Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №2

По дисциплине: «Языковые процессы интеллектуальных систем»

Тема: «Автоэнкодеры»

Выполнил:

Студент 4 курса

Группы ИИ-23

Скварнюк Д. Н.

Проверила:

Андренко К. В.

Брест 2025

Цель: научиться применять автоэнкодеры для осуществления визуализации данных и их анализа

Общее задание

спользуя выборку по варианту, осуществить проецирование данных на плоскость первых двух и трех главных компонент с использованием нейросетевой модели автоэнкодера (с двумя и тремя нейронами в среднем слое);

ыполнить визуализацию полученных главных компонент с использованием средств библиотеки matplotlib, обозначая экземпляры разных классов с использованием разных цветовых маркеров;

еализовать метод t-SNE для визуализации данных (использовать также 2 и 3 компонента), построить соответствующую визуализацию;

рименить к данным метод PCA (2 и 3 компонента), реализованный в ЛР №1, сделать выводы;

формить отчет по выполненной работе, загрузить исходный код и отчет в соответствующий репозиторий на github.

Задание по вариантам

№	Выборка	Класс
	Optical recognition of handwritten digits	Последний признак (tra)

Код программы:

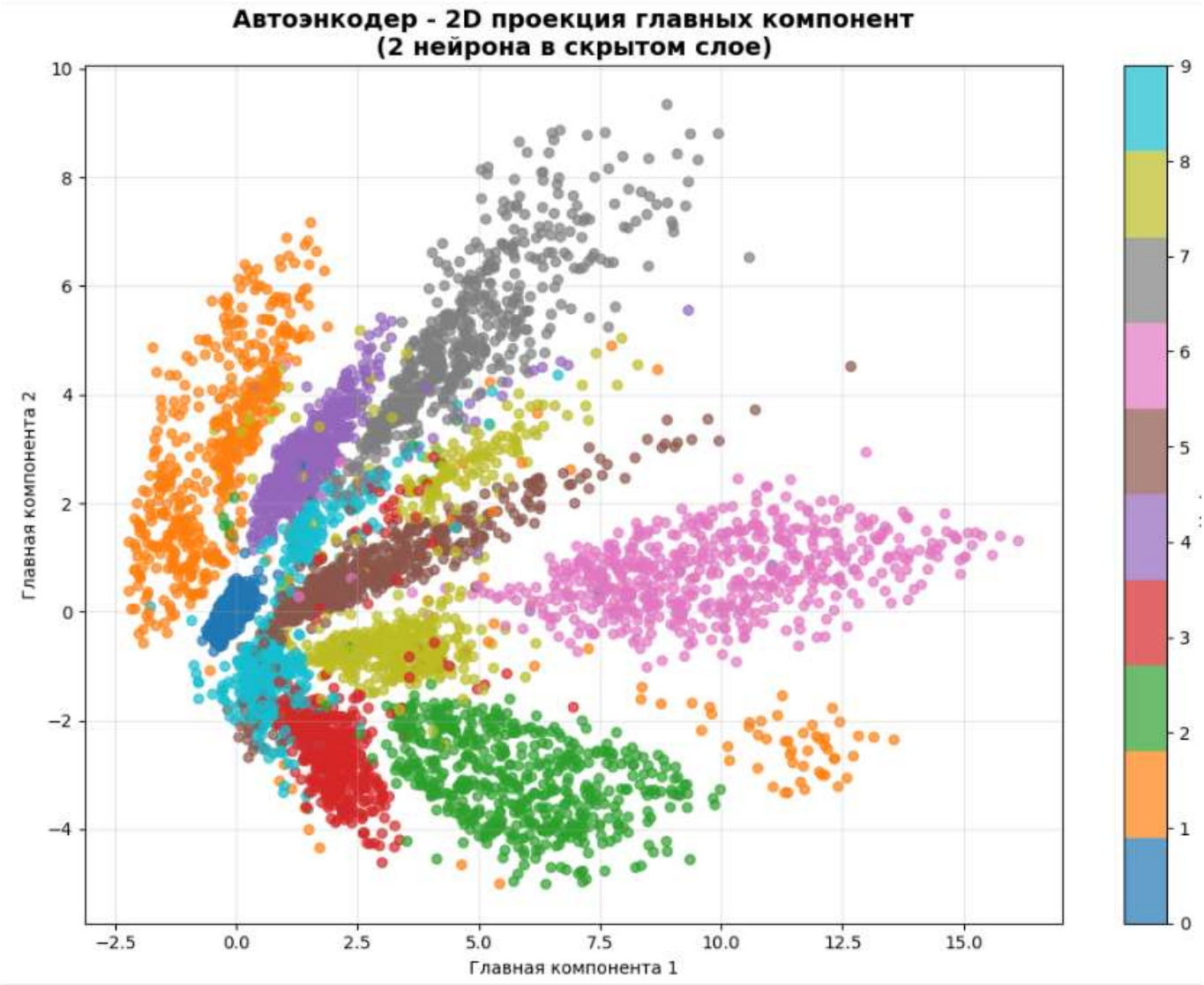
```
!pip install ucimlrepo

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
import seaborn as sns

plt.style.use('default')
sns.set_palette("tab10")


from ucimlrepo import fetch_ucirepo

optical_recognition_of_handwritten_digits = fetch_ucirepo(id=80)

X = optical_recognition_of_handwritten_digits.data.features
y = optical_recognition_of_handwritten_digits.data.targets

print(optical_recognition_of_handwritten_digits.metadata)
```

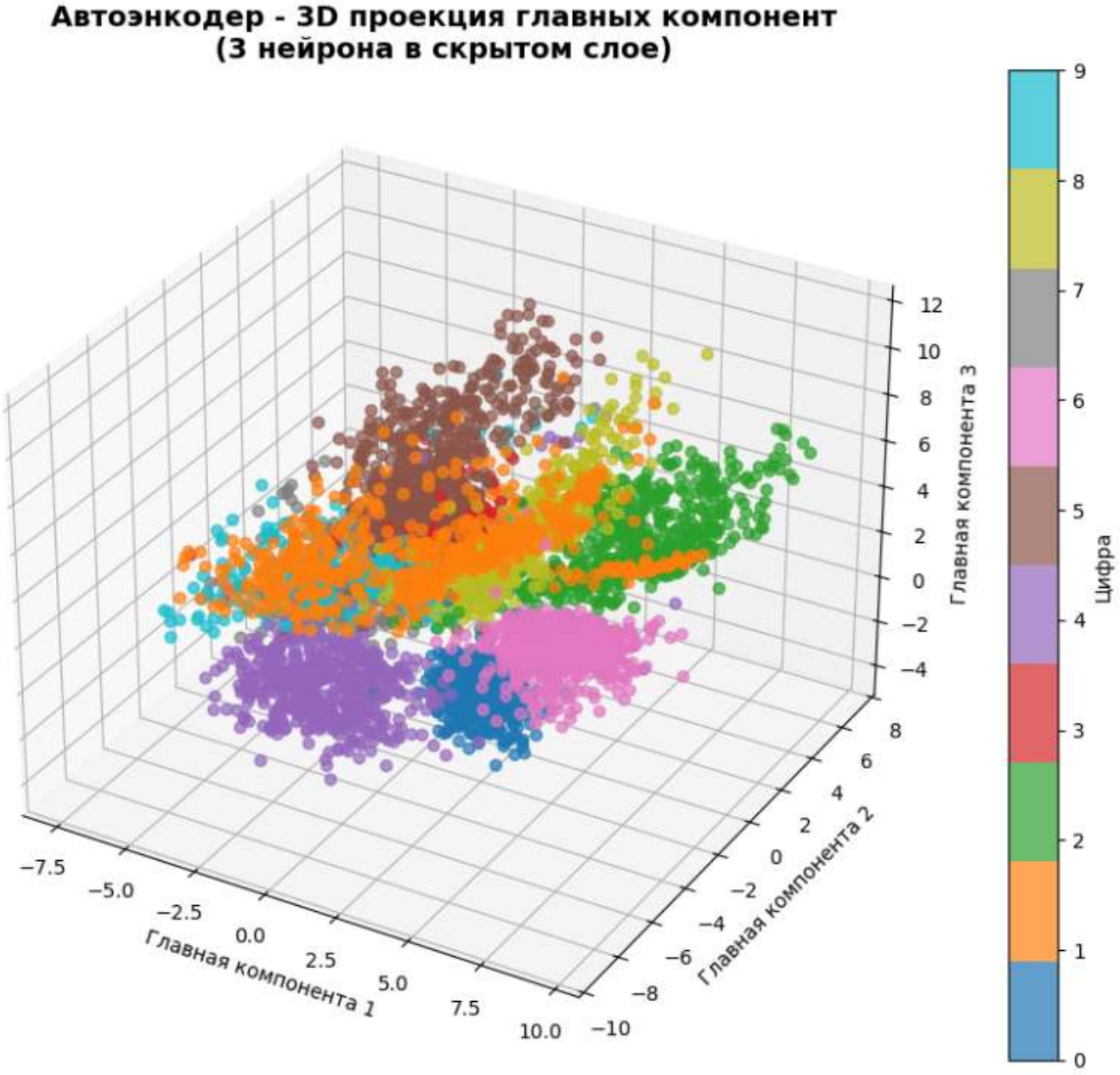
```
print(optical_recognition_of_handwritten_digits.variables)

X = X.values
y = y.values.ravel()

print(f"\nФорма данных: X {X.shape}, y {y.shape}")
print(f"Уникальные классы: {np.unique(y)}")
print(f"Диапазон значений признаков: [{X.min()}, {X.max()}]")

# Нормализация данных в диапазон [0, 1] для лучшей работы автоэнкодера
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
print(f"Диапазон после нормализации: [{X_scaled.min():.3f},
{X_scaled.max():.3f}]")


def create_efficient_autoencoder(encoding_dim, input_dim=64):
    """
    Создает эффективный автоэнкодер с заданной размерностью скрытого слоя
    """
    # Входной слой
    input_layer = Input(shape=(input_dim,))

    # Энкодер
    x = Dense(128, activation='relu')(input_layer)
    x = BatchNormalization()(x)
    x = Dense(64, activation='relu')(x)
    x = BatchNormalization()(x)
    x = Dense(32, activation='relu')(x)

    # Бутылочное горлышко - главные компоненты
    encoded = Dense(encoding_dim, activation='linear', name='bottleneck')(x)

    # Декодер
    x = Dense(32, activation='relu')(encoded)
    x = BatchNormalization()(x)
    x = Dense(64, activation='relu')(x)
    x = BatchNormalization()(x)
    x = Dense(128, activation='relu')(x)

    # Выходной слой
    decoded = Dense(input_dim, activation='sigmoid')(x)

    # Модели
    autoencoder = Model(input_layer, decoded)
    encoder = Model(input_layer, encoded)

    # Компиляция с оптимизированными параметрами
    autoencoder.compile(
        optimizer=Adam(learning_rate=0.001),
```

```
        loss='mse',
        metrics=['mae']
    )

    return autoencoder, encoder



# Создание и обучение автоэнкодера с 2 нейронами

autoencoder_2d, encoder_2d = create_efficient_autoencoder(encoding_dim=2)

# Callbacks для улучшения обучения
callbacks = [
    EarlyStopping(monitor='val_loss', patience=15, restore_best_weights=True),
    ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=10,
min_lr=0.0001)
]

print("Архитектура автоэнкодера:")
autoencoder_2d.summary()

# Обучение модели
history_2d = autoencoder_2d.fit(
    X_scaled, X_scaled,
    epochs=100,
    batch_size=128,
    validation_split=0.2,
    shuffle=True,
    callbacks=callbacks,
    verbose=1
)

# Получение закодированных представлений
X_encoded_2d = encoder_2d.predict(X_scaled)
print(f"Закодированные данные (2D): {X_encoded_2d.shape}")


# Создание и обучение автоэнкодера с 3 нейронами
print("\n" + "=" * 60)
print("ОБУЧЕНИЕ АВТОЭНКОДЕРА С 3 НЕЙРОНАМИ")
print("=" * 60)

autoencoder_3d, encoder_3d = create_efficient_autoencoder(encoding_dim=3)

history_3d = autoencoder_3d.fit(
    X_scaled, X_scaled,
    epochs=100,
    batch_size=128,
    validation_split=0.2,
    shuffle=True,
```

```
    callbacks=callbacks,
    verbose=1
)

# Получение закодированных представлений
X_encoded_3d = encoder_3d.predict(X_scaled)
print(f"Закодированные данные (3D): {X_encoded_3d.shape}")



# Визуализация главных компонент автоэнкодера
print("\n" + "=" * 60)
print("ВИЗУАЛИЗАЦИЯ ГЛАВНЫХ КОМПОНЕНТ АВТОЭНКОДЕРА")
print("=" * 60)

fig = plt.figure(figsize=(20, 8))

# 1. 2D проекция автоэнкодера
plt.subplot(1, 2, 1)
scatter = plt.scatter(X_encoded_2d[:, 0], X_encoded_2d[:, 1],
                      c=y, cmap='tab10', alpha=0.7, s=30)
plt.colorbar(scatter, label='Цифра')
plt.title('Автоэнкодер - 2D проекция главных компонент\n(2 нейрона в скрытом
слое)',
          fontsize=14, fontweight='bold')
plt.xlabel('Главная компонента 1')
plt.ylabel('Главная компонента 2')
plt.grid(True, alpha=0.3)

# 2. 3D проекция автоэнкодера
ax = plt.subplot(1, 2, 2, projection='3d')
scatter_3d = ax.scatter(X_encoded_3d[:, 0], X_encoded_3d[:, 1], X_encoded_3d[:,
2],
                        c=y, cmap='tab10', alpha=0.7, s=30)
plt.colorbar(scatter_3d, label='Цифра')
ax.set_title('Автоэнкодер - 3D проекция главных компонент\n(3 нейрона в скрытом
слое)',
             fontsize=14, fontweight='bold')
ax.set_xlabel('Главная компонента 1')
ax.set_ylabel('Главная компонента 2')
ax.set_zlabel('Главная компонента 3')

plt.tight_layout()
plt.show()

# Графики обучения
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

# Потери 2D автоэнкодера
ax1.plot(history_2d.history['loss'], label='Обучающая', linewidth=2)
ax1.plot(history_2d.history['val_loss'], label='Валидационная', linewidth=2)
```

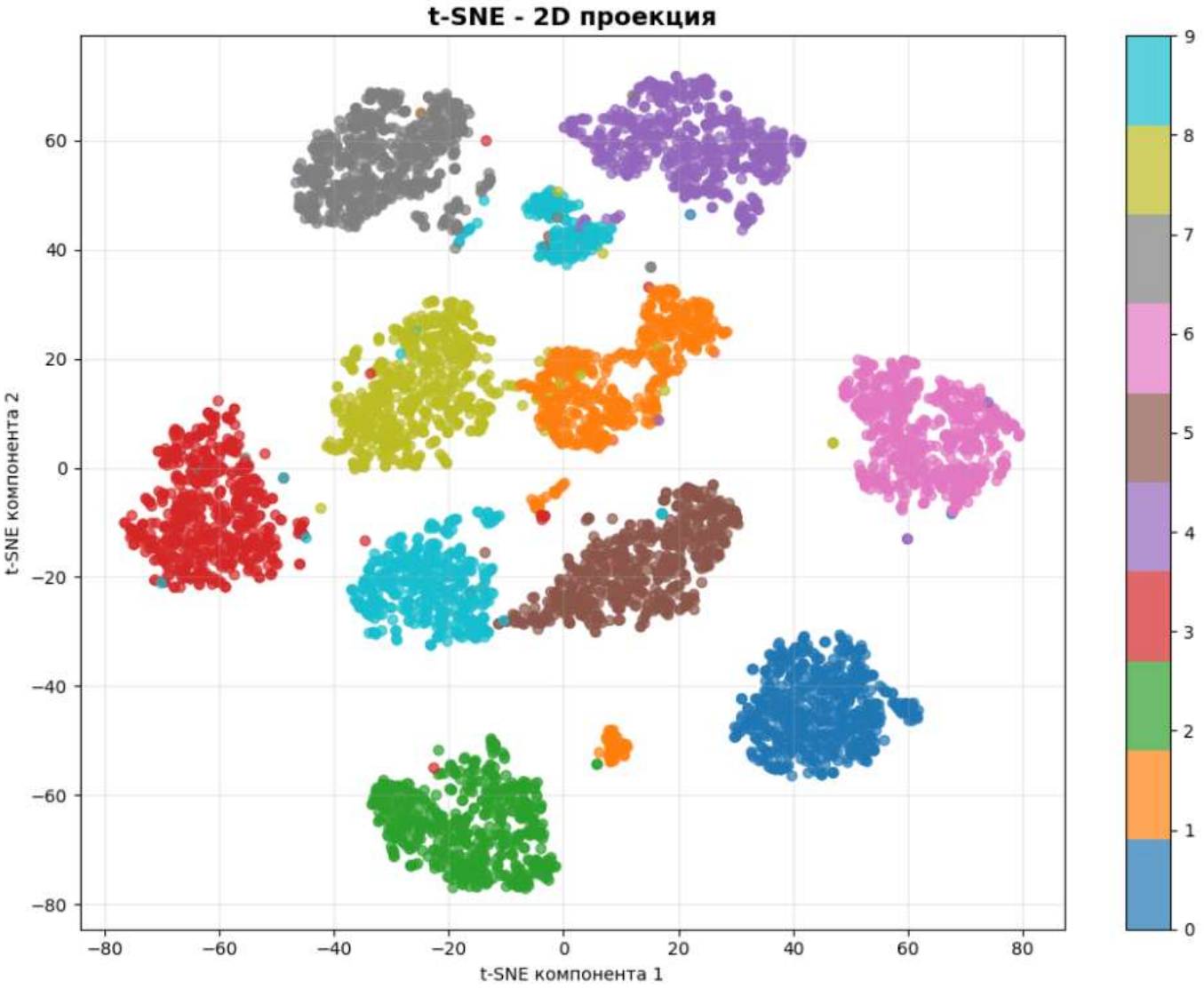
```
ax1.set_title('Потери автоэнкодера (2D)', fontweight='bold')
ax1.set_xlabel('Эпоха')
ax1.set_ylabel('MSE Loss')
ax1.legend()
ax1.grid(True, alpha=0.3)

# Потери 3D автоэнкодера
ax2.plot(history_3d.history['loss'], label='Обучающая', linewidth=2)
ax2.plot(history_3d.history['val_loss'], label='Валидационная', linewidth=2)
ax2.set_title('Потери автоэнкодера (3D)', fontweight='bold')
ax2.set_xlabel('Эпоха')
ax2.set_ylabel('MSE Loss')
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

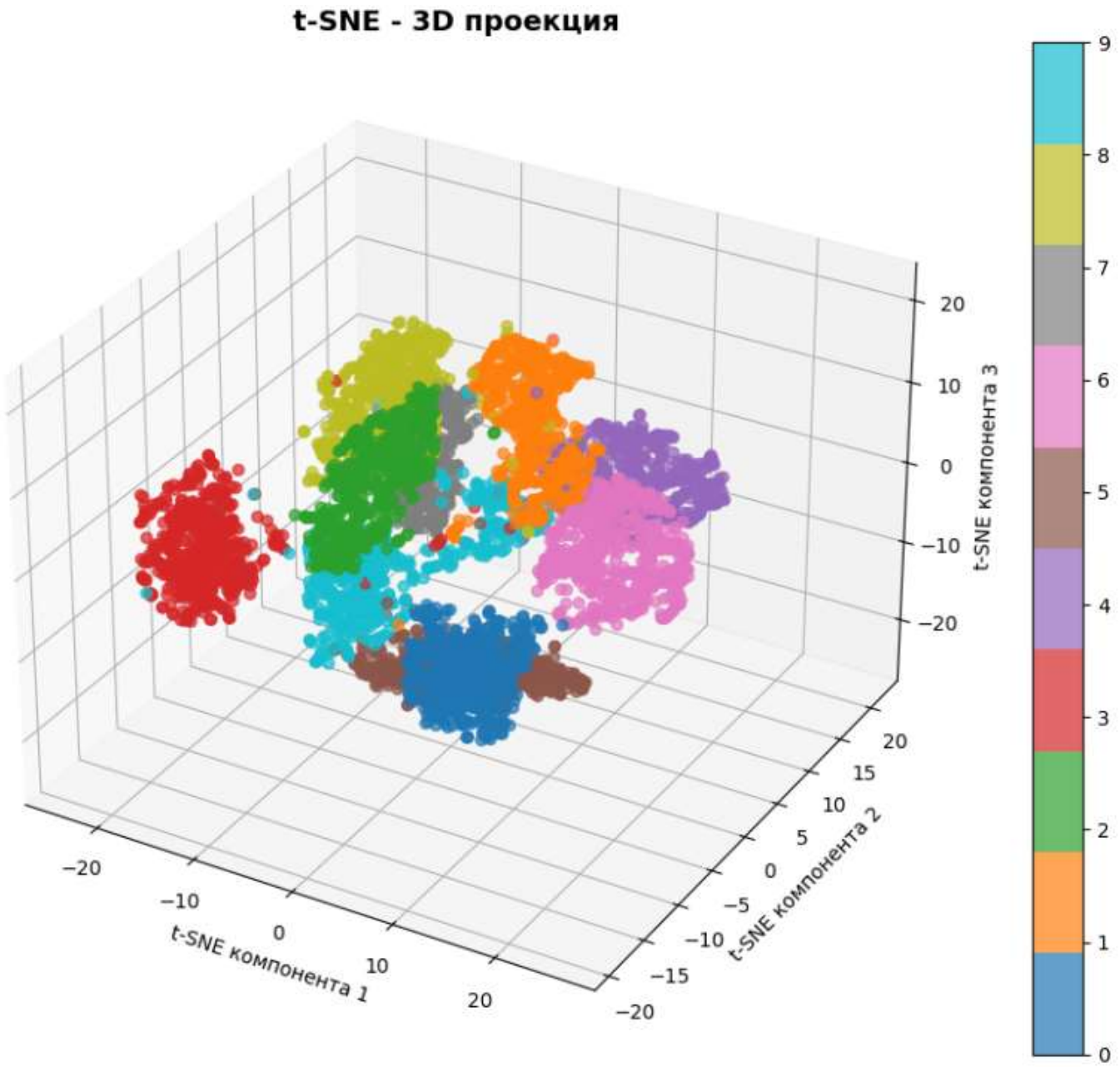





```
# Реализация t-SNE для визуализации данных
print("\n" + "=" * 60)
print("ВИЗУАЛИЗАЦИЯ МЕТОДОМ t-SNE")
print("=" * 60)

# t-SNE с 2 компонентами (исправленная версия без предупреждений)
print("Выполнение t-SNE с 2 компонентами...")
tsne_2d = TSNE(n_components=2, random_state=42, perplexity=30,
               max_iter=1000, learning_rate=200)  # Заменили n_iter на max_iter
X_tsne_2d = tsne_2d.fit_transform(X_scaled)

# t-SNE с 3 компонентами (исправленная версия без предупреждений)
print("Выполнение t-SNE с 3 компонентами...")
tsne_3d = TSNE(n_components=3, random_state=42, perplexity=30,
               max_iter=1000, learning_rate=200)  # Заменили n_iter на max_iter
X_tsne_3d = tsne_3d.fit_transform(X_scaled)
```

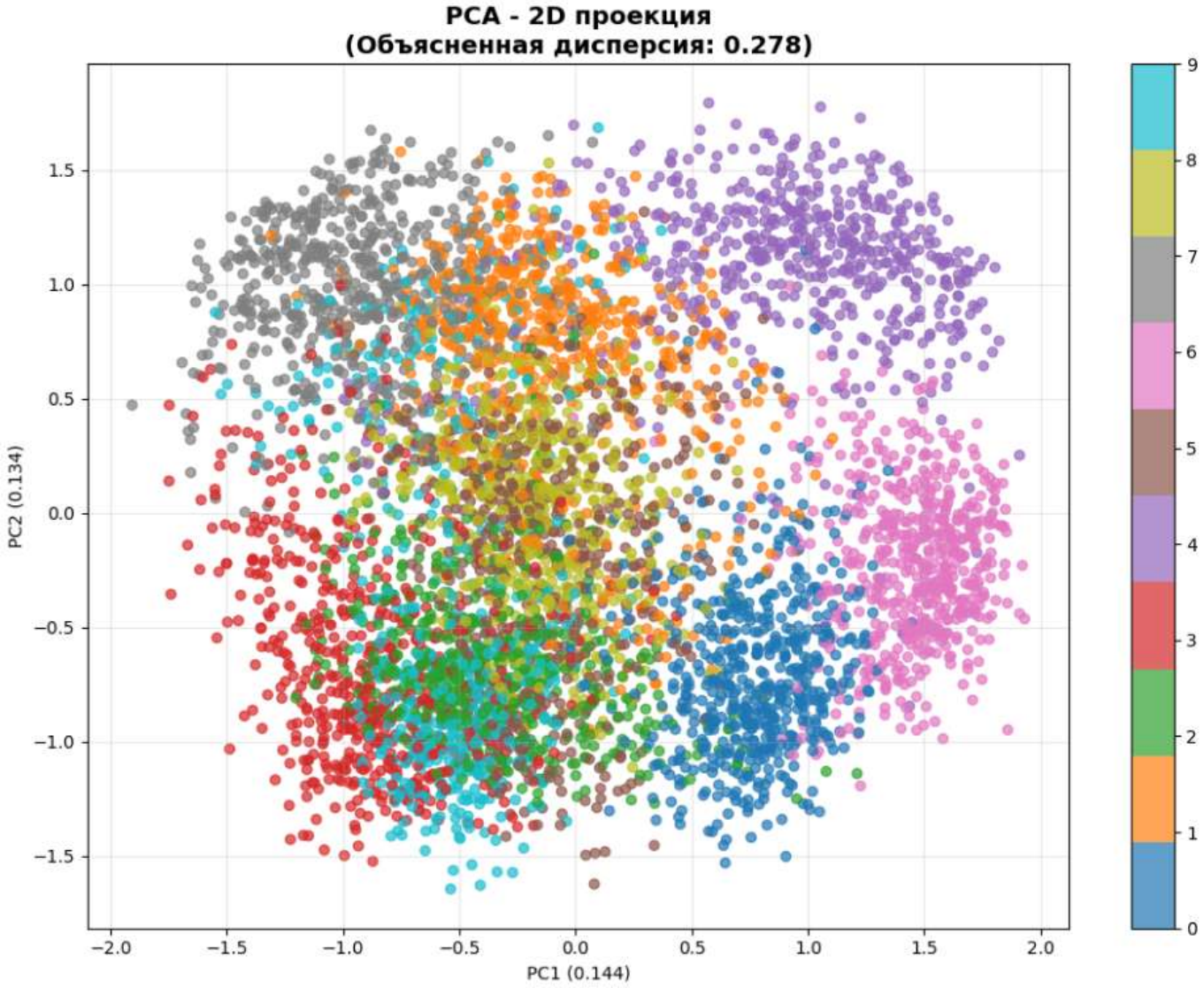
```
print("t-SNE преобразование завершено")

# