz danego wyniku, także analizowano inne otrzymane wyniki, w jednym z pięciu wyników, odnaleziono dwie identyczne molekuły, co

oznacza że otrzymywany rezultat można w dalszym ciągu doskonalić. W celu poszukiwania dalszych udoskonaleń wyników, wykonano eksperymenty opisane w dalszej części tego rozdziału.

## 5.3 Wpływ optymalizatora

Dany rozdział zostanie poświęcony opisaniu eksperymentów gdzie sprawdzono wpływ wybranych optymalizatorów na badaną sieć neuronową. Sprawdzono dwadzieścia pięć architektur sieci, wykorzystując różne kombinacje optymalizatorów: Adam, Nadam, Adagrad, Adadelta i SGD. Wszystkie notatniki, w których zawarte wyniki tych badań są dostępne w repozytorium github pod dalszym linkiem: <https://github.com/anksunamoona/Praca-Dyplomowa/tree/main/PRZEGLĄD%20PRACY%20“WGAN-GP%20with%20R-GCN%20for%20the%20generation%20of%20small%20molecular%20graphs”/5.%203.%20Optimazer>

Dla generatora oraz dyskryminatora, były wybierane różne optymalizatory. Podstawowy eksperyment, przedstawiony w pracy źródłowej, wykorzystywał optymalizator Adam dla obu sieci. W tym eksperymencie były wybrane pięć optymalizatorów, w celu weryfikacji zachowania sieci, przy różnych optymalizatorach.

### 5.3.1 Analiza wyników

Przykładowo algorytmy optymalizatorów, takich jak Adam i Nadam, są najbardziej wydajne, w przypadku badanej sieci neuronowej, w ich przypadku w trakcie trenowania, powstaje najmniej identycznych molekuł. Dane dwa optymalizatory, najlepiej funkcjonują kiedy Nadam, jest optymalizatorem generatora, a Adam z kolei, jest optymalizatorem dyskryminatora. Analiza takiej architektury nie wykazuje występowania identycznych molekuł.

Kolejnym przykładem, kiedy wzorce są dobrze widoczne, może być wiersz reprezentujący wyniki, otrzymane gdy optymalizatorem generatora był Adadelta, w tabeli widoczne jest to że, nie powstają żadne bardziej złożone figury, w tym wierszu przedstawione tylko proste związki chemiczne, które nie posiadają pewnej struktury przestrzennej. Warto także zaznaczyć że gdy Adadelta występuje jako optymalizator, dla dyskryminatora, mogą pojawiać się bardziej złożone struktury przestrzenne, ale

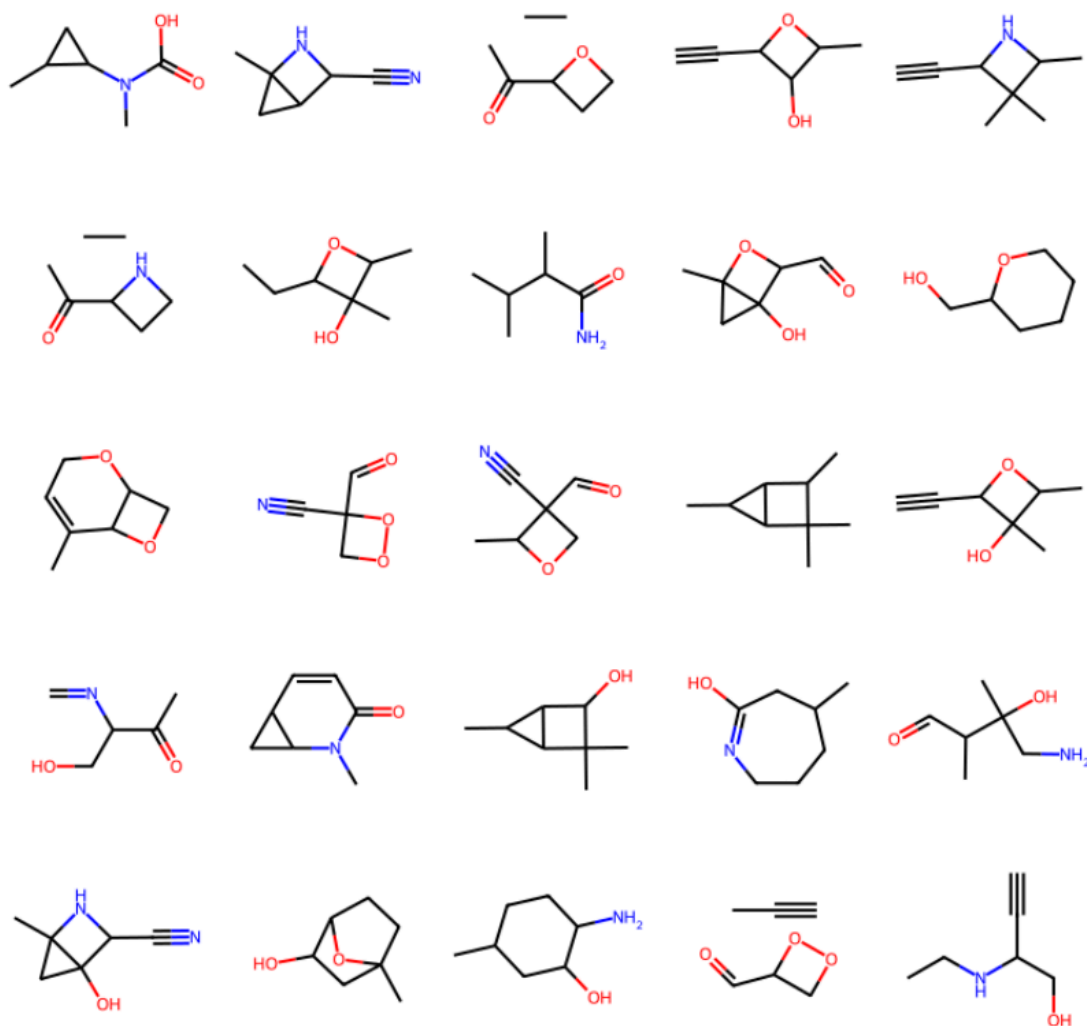
dane struktury, najczęściej są powtarzaniem pewnego wzorca otrzymanego w trakcie trenowania sieci neuronowej. Taka analiza pozwala uznać że optymalizator Adadelta jest najmniej odpowiedni dla danej sieci neuronowej.

Optymalizator SGD, można także nazwać najmniej odpowiednim dla danej sieci neuronowej, gdyż niezależnie, czy on pojawia się jak optymalizator generatora czy dyskryminatora, to pojawiają się wzorce identycznych molekuł. Takie wyniki trenowania świadczą o tym że sieć neuronowa, przy takich ustawieniach nie może dobrze radzić sobie z postawionym zadaniem.

Kolejny optymalizator który zostanie omówiony, będzie Adagrad, w przypadkach, kiedy Adagrad jest optymalizatorem dla generatora, prawie wszędzie pojawiają się pewne wzorce identycznych molekuł. Wyjątek w tym przypadku stanowi tylko przypadek, kiedy optymalizatorem dyskryminatora jest Nadam, wtedy nie pojawiają się wzorce identycznych molekuł. Z kolei w przypadku, kiedy Adagrad jest optymalizatorem dyskryminatora, wszystkie otrzymywane wyniki posiadają wzory identycznych molekuł.

### **5. 3. 2 Wnioski podrozdziału:**

Podsumowując, optymalizatory mają znaczący wpływ na wynik, który otrzymamy po wytrenowaniu sieci. Analiza wszystkich otrzymanych wyników, pozwoliła wydzielić dwie architektury sieci, które najlepiej radzą z problemem pozyskania różnorodnych, unikalnych molekuł. Jedną z nich to kiedy, Nadam, jest optymalizatorem generatora, a Adam jest optymalizatorem dyskryminatora.



Rysunek 43. Wynik: Nadam dla generatora, a Adam dla dyskryminatora.

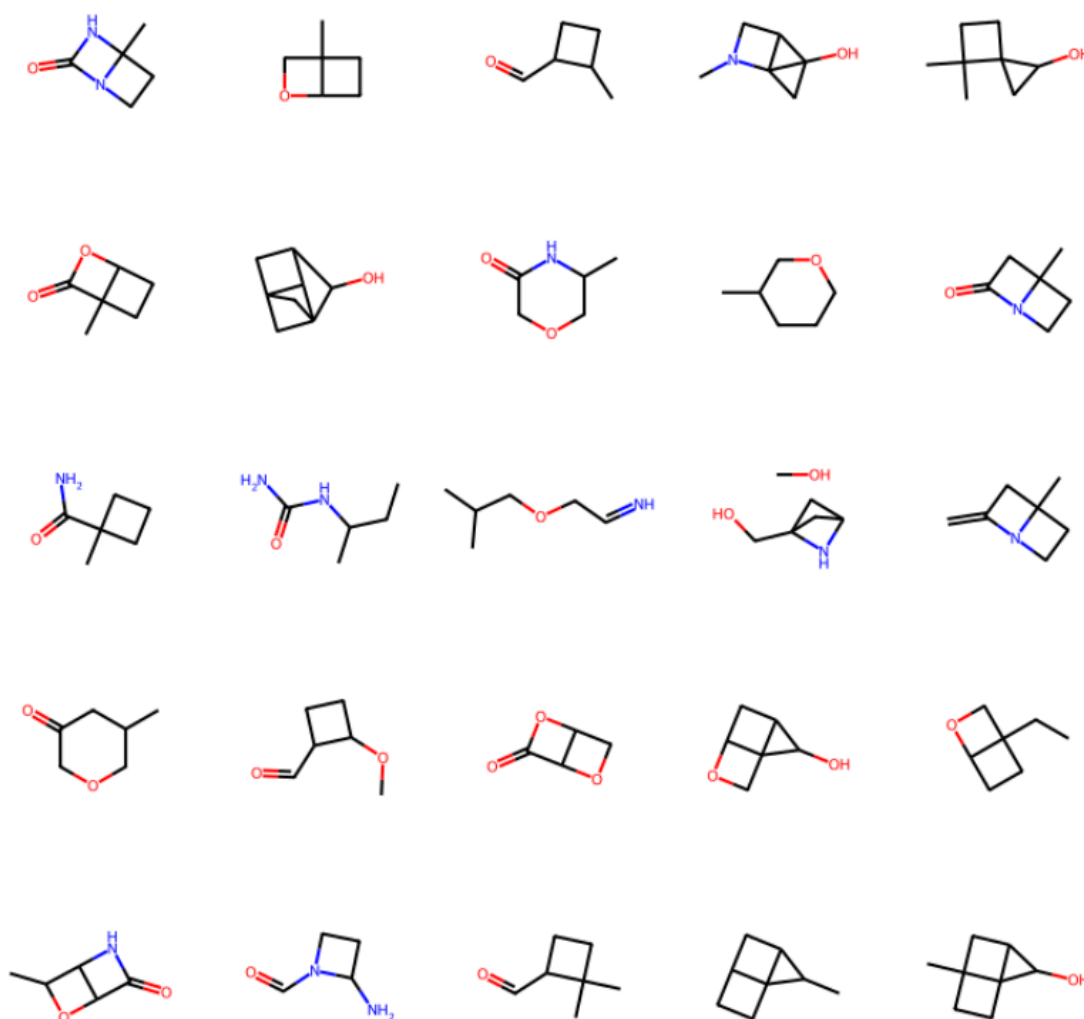
Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [36]

Powyższy rysunek czterdziesty trzeci, reprezentuję wynik otrzymany w efekcie trenowania sieci, gdzie zostały wykorzystane wyżej wspomniane algorytmy optymalizacji. Nadam dla generatora, a Adam dla dyskryminatora. Analiza tego rysunku, nie wykazała obecności identycznych molekuł, co oznacza, że daną architekturę można wykorzystać w kolejnych eksperymentach.

Taki eksperyment, z wykorzystaniem danych optymalizatorów, wykonano także dla zmiennej *dropout\_rate*, której wartość wiązała się z wyłączeniem sześćdziesięciu procent neuronów. Niestety przy takich ustawieniach hiperparametrów, w rezultacie zawsze pojawiała się para identycznych molekuł, co z kolei oznaczało że dana architektura, nie daje możliwości osiągnięcia optymalnych wyników.

Następna architektura, która w pierwotnym trenowaniu sieci neuronowej, pozwoliła otrzymać wynik, gdzie pozyskane molekuły były unikalne. W tym przypadku były

wzięte następujące algorytmy optymalizacji Adagrad - dla generatora, oraz Nadam, dla dyskryminatora.



Rysunek 44. Wynik: Adagrad dla generatora, a Nadam dla dyskryminatora.

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [36]

Powyższy rysunek czterdziesty czwarty był otrzymany, w wyniku trenowania sieci, gdzie algorytm optymalizacji Adagrad, był wykorzystany do trenowania generatora, a algorytm Nadam, był wykorzystany do trenowania dyskryminatora. Analiza danego rysunku, nie wykrywa obecności identycznych molekuł, co czyni daną architekturę istotną dla dalszych badań.

W związku z tym że dane wyniki wyglądały obiecująco, za pomocą danych optymalizatorów, powtórzono eksperyment, gdzie zmienna *dropout\_rate*, przyjęła wartość 0,6, zarówno dla generatora, jak i dyskryminatora. Niestety w efekcie danego trenowania sieci neuronowej, nie udało się otrzymać pożądanych wyników.

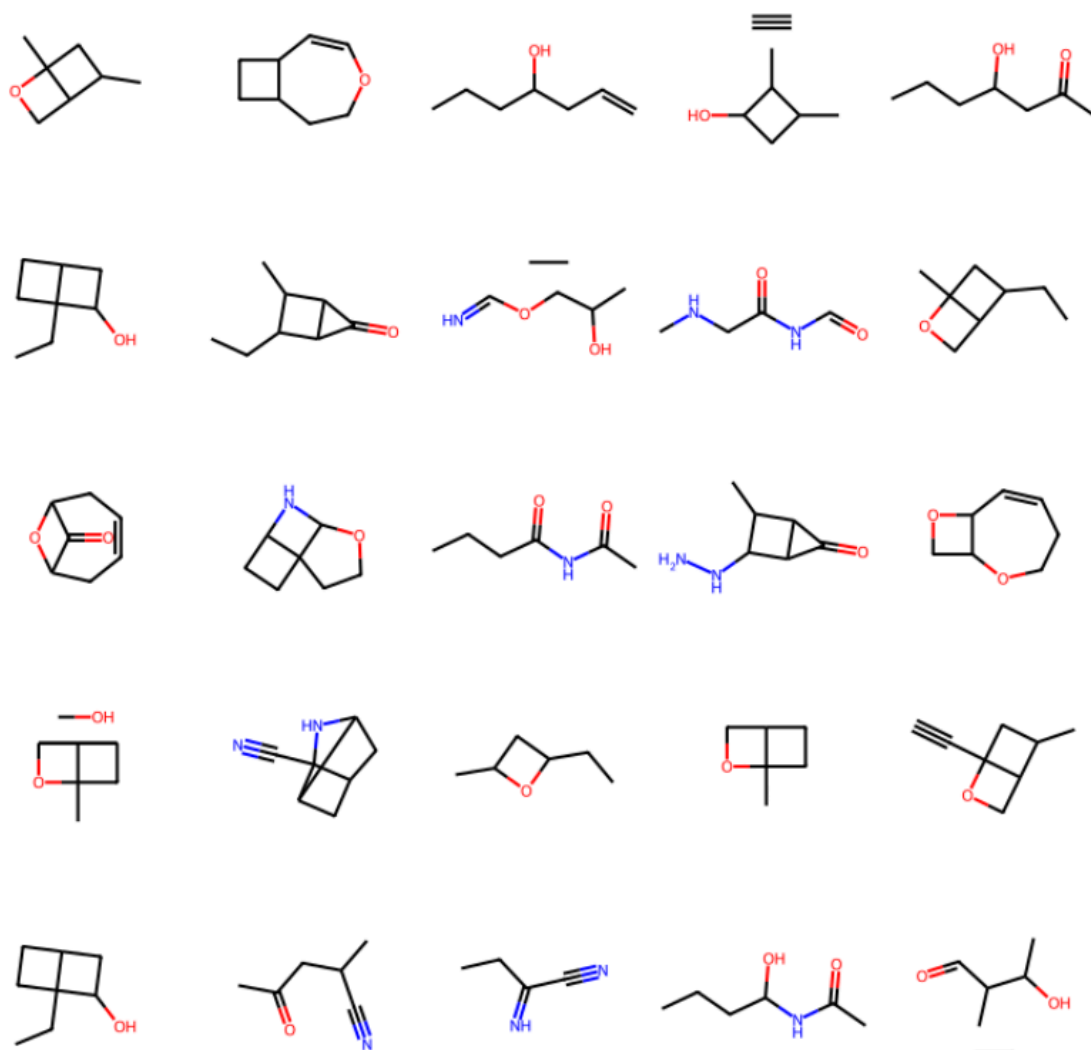
## 5. 4 Regulacja zmiennej *batch size*

Istotność zmiennej, odpowiadającej za określenie wielkości wsadu, została opisana powyżej w podrozdziale piątym rozdziału czwartego. W tym podrozdziale, głównie opisano efekt, który okazała dana zmienna na otrzymane wyniki trenowanej sieci.

Pierwsza wartość dla zmiennej *batch\_size*, która zostanie opisana, składa się z ośmiu próbek danych, które są przetwarzane równolegle w jednej iteracji algorytmu uczenia. Analiza wyniku danego trenowania, wykazała występowanie trzech identycznych molekuł.

Kolejna analizowana wartość to szesnaście, jest to wartość wykorzystana w kodzie podstawowym. W tym wyniku, analiza wykryła dwie pary identycznych molekuł. Warto zaznaczyć, że dany wynik, można również uważać za powtarzanie kodu podstawowego, i jak wskazuje otrzymany wynik, potrzebna dalsza analiza hiperparametrów w celu otrzymania optymalnie działającej sieci neuronowej.





Rysunek 45. Wynik trenowania sieci, gdzie batch\_size był równy 32.

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [36]

Następnie, trenowano sieć w której wykorzystano wsad, składający się z trzydziestu dwóch próbek. Wynik, który otrzymano w rezultacie trenowania wygląda najbardziej obiecująco w porównaniu do poprzednio analizowanych. W efekcie danego trenowania, udało się otrzymać powyższy rysunek czterdzieści pięć, gdzie wszystkie dwadzieścia pięć molekuł są unikalne. Brak występowania identycznych molekuł oznacza to że, dana wartość zmiennej zostanie wykorzystana w dalszym przeglądzie danej sieci neuronowej.

Ostatnia wartość której przejrano się w tym podrozdziale, to wartość zmiennej *batch\_size* równa sześćdziesięciu czterem. W wyniku trenowania sieci, gdzie mini-partia składała się z sześćdziesięciu czterech próbek, wykryto dwie pary identycznych molekuł, co oznacza, że dana wartość, nie będzie wykorzystywana do



We wcześniej opisanych badaniach, czasem udawało się zminimalizować pojawianie się identycznych grafów cząsteczek, za pomocą regulacji różnych hiperparametrów, lecz takie rozwiązanie okazało się być mało skuteczne.

Poniższy rysunek czterdzieści dwa reprezentuje modyfikację wprowadzoną do kodu, który jest odpowiedzialny za generowanie graficznego przedstawienia molekuł, które później ulegną analizie. Wcześniejsze badania, stają się punktem odniesienia, który ujawnia jakie problemy można napotkać bez zastosowania filtra unikalności. A dalsze badania pozwolą sprawdzić wpływ filtra na skuteczność i jakość danego rozwiązania.

```
unique_molecules = set()
for i in range(batch_size):
    molecule = graph_to_molecule([adjacency[i].numpy(), features[i].numpy()])
    if molecule is not None:
        smiles = MolToSmiles(molecule)
        unique_molecules.add(smiles)

unique_molecules = list(unique_molecules)[:batch_size]

return [
    Chem.MolFromSmiles(smiles) for smiles in unique_molecules
]
```

Rysunek 46. Fragment kodu wprowadzający filtr unikalności.

*Źródło: Opracowane własne A. Vezdenetska na podstawie sieci neuronowej [36]*

Ze względu na to że za pomocą filtra można powtarzać kod podstawowy, nie obawiając się otrzymania zduplikowanych cząsteczek. To pozwoli powtórzyć wcześniej opisane eksperymenty opisane w podrozdziale 5. 4. Ze względu na to że większość architektur opisanych powyżej generowała przynajmniej jedną parę zduplikowanych molekuł, to nie pozwalało oceniać tę architektury jako optymalne. Z kolei zastosowanie filtra unikalności pozwoli otrzymać lepsze wyniki.

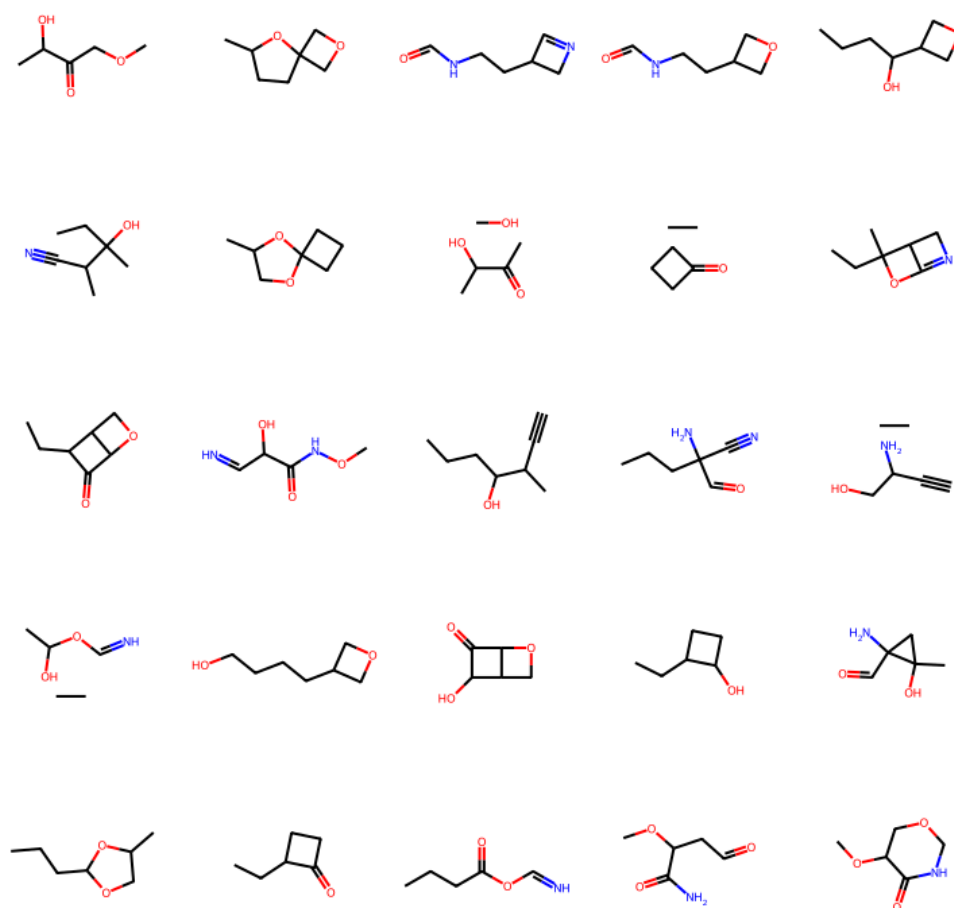
### 5. 5. 1 Eksperymenty z filtrem unikalności

W tym podrozdziale zostaną powtórzone i opisane wyniki badań, które zostały wybrane, jako najbardziej wydajne, czyli te które pozwoliły otrzymać wyniki gdzie było najmniej duplikatów. Zastosowanie danego filtra, pozwala skupić się także na tym, by wyeliminować architektury sieci, które generują wynik, gdzie cząsteczka

rozpada się, a obok pojawiają się cząsteczki etanu ( $C_2H_6$ ) [2]. Ze względu na to że filtr “widzi” cząsteczkę jako całość, on nie będzie jej eliminował, jeśli nie zostanie powtórzony całokształt molekuly.

#### 5. 5. 1. 1 Trenowanie sieci o podstawowej architekturze

Podstawowa architektura, czyli ta, która nie ulegała modyfikacji, to dokładnie ta sama architektura, która została przedstawiona w pracy “WGAN-GP with R-GCN for the generation of small molecular graphs”, z zastosowaniem filtra unikalności dla generowanego wyniku. Jego zastosowanie pomaga wyeliminować wszystkie duplikaty molekuł.



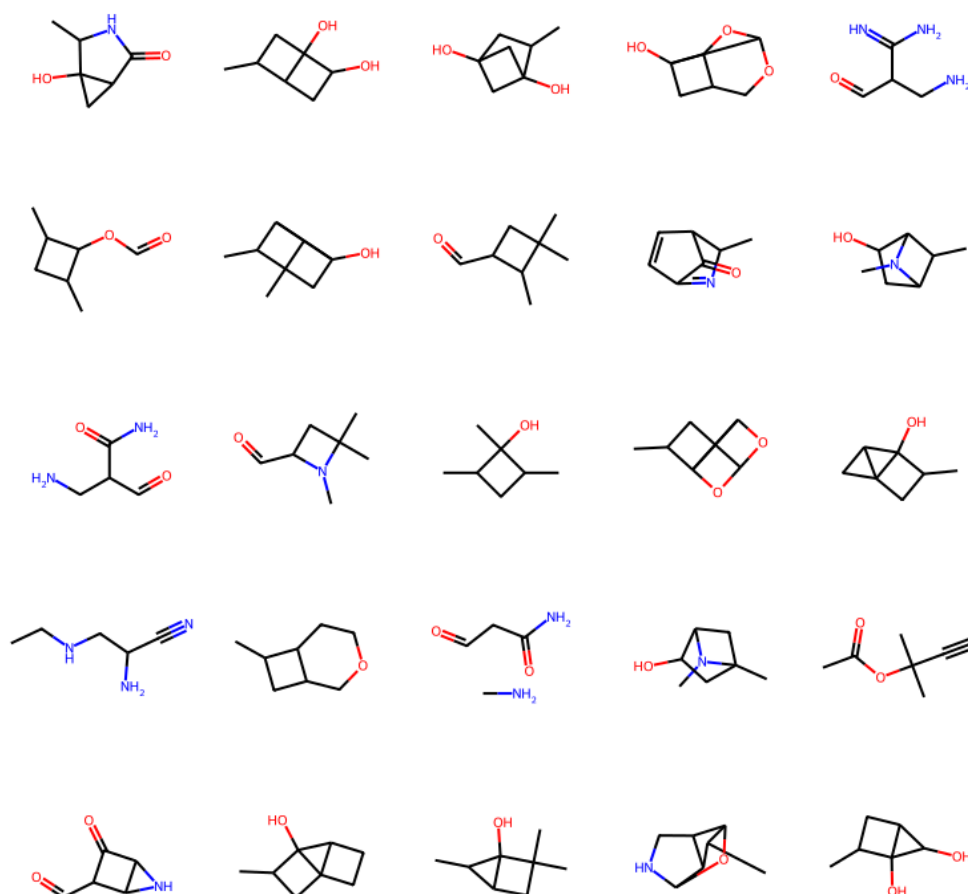
Rysunek 47. Wynik eksperymentu opisanego w podrozdziale 5. 5. 1. 1.

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [36]

Powyższy rysunek przedstawia jeden z dziesięciu wyników, otrzymanych w trakcie trenowania powyżej opisanej architektury sieci. Przegląd otrzymanych wyników, pozwala twierdzić że filtr unikalności pozwolił otrzymać dwadzieścia pięć unikalnych struktur chemicznych. Jednak, warto zwrócić uwagę na to że zostały wygenerowane trzy identyczne cząsteczki etanu ( $C_2H_6$ ) [2], które zostały odebrane przez filtr jako część jednej molekuly.

#### 5. 5. 1. 2 Trenowanie sieci o wybranej wartości zmiennej *dropout\_rate*

Wybrana została architektura, gdzie zostały wyłączona sześćdziesiąt procent neuronów generatora i dyskryminatora. Poniższy rysunek reprezentuję jeden z dziesięciu otrzymanych wyników.



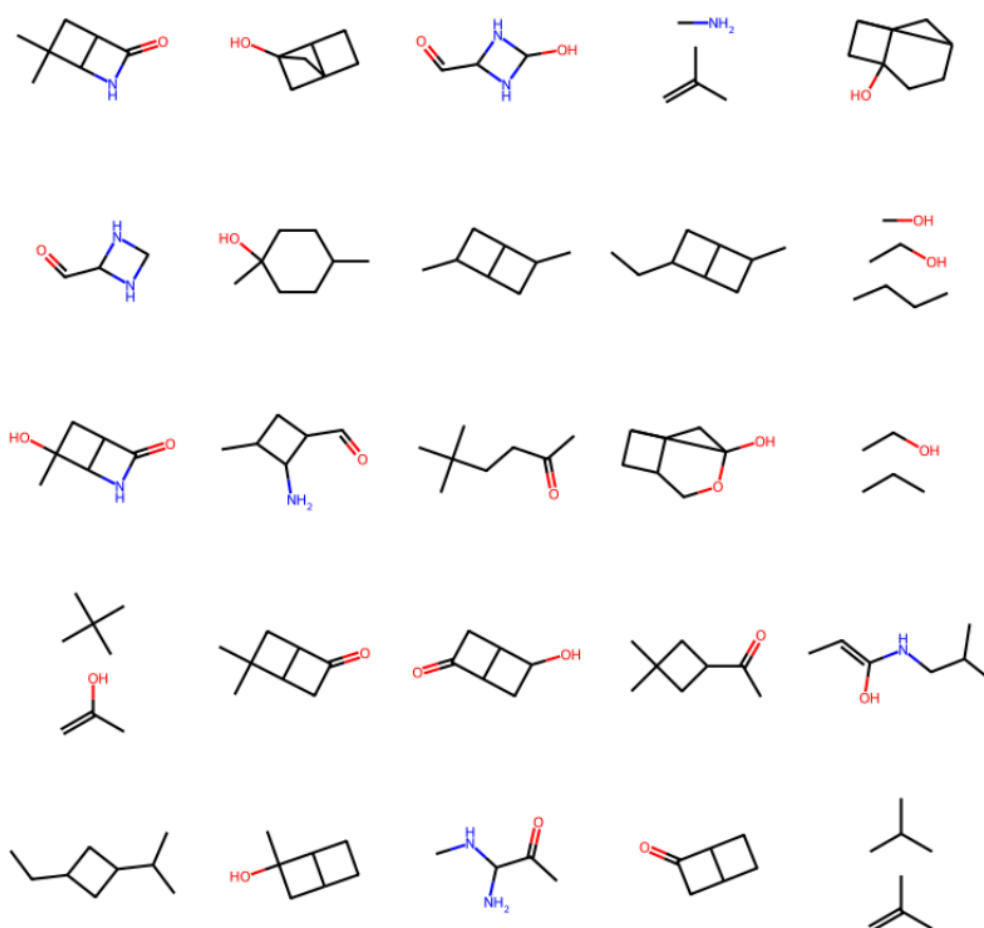
Rysunek 48. Wynik eksperymentu opisanego w podrozdziale 5. 5. 1. 2.

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [36]

Analiza powyższego rysunku, wykazała obecność trzech identycznych molekuł izopropanolu. Dwie z nich, przedstawione jako jedna cząsteczka. Inna cząsteczka również zawiera molekułę izopropanolu, i reprezentują się z nią jako jedna cząsteczka. Ze względu na to że cząsteczki te były odebrane jako jedna całość, nie zostały one wyeliminowane za pomocą filtra.

### 5. 5. 1. 3 Trenowanie sieci o wybranych optymalizatorach

Dany eksperyment wykonano za pomocą zmiany optymalizatorów w architekturze sieci neuronowej. Dla generatora wykorzystano optymalizator Adagrad, natomiast optymalizator Nadam dla dyskryminatora.

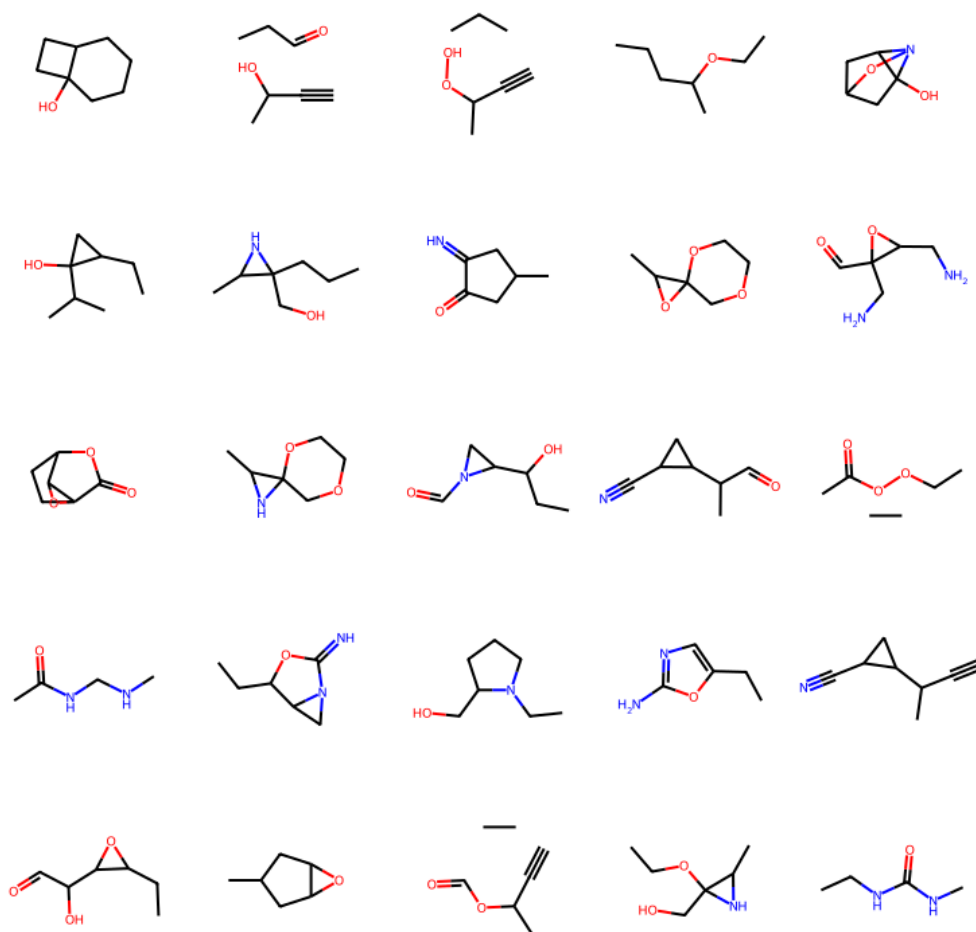


Rysunek 49. Wynik eksperymentu dla optymalizatorów Adagrad i Nadam.

Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [36]

Analiza powyższego rysunku, wykazała obecność dwóch identycznych molekuł etanolu ( $C_2H_5OH$ ) [2]. Spośród wszystkich dziesięciu wyników, otrzymanych w tym badaniu, tylko w tym jednym pojawił się dany problem. Oznacza to że ten model jest bardziej wydajny w porównaniu do wyżej analizowanych.

Następnie wybrano optymalizator Nadam dla generatora, a optymalizator Adam dla dyskryminatora. Poniższy rysunek prezentuję jeden z wyników otrzymanych, w trakcie dziesięciokrotnego trenowania sieci.



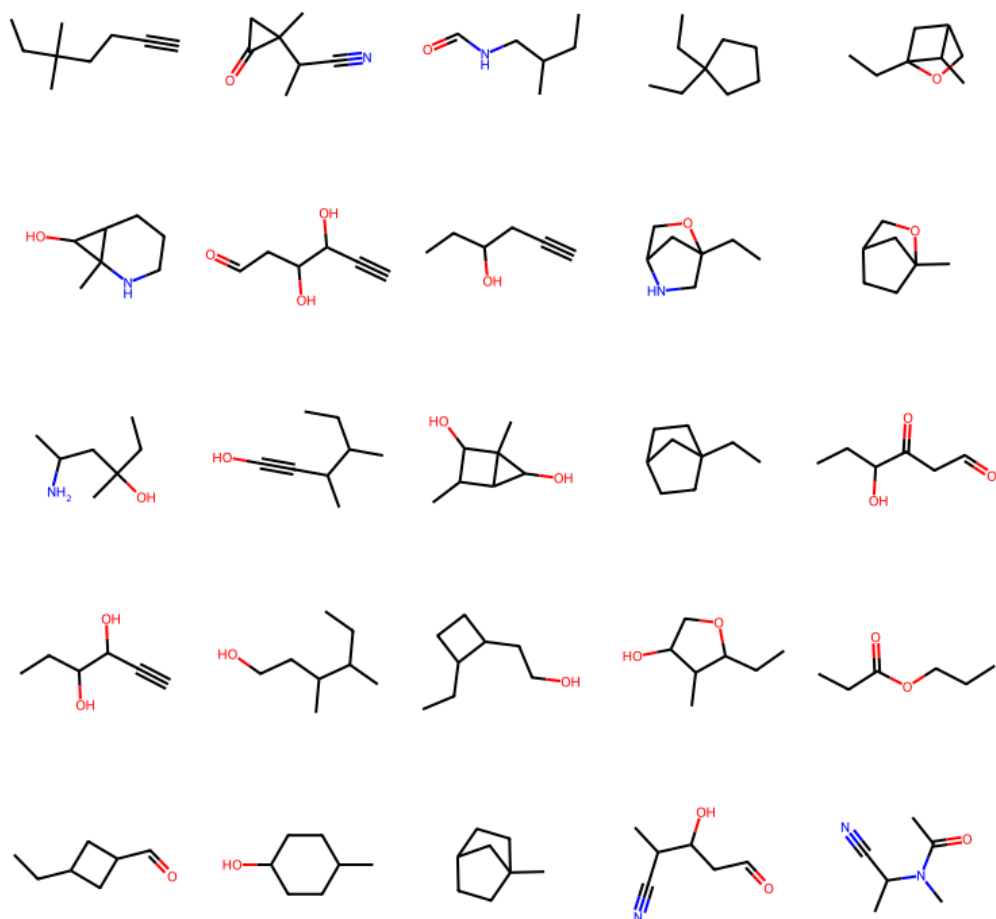
Rysunek 50. Wynik eksperymentu dla optymalizatorów Nadam i Adam.

*Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [36]*

Analiza powyższego rysunku, wykazała obecność dwóch molekuł etanu ( $C_2H_6$ ), które przez sieć odbierane są, jako część cząsteczki obok której się znajdują [2]. Spośród wszystkich dziesięciu wyników w połowie wyników, pojawia się powtarzająca się cząsteczka etanu.

#### 5. 5. 1. 4 Trenowanie sieci o wybranym wsadzie

Ostatnia architektura którą przeanalizujemy, to ta w której wsad trenowanej sieci został zwiększony dwukrotnie. Wsad w analizowanej dalej architekturze składa się trzydziestu dwóch próbek.



Rysunek 51. Wynik eksperymentu 5. 5. 1. 4.

*Źródło: Opracowanie własne A. Vezdenetska na podstawie sieci neuronowej [36]*

Powyższy rysunek przedstawia wynik, otrzymany w trakcie jednego z dziesięciu trenowań sieci neuronowej. Tak jak przedstawia rysunek, nie można w nim odnaleźć żadnych identycznych molekuł, ani ułamków molekuł, które mogły by spowodować otrzymanie duplikatów cząsteczek. Pomimo to, dana architektura nie jest idealna, gdyż w wyniku dwóch z dziesięciu trenowań sieci, otrzymano wynik, gdzie pojawiają się ułamki większych cząsteczek, takie jak etan, etanol lub inne.



### **5. 5. 2 Wnioski podrozdziału:**

Jak wykazują otrzymane wyniki, zastosowanie filtra unikalności pozwoliło wyeliminować wszystkie duplikaty molekuł. Zastosowanie wcześniej wybranych architektur pozwoliło otrzymać wyniki, gdzie ilość ułamków molekuł, które mogłyby się powtarzać była mniejsza niżeli w podstawowym kodzie. Ułamki przez filtr odbierane jako cała cząsteczka, dlatego dany filtr nie może wyeliminować duplikatów takich ułamków.

## **5. 6 Podsumowanie przeglądanej pracy**

Dodanie filtra unikalności do przeglądanego modelu sieci neuronowej, pozwoliło uzyskać dwadzieścia pięć unikalnych cząsteczek w efekcie każdego trenowania sieci. Poprzednie eksperymenty pozwoliły wybrać optymalne ustawienia, które można było wykorzystać dla dalszej analizy.

Najlepszy wynik otrzymany przy trenowaniu sieci neuronowej wybranej w tym rozdziale udało się otrzymać, wykorzystując architektury wcześniej analizowane.

Również, interesujący wynik, udało się otrzymać przy zmianie optymalizatorów wraz z wykorzystaniem filtra unikalności. Dokładnie to był następny układ, dla generatora wykorzystano optymalizator Adagrad, a dla dyskryminatora wzięto optymalizator Nadam. W tym przypadku dziewięć z dziesięciu otrzymanych grafów, przedstawiało tylko unikalne związki chemiczne, bez powtórzeń innych małych molekuł.

W związku z otrzymanymi wynikami, można stwierdzić że wykorzystanie filtra unikalności pozwoliło otrzymać w całości unikalne molekuły. Natomiast wykorzystanie wiedzy zdobytej w eksperymentach opisanych we wcześniejszych podrozdziałach pozwoliło udoskonalić końcowy wynik, gdyż udało się wyeliminować małe identyczne cząsteczki, które model widział jako całość, lub część innej cząsteczki.

# ZAKOŃCZENIE

Dana praca została poświęcona przeglądowi dwóch sieci neuronowych, które wykorzystują dwie różne techniki używane w dziedzinie uczenia maszynowego. Obydwa przeglądanych podejścia stosowane są do generowania nowych, unikalnych cząsteczek. W pierwszej przeglądanej pracy wykorzystaną technikę VAE, czyli technika z wykorzystaniem autoenkodera wariacyjnego. W drugiej pracy natomiast, wykorzystano hybrydowe podejście gdzie R-GCN tworzy zakodowaną strukturę molekuł, podczas gdy WGAN-GP generuje nowe różnorodne graficzne reprezentację cząsteczek w oparciu o dane otrzymane od R-GCN.

## 6. 1 Porównanie przeglądanych prac

Porównując przeglądane prace warto także zaznaczyć, że różnią się one nie tylko technikami, które zostały użyte dla trenowania każdej sieci neuronowej. Każda z nich wykorzystują różne bazy danych, dla przykładu dla trenowania sieci opisanej w pracy *“Drug Molecule Generation with VAE”* wykorzystaną bazę danych ZINC, natomiast w pracy *“WGAN-GP with R-GCN for the generation of small molecular graphs”* wykorzystaną bazę danych QM9.

Baza danych ZINC, jest bezpłatną bazą danych w której zebrano bibliotekę o zawartości 727 842 cząsteczek. ZINC zawiera informacje o strukturze związków chemicznych, w tym ich identyfikatory, dane dotyczące dostępności rynkowej oraz inne informacje związane z właściwościami chemicznymi i fizycznymi [17]. Dany zbiór molekuł, bez ograniczenia ilości ciężkich atomów, pozwala na przykładzie takiej bazy danych otrzymać różnorodne cząsteczki w wyniku uczenia sieci neuronowej.

Z kolei baza danych QM9 zawiera informacje o stu trzydziestu czterech tysiącach unikalnych molekuł. Wybrane cząsteczki zawierają maksymalnie dziewięć atomów ciężkich: węgla (C), azotu (N), tlenu (O), fluoru (F). Dane obejmują szeroki zakres właściwości i parametrów molekularnych, takich jak energia, entropia, ciepło właściwe, momenty dipolowe, polaryzowalność, elektryczność, długość wiązań, kąty wiązań, a także inne kluczowe właściwości [27; 29]. Baza "QM9" jest zoptymalizowana do analizy molekuł o niewielkich rozmiarach, w związku z czym w trakcie trenowania sieci nie otrzymano wielkich cząsteczek.

Porównując obie bazy danych, "QM9" jest bardziej zoptymalizowana do analizy molekuł o niewielkich rozmiarach, a z kolei "ZINC" jest bardziej rozbudowaną bazą danych, która może zawierać różnorodne molekuły.

Porównując oba podejścia w pierwszej przeglądanej pracy, zatytułowanej "*Drug Molecule Generation with VAE*", technika VAE jest probabilistycznym modelem autoenkodera i wyraźnie modeluje podstawowy rozkład prawdopodobieństwa danych. Za pomocą elementu wariacyjności ten model powinien generować bardziej różnorodne dane na wyjściu. Jest to możliwe, gdyż w tym modelu enkoder przekształca dane wejściowe w rozkład prawdopodobieństwa w przestrzeni ukrytej, co pozwala otrzymać różnorodne reprezentacje w przestrzeni ukrytej. Dodatkowo element losowego pobierania próbek z przestrzeni ukrytej ma przekładać się na bardziej zróżnicowane wyniki. Żeby otrzymywane wyniki były bliższe rzeczywistości, funkcja straty minimalizuje różnicę między rozkładem prawdopodobieństwa przestrzeni ukrytej, a rozkładem prawdopodobieństwa reprezentacji danych wejściowych.

W drugiej, pracy, której przegląd został wykonany, zostało zastosowane hybrydowe podejście. W pracy "*WGAN-GP with R-GCN for the generation of small molecular graphs*" połączono dwie różne technologie: WGAN-GP i R-GCN, aby otrzymać realistyczne graficzne przedstawienia molekuł. Dane podejście polega na zastosowaniu WGAN-GP do modelowania generatywnego i R-GCN do modelowania relacji między elementami w grafach molekuł.

## 6. 2 Porównanie otrzymanych wyników.

Jeśli porównamy te dwa podejścia, model z zastosowaniem wariacyjnego autoenkodera "*Drug Molecule Generation with VAE*" daje gorsze wyniki, w porównaniu do wyników, otrzymywanych w czasie przeglądu pracy "*WGAN-GP with R-GCN for the generation of small molecular graphs*".

W trakcie przeglądu pierwszej pracy o tytule "*Drug Molecule Generation with VAE*", większość otrzymanych wyników w trakcie badań wskazała na to że ilość wygenerowanych unikalnych cząsteczek, jest dość niska. Wynik najczęściej staje się lepszy, kiedy sieć jest przeciążona i zaczyna generować dane bliższe szumu niż danych

rzeczywistych. W przypadku danej sieci takie przeciążenie pozwoliło otrzymać wyniki oddalone od danych wejściowych, ale przy tym bardziej zróżnicowane.

Można powiedzieć że zastosowanie hybrydowego podejścia w drugiej pracy, stało się jednym z głównych czynników, przy którym otrzymano zróżnicowane cząsteczki. Dzięki czemu, od początku generowała się wielka ilość różnorodnych cząsteczek, problem występowania graficznych duplikatów był rzadki w porównaniu do pierwszej pracy. W celu poprawienia danych wyników, została przeprowadzona seria badań, która pomogła odnaleźć rozmaite architektury sieci warte uwagi. Zastosowanie filtra unikalności całkowicie rozwiązało problem generowania duplikatów molekuł.

Połączenie w tym modelu WGAN-GP i R-GCN, pozwoliło stworzyć nie tylko różnorodne, ale także realistyczne molekuły. To połączenie może pomóc w procesie tworzenia różnorodnych zbiorów danych chemicznych.

## BIBLIOGRAFIA:

### Pozycje zwarte:

1. Bear M. F., Connors B. W., Paradiso M. A.: *Neuroscience: Exploring the Brain*. Philadelphia 2007.
2. Bruice P. Y.: *Organic Chemistry*. New Jersey 2016.
3. Géron A.: *Hands-On Machine Learning with Scikit-Learn and TensorFlow Concepts, Tools, and Techniques to Build Intelligent Systems*. Sebastopol 2017.
4. Goodfellow I., Bengio Y., Courville A.: *Deep Learning*. Cambridge 2016.
5. Haykin S.: *Neural Networks and Learning Machines*. New Jersey 2009.
6. Raschka S., Mirjalili V.: *Python Machine Learning*. Birmingham 2017.
7. Russell S., Norvig P.: *Artificial Intelligence: A Modern Approach*". New Jersey 2010.
8. Shalev-Shwartz Sh., Ben-David Sh.: *Understanding Machine Learning: From Theory to Algorithms*. Cambridge 2014.
9. Szaleniec M., Tadeusiewicz R.: *Leksykon sieci neuronowych*. Wrocław 2015.

### Artykuły w czasopismach:

10. Arjovsky M., Chintala S., Bottou L.: "Wasserstein GAN". arXiv 2017.
11. Doersch C.: "Tutorial on Variational Autoencoders". arXiv 2021.
12. Duchi J., Hazan E., Singer Y.: *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization*". J.of Machine Learning Research 2011.
13. Gulrajani I., Faruk A., Arjovsky M., Dumoulin V., Courville A.: *Improved Training of Wasserstein GANs*. arXiv 2017.
14. Han S. H. i in.: *Artificial Neural Network: Understanding the Basic Concepts without Mathematics*. NCBI 2018.
15. Hinton G. E. i in.: *Improving neural networks by preventingco-adaptation of feature detectors*. arXiv 2012
16. Ioffe S., Szegedy Ch.: *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. arXiv 2015.

17. Irwin J. J., Shoichet B. K.: *ZINC – A Free Database of Commercially Available Compounds for Virtual Screening*. NCBI 2006.
18. Jensen W. B.: *Abegg, Lewis, Langmuir, and the Octet Rule*. Rochester Institute of Technology, Rochester 2009.
19. Juorio A. V.: *The Synaptic Organization of the Brain, 4th edition*. J Psychiatry Neurosci 1998.
20. Kingma D., Ba J.: *Adam: A method for stochastic optimization*. arXiv 2014.
21. Kingma D., Welling M.: "An Introduction to Variational Autoencoders". arXiv 2019.
22. Kingma D., Welling M.: "Auto-Encoding Variational Bayes". arXiv 2022.
23. Kipfa Th. N. i in.: *Relational Graph Convolutional Networks*. arXiv 2017.
24. Liu Y. i in.: *Variational Autoencoder: An Unsupervised Model for Learning Latent Representations*. IEEE Access. 2020.
25. Lengerich B., Xing E. P., Caruana R.: *Dropout as a Regularizer of Interaction Effects*. arXiv 2021.
26. Milne T., Nachman A.: *Wasserstein GANs with Gradient Penalty Compute CongestedTransport*. arXiv 2022.
27. Ramakrishnan R. i in.: *Quantum chemistry structures and properties of 134 kilocal/mol molecular formation enthalpies*. Journal of Chemical Physics 2014.
28. Ruder, S.: *An overview of gradient descent optimization algorithms*. arXiv 2017.
29. Ruddigkeit L., i in.: *Enumeration of 166 billion organic small molecules in the chemical universe database GDB-17*. J. Chem. Inf. 2012.
30. Schaul T., S. Zhang, LeCun Y.: *No more pesky learning rates*. arXiv 2013.
31. Schlichtkrull M., Kipf Th. N., Bloem P., van den Berg R., Titov I., Welling M.: "Modeling Relational Data with Graph Convolutional Networks". arXiv 2017.
32. Sun M., i in.: *Graph convolutional networks for computational drug development and discovery*. Brief Bioinform. 2020.
33. Weininger D.: *SMILES, a chemical language and information system*" J. Chem. Inf. Comput. Sci. 1988.
34. Zeiler M. D.: *ADADELTA: An Adaptive Learning Rate Method*. arXiv 2012

## Strony internetowe:

35. Busu V.: *“Drug Molecule Generation with VAE”*. Keras. 2022 (Dostęp: 24.02.2023: [https://keras.io/examples/generative/molecule\\_generation/](https://keras.io/examples/generative/molecule_generation/)).

