

Raport z Projektu

Generowanie pixelartów minecraft

Generator Frajdy

Wakacyjne Wyzwanie Solvra 2025

28 września 2025



Zespół: Kamil Borkowski, Patryk Mikołajewicz, Jakub Wieśniak, Jakub Smolarczyk
Koordynator: Szymon Woźniak

Spis treści

Streszczenie	2
1 Wprowadzenie	3
1.1 Cel projektu	3
1.2 Zakres projektu	3
2 Metody	3
2.1 GAN	4
2.2 Model dyfuzyjny	5
2.3 Variational Autoencoder (VAE)	6
2.4 Conditional VQ-VAE	7
2.5 Transformer + Conditional VQ-VAE	8
3 Wyniki i analiza	9
3.1 Wyniki Conditional VQ-VAE	9
3.2 Wyniki Transformer + Conditional VQ-VAE	10
4 Dyskusja	10
4.1 Napotkane problemy i ograniczenia	10
4.2 Dalszy rozwój	10
4.3 Podsumowanie	11
Bibliografia	11



Streszczenie

Główne założenie projektu stanowiło wytrenowanie modelu generatywnego, który będzie tworzył nowe pixel art'y do gry minecraft. W tym celu zbadaliśmy działanie wybranych architektur (VAE, GAN, modele dyfuzyjne) oraz ich różne warianty. Największą skuteczność w rekonstrukcji danych wejściowych wykazał model conditional VQ-VAE, do którego dołączliśmy potem zaimplementowanego przez nas transformera w celu możliwości generowania nowych tekstur. Do wytrenowanego modelu dopisaliśmy również aplikację web'ową dla większej wygody korzystania z modelu. Wśród największych trudności napotkanych podczas realizacji projektu należały: ograniczona ilość danych, ich niezbalansowanie oraz ograniczona moc obliczeniowa.

Słowa kluczowe: modele generatywne, VQ-VAE, transformer

1. Wprowadzenie

1.1. Cel projektu

Głównym celem naszego projektu było stworzenie rozwiązania pozwalającego na generowanie nowych pixelartów do gry minecraft na podstawie reprezentacji stworzonej za pomocą narzędzi które musielismy wybrać.

1.2. Zakres projektu

Zakres naszego projektu zawierał znalezienie oraz odpowiednie przygotowanie zbioru danych z gry minecraft, zbadanie różnych modeli reprezentacji grafik, wybranie najlepiej sprawdzającego się modelu, dostosowanie go oraz przetestowanie do naszego zadania, stworzenie api oraz webowego interfejsu użytkownika a na końcu opublikowanie rozwiązania na chmurze.

2. Metody

Do zadania reprezentacji stanu oraz generowania grafik przetestowaliśmy kilka podejść opartych na modelach głębokiego uczenia:

- VAE
- Conditional VQ-VAE
- Transformer + conditional VQ-VAE
- Model dyfuzyjny
- GAN

2.1. GAN

Generative Adversarial Network (GAN) to jedno z najczęściej stosowanych podejść do generowania danych syntetycznych, w tym obrazów. Architektura GAN składa się z dwóch sieci neuronowych uczonych jednocześnie: generatora oraz dyskryminatora. Generator uczy się mapować losowy wektor latentny $z \sim p(z)$ na próbki danych $G(z)$, natomiast dyskryminator rozróżnia pomiędzy próbками rzeczywistymi $x \sim p_{\text{data}}(x)$ a próbками wygenerowanymi. Proces uczenia przebiega w formie gry o sumie zerowej, w której generator dąży do oszukania dyskryminatora, a dyskryminator do poprawnego klasyfikowania danych. Funkcja celu przyjmuje postać:

$$\min_G \max_D V(D, G) = E_{x \sim p_{\text{data}}(x)}[\log D(x)] + E_{z \sim p_z(z)}[\log(1 - D(G(z)))].$$

[3]

Podczas przeglądu literatury rozważane było kilka modyfikacji podstawowego modelu GAN. Przegląd obejmował takie modele jak:

- Conditional GAN (cGAN) - cGAN umożliwia generowanie obrazów warunkowych, czyli sterowanych dodatkowymi informacjami, w naszym przypadku typem przedmiotu, jego barwą oraz informacją czy tekstura dotyczy bloku czy itemu. Generator i dyskryminator otrzymują wówczas nie tylko wektor losowy, lecz także wektor warunkowy, co pozwala kontrolować cechy wygenerowanego obrazu[4].
- VQ-GAN - łączy zalety autoenkoderów (VQ-VAE) i uczenia kontradyktoryjnego wykorzystywanego w klasycznym GAN. Rozwiązanie takie pozwala na generowanie obrazów o większej ostrości[5].
- Wasserstein GAN (wGAN) - wGAN wprowadza metrykę Wassersteina do oceny różnic między rozkładem danych rzeczywistych i generowanych. Rozwiązanie to ma za zadanie ustabilizować proces uczenia i ograniczyć problem zanikania gradientu, częsty w klasycznym GAN[6].

Pomimo wymienionych zalet, ostatecznie zdecydowaliśmy się wybrać inny model w naszym projekcie. Ze wstępnych badań wynikało, że proces uczenia sieci GAN jest zauważalnie bardziej czasochłonny od sieci VQ-VAE oraz powstałe obrazy mimo że wykazywały dobrą ostrość, tak ich tekstura nie była odtwarzana na satysfakcyjnym poziomie. Ponadto sam proces uczenia sieci GAN pomimo zastosowanych wyżej metod okazał się mało stabilny, co prowadziło sporadycznie do zjawiska określonego jako "**mode collapse**", czyli sytuacji gdy model ogranicza różnorodność generowanych obrazów. Powody tego zjawiska mogą być różne, co ograniczyło skuteczną identyfikację źródła problemu.

2.2. Model dyfuzyjny

Model dyfuzyjny to generatywne podejście probabilistyczne, w którym proces uczenia polega na odwzorowaniu sekwencji kroków **odszumiania** danych. W fazie treningu do obrazów stopniowo dodawany jest szum gaussowski aż do całkowitego zniszczenia struktury, natomiast model uczy się odwrotnej transformacji, która w kolejnych krokach pozwala odtwarzać dane od czystego szumu do realistycznego obrazu.

Formalnie proces dyfuzji definiuje się jako:

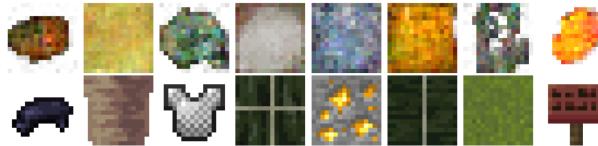
$$q(x_t | x_{t-1}) = \mathcal{N}(x_t; \sqrt{1 - \beta_t} x_{t-1}, \beta_t I),$$

gdzie β_t oznacza poziom szumu dodawanego w kroku t .

Odwrotny proces, aproksymowany przez sieć neuronową p_θ , ma postać:

$$p_\theta(x_{t-1} | x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)).$$

W naszych testach odrzuciliśmy to podejście z powodu **zbyt dużej liczby klas** w zbiorze danych, co znaczająco zwiększało złożoność uczenia i czas treningu.



Rysunek 1: Odwzorowanie dyfuzyjny

2.3. Variational Autoencoder (VAE)

Variational Autoencoder (VAE) stanowi rozwinięcie klasycznego autoenkodera. Zamiast kompresować dane wejściowe do deterministycznego wektora cech, enkoder w VAE uczy się parametrów rozkładu probabilistycznego (najczęściej wielowymiarowego rozkładu normalnego). Dla każdej próbki wejściowej enkoder zwraca wektor średnich μ oraz odchyлеń standardowych σ , które opisują rozkład przestrzeni ukrytej:

$$q_\phi(z|x) = \mathcal{N}(z; \mu(x), \sigma^2(x)I)$$

Następnie, zamiast przekazywać deterministyczny wektor, stosuje się *reparametrization trick*, aby umożliwić propagację gradientu przez próbkę:

$$z = \mu(x) + \sigma(x) \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

Dekoder rekonstruuje dane z wylosowanej próbki z . Całość jest trenowana poprzez minimalizację funkcji straty składającej się z dwóch składników:

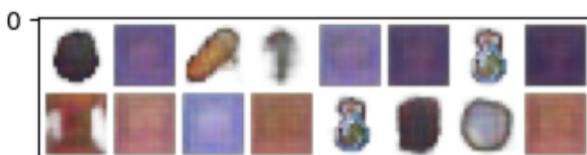
$$\mathcal{L}(x) = E_{q_\phi(z|x)} \left[-\log p_\theta(x|z) \right] + KL(q_\phi(z|x) \parallel p(z))$$

Pierwszy człon odpowiada za dokładność rekonstrukcji danych, natomiast drugi (dywergencja Kullbacka–Leiblera) regularizuje przestrzeń latentną, wymuszając podobieństwo rozkładu $q_\phi(z|x)$ do przyjętego priora $p(z)$, najczęściej standardowego rozkładu normalnego $\mathcal{N}(0, I)$.

VAE umożliwia generowanie nowych danych poprzez próbkowanie ze zdefiniowanego rozkładu latentnego. W naszym przypadku, gdzie analizowaliśmy obrazy z naszego zbioru podejście to się nie sprawdziło. Minimalizacja funkcji rekonstrukcji prowadzi do **rozmytych obrazów**, gdy model uśrednia wiele możliwych wariantów. W konsekwencji generowane obrazy traciły ostre krawędzie i charakterystyczne detale, które są kluczowe dla pixel artów. Z tego względu VAE okazał się mało efektywny w naszym zadaniu i szybko przeszliśmy do analizy VQ-VAE, które lepiej zachowuje dyskretne, wyraziste struktury obrazu.



Rysunek 2: Oryginalne zdjęcie



Rysunek 3: Rekonstruowane przez vae

2.4. Conditional VQ-VAE

Vector Quantized Variational Autoencoder (VQ-VAE) stanowi rozwinięcie klasycznego VAE, w którym przestrzeń latentna nie jest modelowana jako rozkład ciągły, lecz **dyskretyzowana** poprzez mapowanie wektorów latentnych do elementów ze skończonego *codebooku*. Dzięki temu unika się problemu rozmytych rekonstrukcji, a generowane obrazy zachowują wyraźniejsze krawędzie i ostrą strukturę.

Proces kwantyzacji polega na zastąpieniu wektora latentnego $z_e(x)$ najbliższym wektorem kodu e_k z codebooku $E = \{e_1, e_2, \dots, e_K\}$:

$$z_q(x) = e_k \quad \text{gdzie} \quad k = \arg \min_j \|z_e(x) - e_j\|_2$$

Funkcja straty w VQ-VAE składa się z trzech elementów:

$$\mathcal{L} = \underbrace{\|x - \hat{x}\|^2}_{\text{reconstruction loss}} + \underbrace{\|\text{sg}[z_e(x)] - e\|_2^2}_{\text{codebook loss}} + \beta \underbrace{\|z_e(x) - \text{sg}[e]\|_2^2}_{\text{commitment loss}}$$

gdzie $\text{sg}[\cdot]$ oznacza operator *stop-gradient*, a β jest hiperparametrem równoważącym wpływ terminu commitment.

W naszej modyfikacji wprowadziliśmy **informacje warunkowe** opisujące dodatkowe cechy przedmiotów, takie jak *kolor*, *czy_blok* oraz *typ przedmiotu*. Metadane te zostały dołączone do wektora latentnego, co umożliwiło dekoderowi generowanie obrazów bardziej spójnych i zgodnych z kontekstem, a także ułatwiło kontrolowanie procesu generacji w późniejszym zastosowaniu w transformerze.



Rysunek 4: Oryginalne zdjęcie



Rysunek 5: Odtworzone przez sieć

2.5. Transformer + Conditional VQ-VAE

Po wybraniu reprezentacji Conditional VQ-VAE zastosowaliśmy transformera, który operuje na zakodowanych wektorach. Model ten wykorzystuje sinusoidalne osadzenia pozytyjne oraz dodatkowe informacje warunkujące (*blok, typ, kolor*), które są wprowadzane do sieci poprzez mechanizm uwagi. Każde z warunkowań jest rzutowane na przestrzeń latentną i integrowane z sekwencją wejściową poprzez warstwę `MultiheadAttention`, a następnie łączone wagowo:

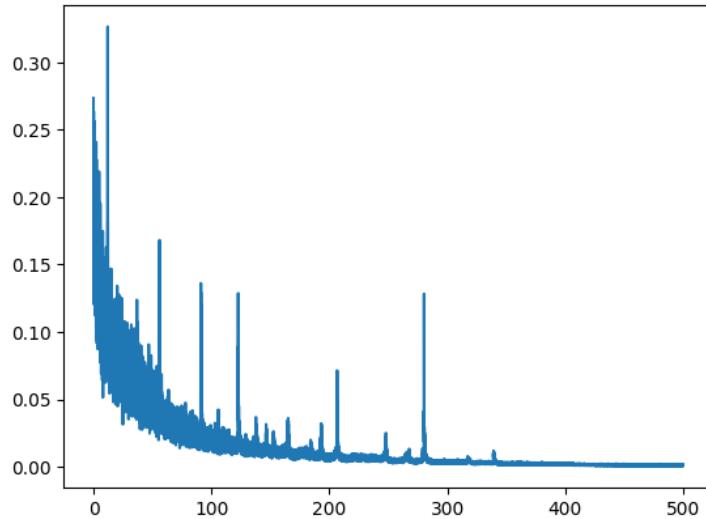
$$E = X + w_b \cdot \text{Attn}(X, B) + w_t \cdot \text{Attn}(X, T) + w_c \cdot \text{Attn}(X, C),$$

gdzie X to osadzenia wejściowe, B, T, C odpowiadają warunkom (*blok, typ, kolor*), a w_b, w_t, w_c to parametry uczone.

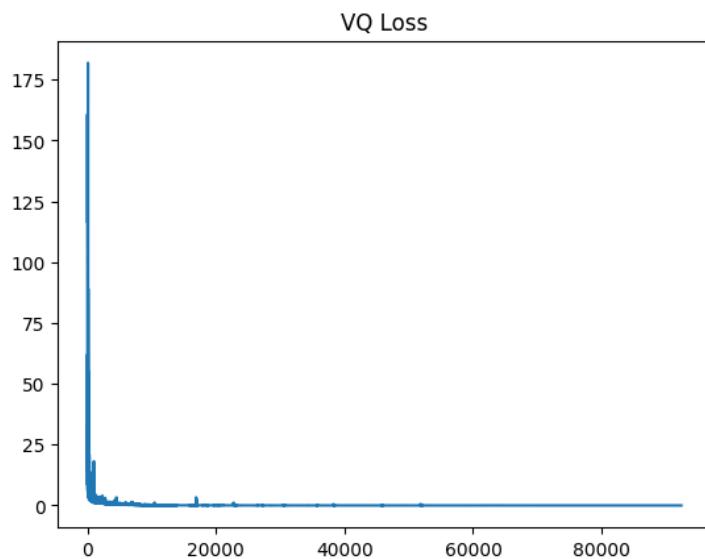
Tak przygotowana sekwencja trafia do enkodera zbudowanego z kilku bloków transformera (warstwy uwagi i sieci feed-forward), a na końcu do warstwy liniowej przewidującej kolejne tokeny.

3. Wyniki i analiza

3.1. Wyniki Conditional VQ-VAE



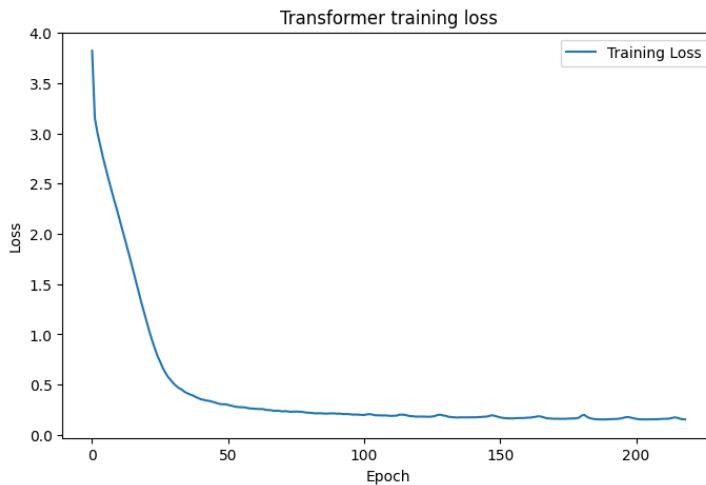
Rysunek 6: Wykres zależności wartości funkcji straty modelu od numeru epoki



Rysunek 7: Wartość VQ loss dla kolejnych batch'ów (około 190 na epokę)

Na powyższych wykresach zauważać możemy wyniki treningu najlepszego z rozpatrywanych modeli (cVQ-VAE). Rysunek 6 przedstawia jak dobrze model rekonstruuje dane wejściowe w kolejnych epokach, zaś strata VQ loss pokazana na rysunku 7 opisuje czy codebook naszego autoenkodera jest stabilny i dobrze wykorzystany. Szybki spadek wartości VQ loss świadczy o dobrej pracy enkodera w trenowanym modelu.

3.2. Wyniki Transformer + Conditional VQ-VAE



Rysunek 8: Wykres zależności wartości funkcji straty transformera od numeru epoki

Powyższy rysunek przedstawia wartość funkcji straty dla trenowanego po modelu transformera. Podobnie jak w przypadku VQ-VAE, transformer szybko zmniejszał swoją funkcję straty, czyli jakość generowanych obrazów znacząco się poprawiała.

4. Dyskusja

4.1. Napotkane problemy i ograniczenia

W trakcie tworzenia naszego rozwiązania pojawiły się następujące trudności:

- **Ograniczona ilość danych** – gra *Minecraft* posiada stosunkowo mały zbiór danych, dlatego rozszerzyliśmy dane o grafiki pochodzące z modów.
- **Ograniczona moc obliczeniowa** – przez większość czasu korzystaliśmy z platformy Google Colab, gdzie dostęp do GPU/CPU jest ograniczony czasowo, co spowalniało proces eksperymentów.
- **Niebalansowanie danych z datasetu** – znaczną część datasetu zajmują często pojawiające się (w różnych wariantach kolorystycznych) przedmioty takie jak: mikstury, narzędzia, bloki wełny, cementu, czy betonu. W efekcie model często upraszczał proces uczenia do najczęściej występującego przedmiotu, co było szczególnie uciążliwe przy próbie generowania nowych bloków. Model częściej zapamiętywał kolory oraz kształt niż teksturę bloku.

4.2. Dalszy rozwój

W dalszym etapie rozwoju zakładamy poprawę architektury sieci w celu poprawienia jej możliwości generowania nowych grafik oraz usunięcia błędów które powodują generowania niewyraźnych grafik.



4.3. Podsumowanie

Najważniejsze wnioski z realizacji projektu:

- Spośród rozważanych modeli do reprezentacji stanu i generacji obrazów odrzuciliśmy podejścia oparte na modelach dyfuzyjnych oraz GAN, głównie z powodu ograniczonej ilości danych.
- Ostatecznie wybraliśmy model **VQ-VAE**, wzbogacony o dodatkowe warunki (informacje o kolorze, typie oraz atrybutie *czy_blok*), co pozwoliło uzyskać lepszą kontrolę nad procesem generacji.

Literatura

- [1] A. Van Den Oord, O. Vinyals, *et al.*, "Neural discrete representation learning," Advances in Neural Information Processing Systems, vol. 30, 2017.
- [2] A. Saravanan, M. Guzdial, "Pixel VQ-VAEs for improved pixel art representation," arXiv preprint arXiv:2203.12130, 2022.
- [3] I. J. Goodfellow, *et al.*, "Generative Adversarial Nets" arXiv:1406.2661, 2014
- [4] M. Mirza, S. Osindero, "Conditional Generative Adversarial Nets" arXiv:1411.1784, 2014
- [5] P. Esser, R. Rombach, B. Ommer, "Taming Transformers for High-Resolution Image Synthesis" arXiv:2012.09841, 2020
- [6] M. Arjovsky, S. Chintala, L. Bottou, "Wasserstein GAN" arXiv:1701.07875, 2017