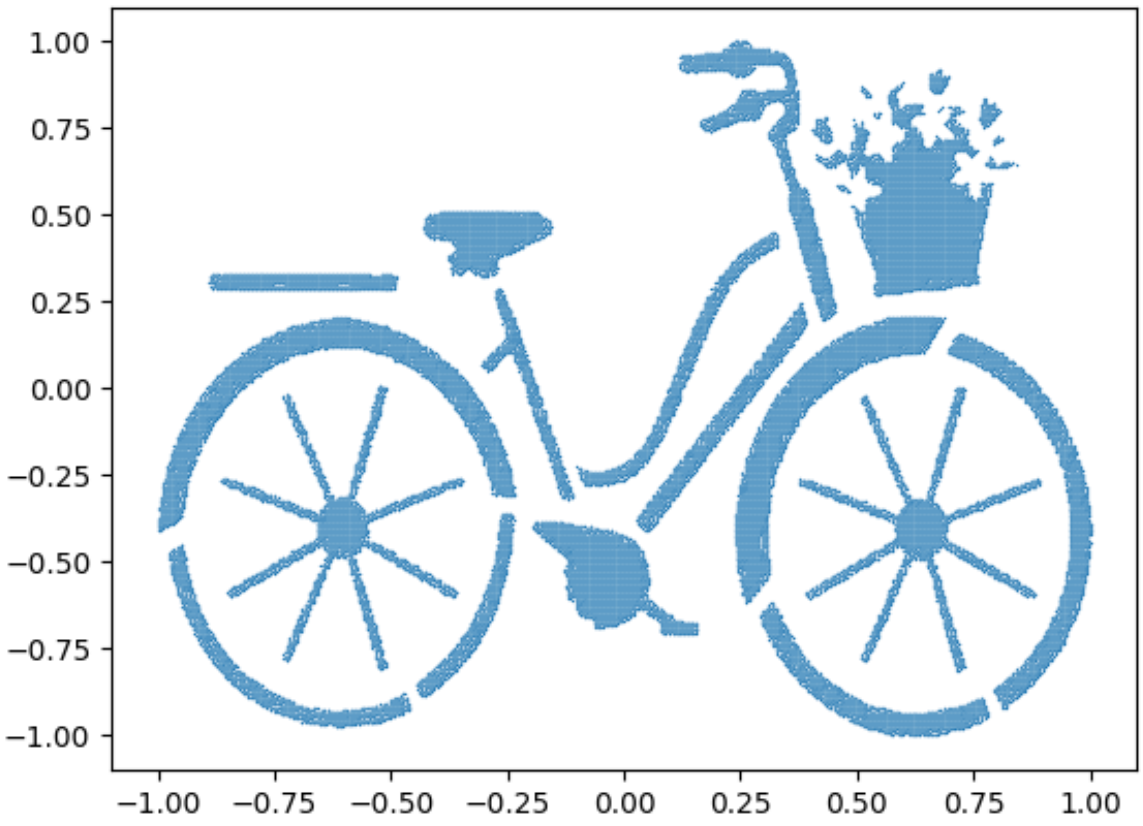
Metody Rozpoznawania Obrazów
Dominik Szot
Zadanie 05

Głównym tematem laboratorium było zaimplementowanie klasycznego generatywnego modelu dyfuzyjnego DDPM (Denoising Diffusion Probabilistic Models).

Zbiorem treningowym jest 50 tysięcy obserwacji, które łącznie układają się w obiekt o kształcie roweru.

1. Wczytanie zbioru danych

Pierwszym zadaniem było wyświetlenie zawartościu zbioru danych. Jak widać układają się w bardzo charakterystyczny kształ roweru.



2. Dyfuzja w przód

W tym punkcie będziemy startać się zaszumieć nasz początkowy obraz korzystając z równania, które opisuje proces dyfuzji w przód.

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

Uproszczony wzór ma postać

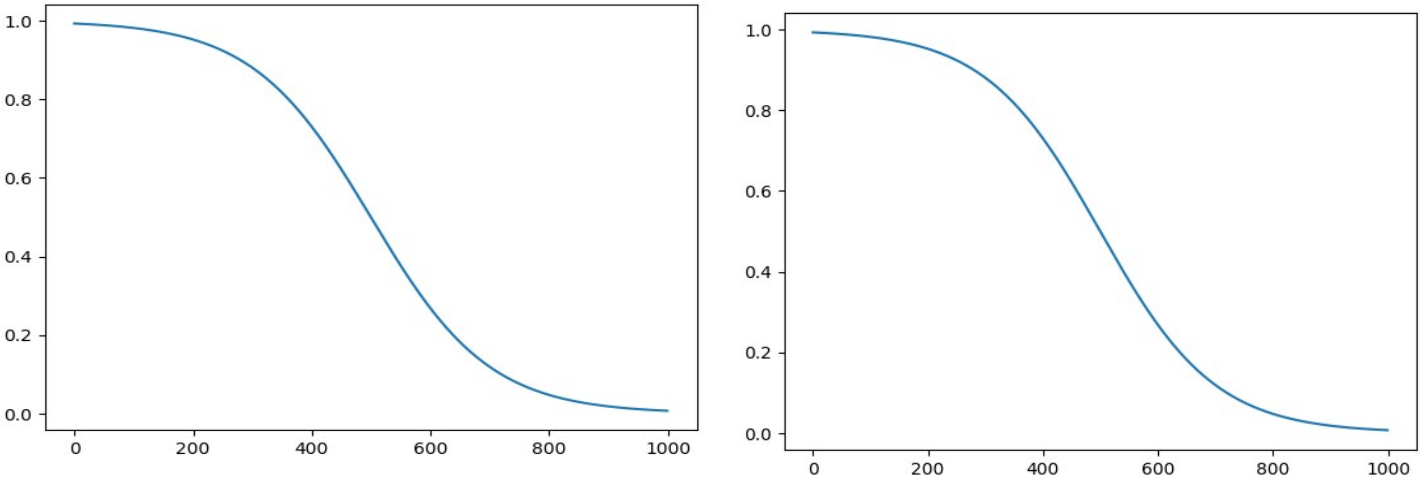
$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})$$

a prościej w rozumieniu:

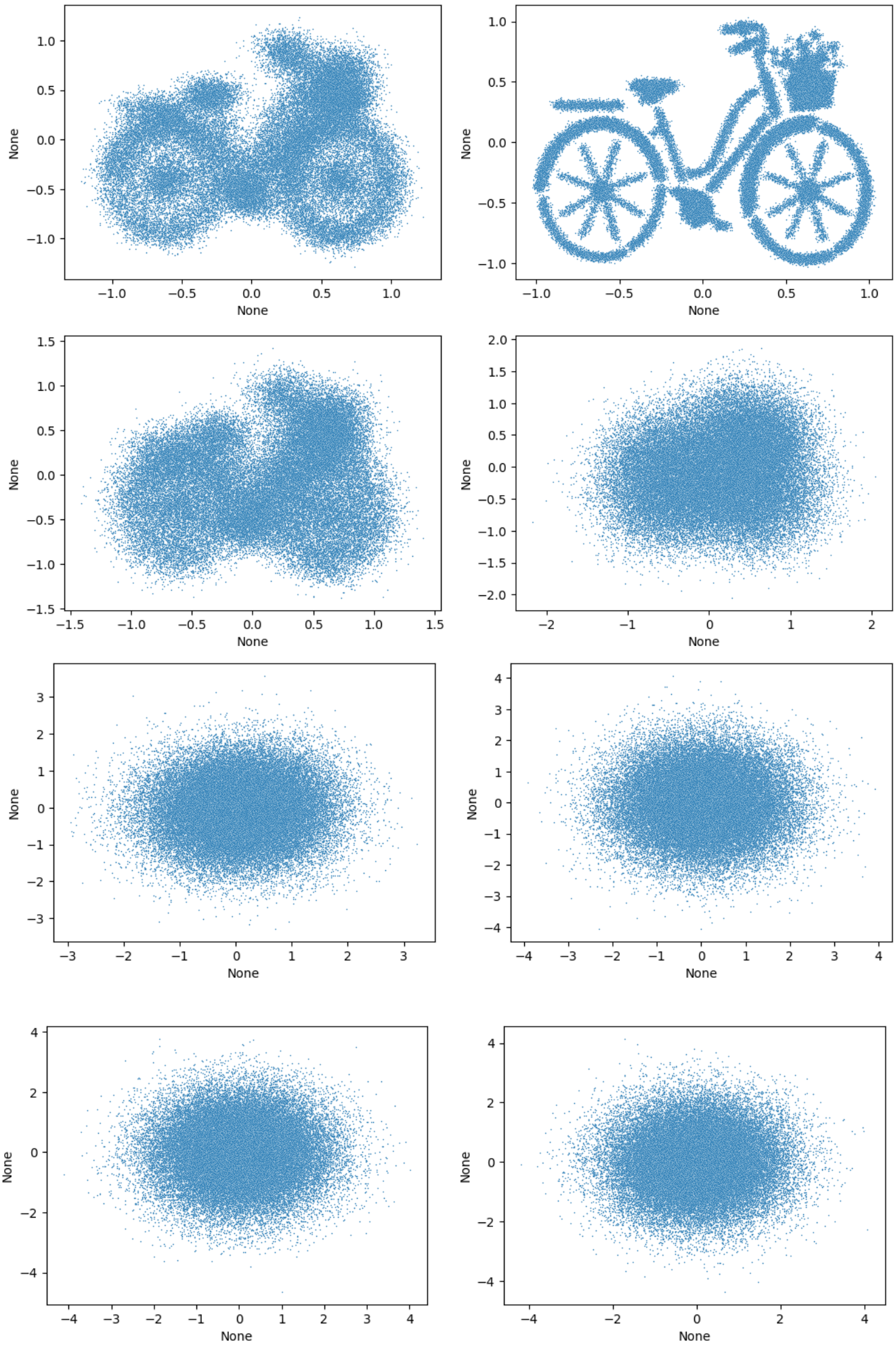
$$x_t = \sqrt{(a_t)} * x_0 + \sqrt{(1-a_t)} * epsilon$$

Funkcja której będziemy używać w procesie odszumiania będzie oparta na funkcji sigmoidalnej. W dlalszych krokach postaram się zobrazować różnicę w działaniu funkcji dyfuzji w przód w zależności od funkcji.

```
if sigmoid:
        self.alpha_prod = sigmoid(torch.linspace(1, -1, num_steps))
        alphas[0] = 1 - beta_start
        alphas[1:] = self.alpha_prod [1:] / self.alpha_prod [:-1]
        self.alphas = alphas
        self.betas = 1 - alphas
else:
        self.betas = torch.linspace(beta_start, beta_end, num_steps)
        self.alphas = 1 - self.betas
        self.alpha_prod = torch.cumprod(self.alphas, dim=0)
```



Obraz 1: Zmiana wartości sumy alfa z zależności od t oraz funkcji sigmoidalnej (prawa) lub liniowej (lewa)



Obraz 2: Wynik działania dyfuzji w przód dla 10, 100, 500, 1000 kroków

3. Definiowanie modelu predykcyjnego

W celu dodania informacji o krokach czasowych dodano kodowanie pozycyjne, obliczane na podstawie numeru kroku czasowego. Reszta jest odpowiednikiem architektury przedstawionej w treści zadania

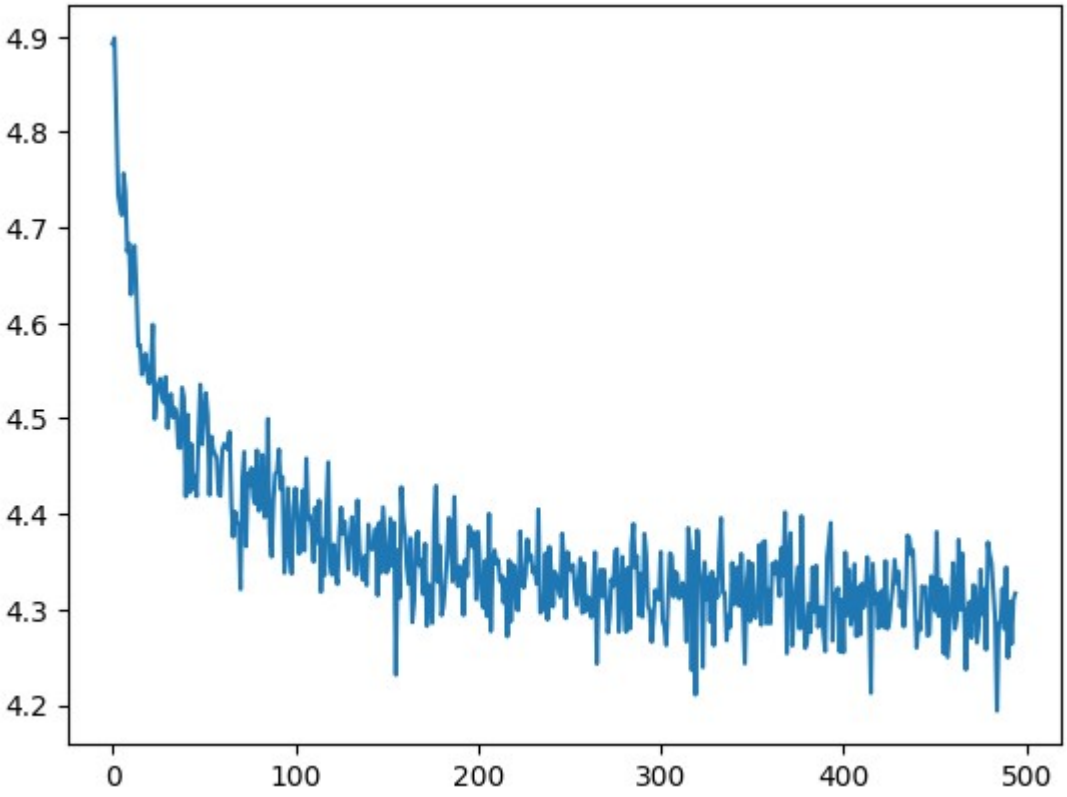
```
class PositionalEncoding(nn.Module):
        def __init__(self, dim, max_len=1000):
                super().__init__()
                self.register_buffer("pos_enc", self._init_pos_enc(dim, max_len))

        def _init_pos_enc(self, d_model, max_len):
                position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
                div_term = torch.exp(torch.arange(0, d_model, 2).float() * (-math.log(10000.0) / d_model))

                pos_enc = torch.zeros(max_len, d_model)
                pos_enc[:, 0::2] = torch.sin(position * div_term)
                pos_enc[:, 1::2] = torch.cos(position * div_term)

                return pos_enc

        def forward(self, t):
                t = torch.clamp(t, max=999)
                return self.pos_enc[t]

class BackwardsDiffusionModel(nn.Module):
        def __init__(self, in_dim=2, dim=128, num_steps=1000):
                super().__init__()

                self.pos_enc = PositionalEncoding(dim)

                self.linear1 = nn.Linear(in_dim, dim)
                self.linear2 = nn.Linear(dim, dim)
                self.linear3 = nn.Linear(dim, dim)
                self.linear4 = nn.Linear(dim, in_dim)

                self.relu = nn.ReLU()

        def forward(self, x: torch.Tensor, t: int):
                pos_enc = self.pos_enc(t)

                x = self.relu(self.linear1(x) + pos_enc)
                x = self.relu(self.linear2(x) + pos_enc)
                x = self.relu(self.linear3(x) + pos_enc)
                x = self.linear4(x)

        return x
```

```
model = BackwardsDiffusionModel(in_dim=2, dim=128, num_steps=1000)
x = torch.randn(4, 2)
t = torch.tensor(42)
output = model(x, t)
print("Wymiary wejścia:", x.shape)
print("Wymiary wyjścia:", output.shape)
```

Wymiary wejścia: torch.Size([4, 2])
Wymiary wyjścia: torch.Size([4, 2])

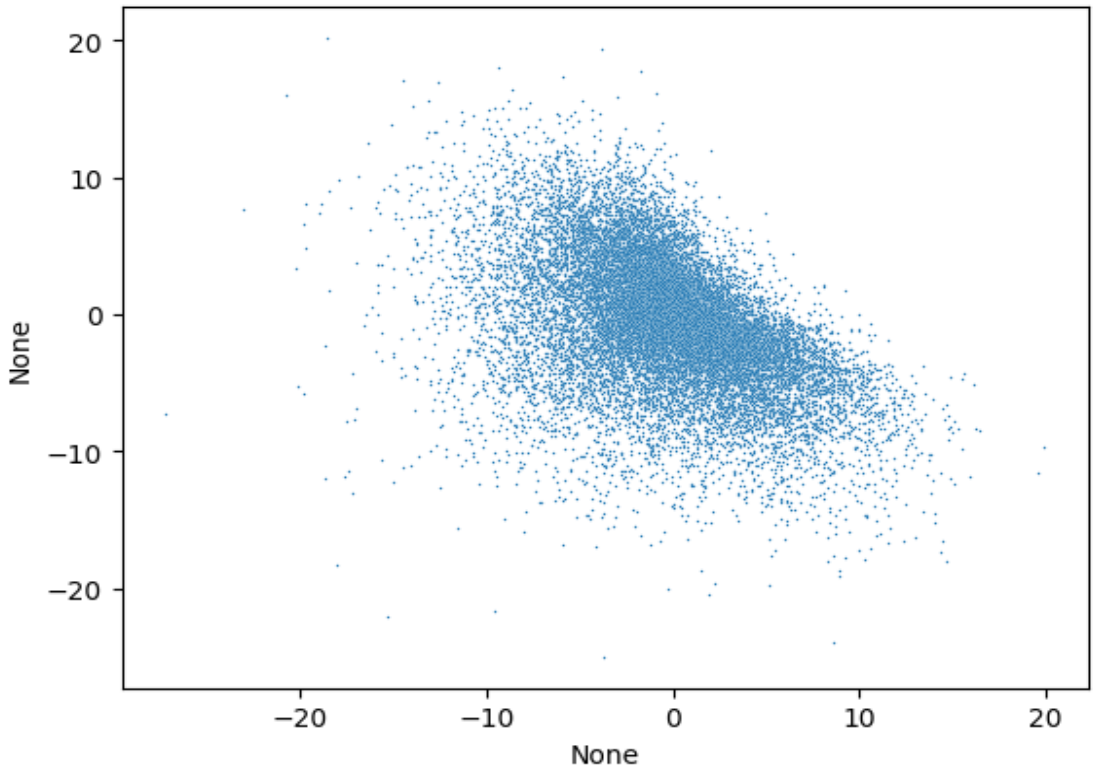
4. Trening modelu

Trening modelu był raczej standardowy, z ciekawszych rzeczy to zauważono fakt że model od pewnego momentu odmawiał poprawy swoich predykcji – generowania dokładniejszych obrazów roweru.

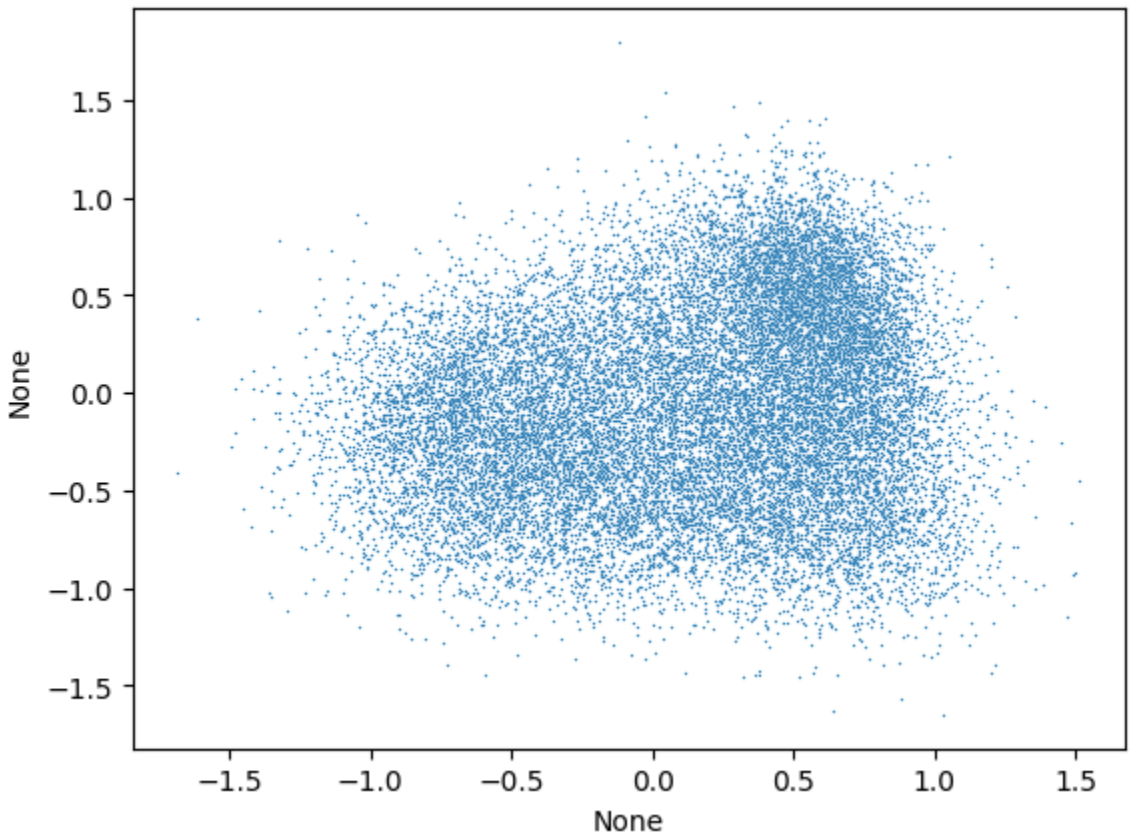


Obraz 3: Wykres funkcji kosztu

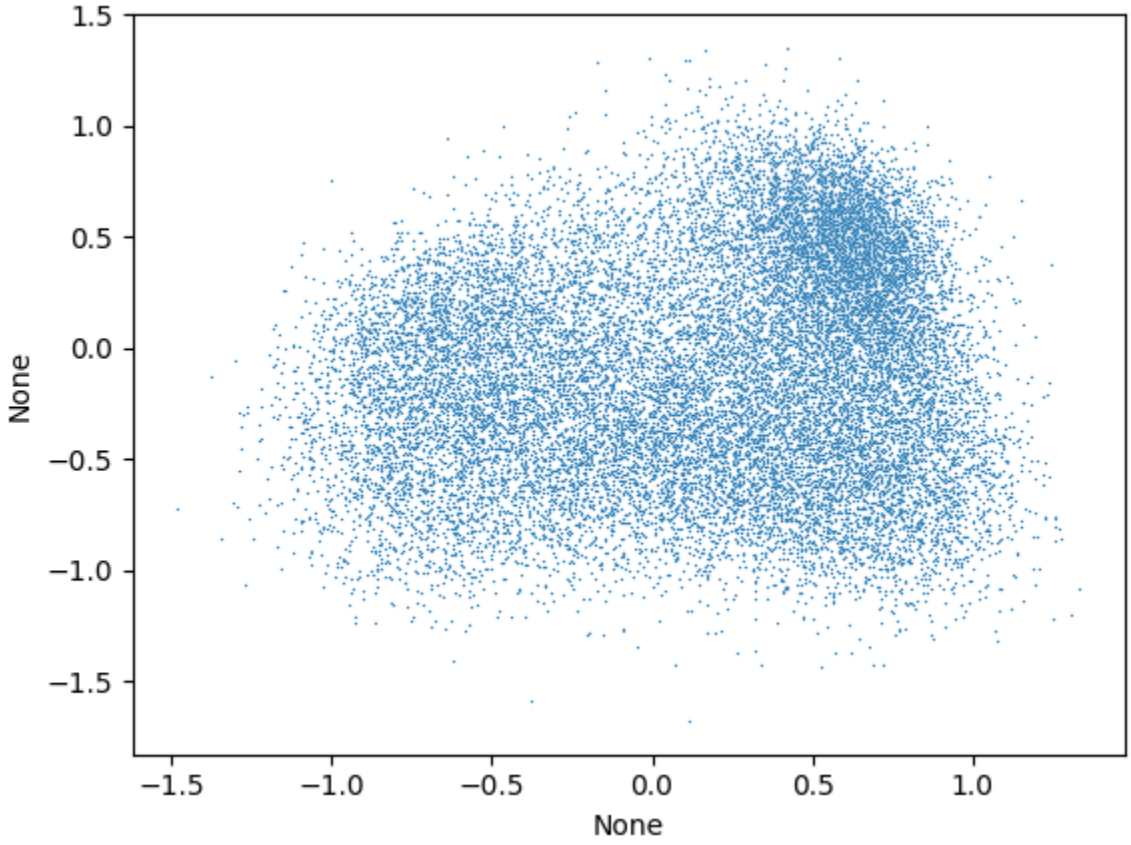
Jeśli zaś chodzi o wyniki generowania:



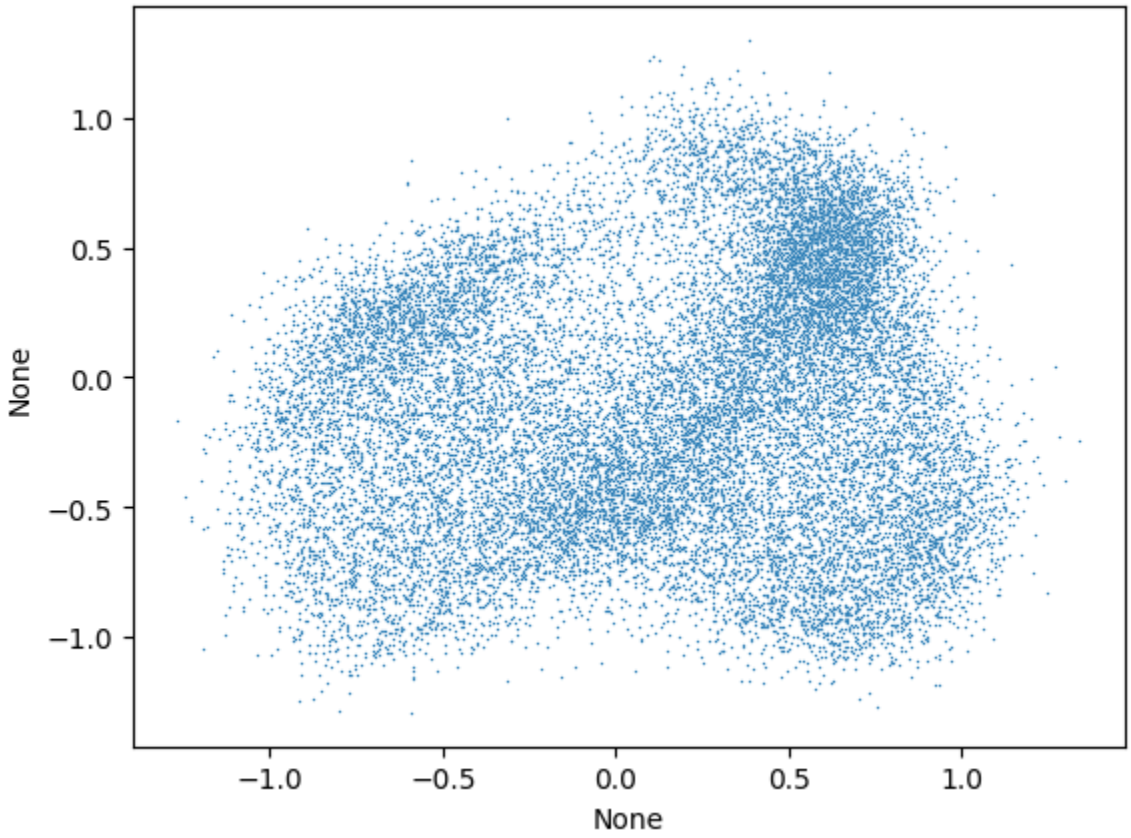
Obraz 4: Wynik dla modelu trenowanego przez [0] epok



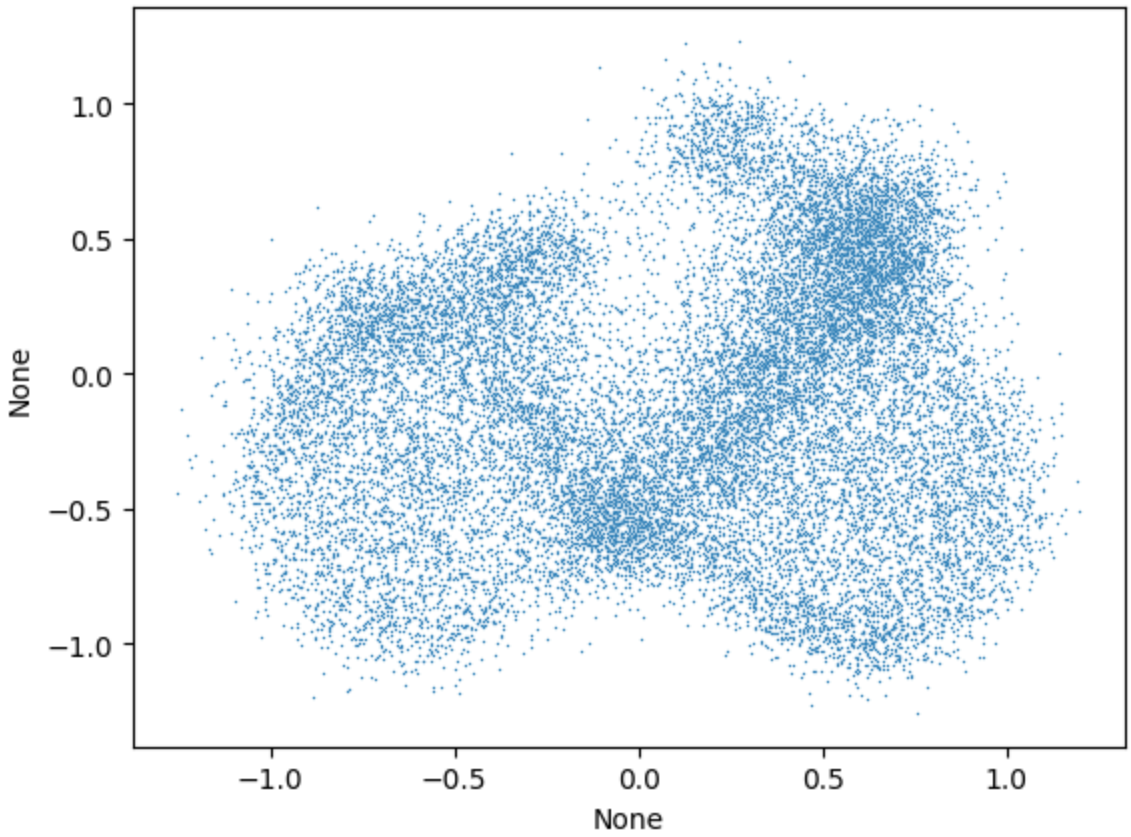
Obraz 5: Wynik dla modelu trenowanego przez [25] epok



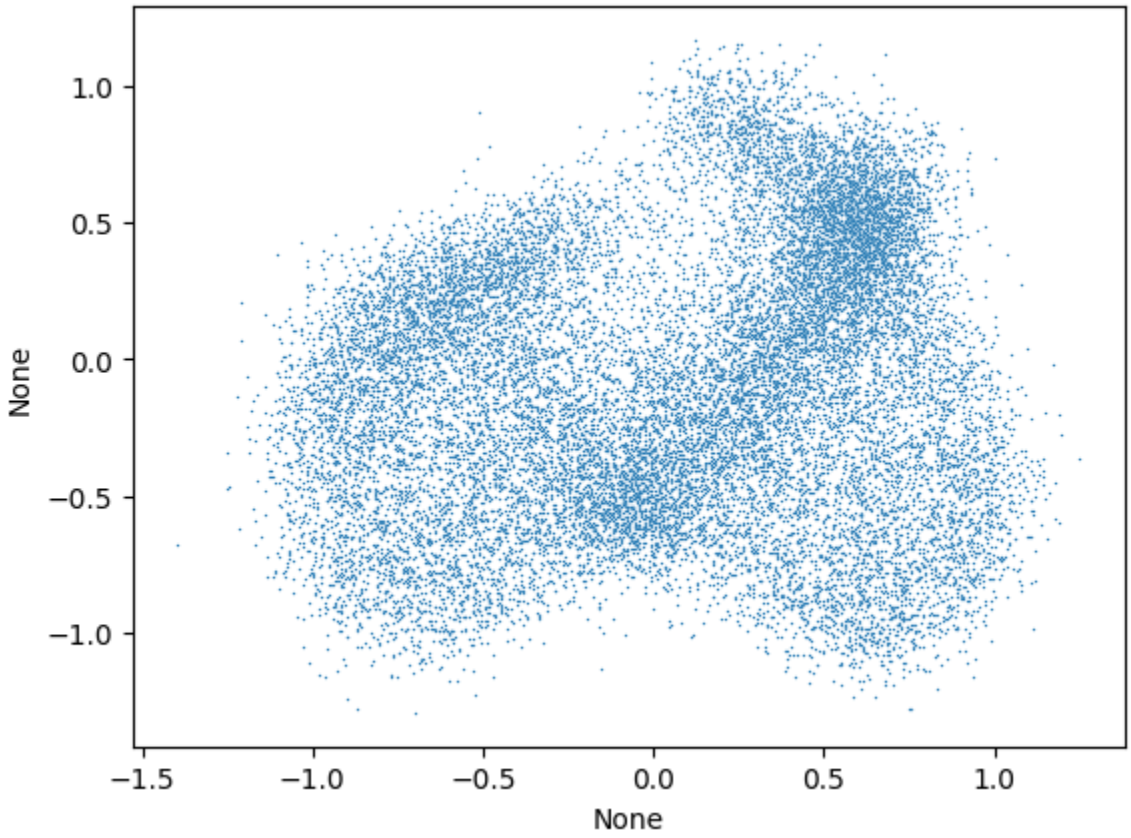
Obraz 6: Wynik dla modelu trenowanego przez [75] epok



Obraz 7: Wynik dla modelu trenowanego przez [125] epok



Obraz 8: Wynik dla modelu trenowanego przez [250] epok



Obraz 9: Wynik dla modelu trenowanego przez [450] epok