36. Kensert A.: *“WGAN-GP with R-GCN for the generation of small molecular graphs”*. 2021. (Dostęp: 13.04.2023: <https://keras.io/examples/generative/wgan-graphs/>).

37. Przyborowski M.: *“Modele mieszanin Gaussowskich i odległość Wassersteina w zagadnieniu porównywania dużych zbiorów danych”*. 2021 (Dostęp: 08.05.2023: [https://www.mimuw.edu.pl/aktualnosci/seminaria/modele-mieszanin-gaussowskich-i-o-dleglosc-wassersteina-w-zagadnieniu#:~:text=Odległość%20Wassersteina%20\(równie%20znana%20jako,przekształcenia%20jednego%20rozkładu%20w%20drugi\)](https://www.mimuw.edu.pl/aktualnosci/seminaria/modele-mieszanin-gaussowskich-i-o-dleglosc-wassersteina-w-zagadnieniu#:~:text=Odległość%20Wassersteina%20(równie%20znana%20jako,przekształcenia%20jednego%20rozkładu%20w%20drugi).)).

38. Pramoditha R.: *The Concept of Artificial Neurons (Perceptrons) in Neural Networks*. 2021 (Dostęp: 24.02.2023: <https://towardsdatascience.com/the-concept-of-artificial-neurons-perceptrons-in-neural-networks-fab22249cbfc>).

39. Singh G.: *“Introduction to Artificial Neural Networks datascience”*. 2023 (Dostęp: 24.08.2023 <https://www.analyticsvidhya.com/blog/2021/09/introduction-to-artificial-neural-networks/>).

## Materiały niepublikowane:

40. Dozart T.: *“INCORPORATING NESTEROV MOMENTUM INTO ADAM”*. OpenReview 2016.

# SPIS RYSUNKÓW

Rysunek 1. Budowa neuronu	8
Rysunek 2. Struktura sieci neuronowej ze sprzężeniem do przodu	10
Rysunek 3. Ogólna struktura autoenkodera.	13
Rysunek 4. Unikalne molekuly, zebrane z kilku epok	23
Rysunek 5. Wynik dla wartości zmiennej dropout_rate 0.0	26
Rysunek 6. Wynik dla wartości zmiennej dropout_rate 0.1	27
Rysunek 7. Wynik dla wartości zmiennej dropout_rate 0.2	27
Rysunek 8. Wynik dla wartości zmiennej dropout_rate 0.3	28
Rysunek 9. Wynik dla wartości zmiennej dropout_rate 0.4	28
Rysunek 10. Wynik dla wartości zmiennej dropout_rate 0.5	29
Rysunek 11. Część wyniku dla wartości zmiennej dekodera dropout_rate 0.0	31
Rysunek 12. Wynik dla wartości zmiennej dekodera dropout_rate 0.1	32
Rysunek 13. Wynik dla wartości zmiennej dekodera dropout_rate 0.2	32
Rysunek 14. Wynik dla wartości zmiennej dekodera dropout_rate 0.3	33
Rysunek 15. Wynik dla wartości zmiennej dekodera dropout_rate 0.4	33
Rysunek 16. Wynik dla wartości zmiennej dekodera dropout_rate 0.5	34
Rysunek 17. Część wyniku, dropout_rate enkodera i dekodera jest równe 0.1	35
Rysunek 18. Wynik dla optymalizatora Nadam	39
Rysunek 19. Wynik przy wartości zmiennej batch size 256	41
Rysunek 20. Wynik przy wartości zmiennej batch size 128	41
Rysunek 21. Wynik przy wartości zmiennej batch size 64	42
Rysunek 22. Wynik przy wartości zmiennej batch size batch size 32	43
Rysunek 23. Wynik przy wartości zmiennej batch size 100	44
Rysunek 24. Część wyniku, momentum 0,3	46
Rysunek 25. Część wyniku, momentum 0,6	46
Rysunek 26. Część wyniku, momentum 0,9	47
Rysunek 27. Wynik, 1024 i 256 neuronów.	48
Rysunek 28. Wynik, 1024 i 1024 neuronów.	49
Rysunek 29. Wynik, 1024, 512, 256 neuronów.	49
Rysunek 30. Uzyskany wynik z dziesięciu epok. 512, 256, 512	51
Rysunek 31. Powtórzenie eksperymentu.	55



Rysunek 32. Wynik dropout_rate generatora równy 0.0.	56
Rysunek 33. Wynik dropout_rate generatora równy 0.2.	57
Rysunek 34. Wynik dropout_rate generatora równy 0.4.	57
Rysunek 35. Wynik dropout_rate generatora równy 0.6.	58
Rysunek 36. Wynik dropout_rate generatora równy 0.8.	58
Rysunek 37. Wynik dropout_rate dyskryminatora równy 0.0.	60
Rysunek 38. Wynik dropout_rate dyskryminatora równy 0.2.	61
Rysunek 39. Wynik dropout_rate dyskryminatora równy 0.4.	61
Rysunek 40. Wynik dropout_rate dyskryminatora równy 0.6.	62
Rysunek 41. Wynik dropout_rate dyskryminatora równy 0.8.	63
Rysunek 42. Dropout 0,6 dla dekodera i enkodera..	64
Rysunek 43. Wynik: Nadam dla generatora, a Adam dla dyskryminatora.	67
Rysunek 44. Wynik: Adagrad dla generatora, a Nadam dla dyskryminatora.	68
Rysunek 45. Wynik trenowania sieci, gdzie batch_size był równy 32.	69
Rysunek 46. Fragment kodu wprowadzający filtr unikalności.	71
Rysunek 47. Wynik eksperymentu opisanego w podrozdziale 5. 5. 1. 1.	72
Rysunek 48. Wynik eksperymentu opisanego w podrozdziale 5. 5. 1. 2.	73
Rysunek 49. Wynik eksperymentu dla optymalizatorów Adagrad i Nadam.	74
Rysunek 50. Wynik eksperymentu dla optymalizatorów Nadam i Adam.	75
Rysunek 51. Wynik eksperymentu dla optymalizatorów Nadam i Adam.	76

## SPIS TABEL

Tabela 1. Lista wybranych przykładów alkanów	20
Tabela 2. Lista wybranych przykładów alkenów	21
Tabela 3. Lista wybranych przykładów alkinów	21
Tabela 4. Lista wybranych przykładów alkinów	22
Tabela 5. Wyniki testów wykonanych dla podrozdziału 4. 2.	29
Tabela 6. Wyniki testów wykonanych dla podrozdziału 4. 3.	34
Tabela 7. Wyniki otrzymane w trakcie weryfikacji zmiennej batch size	44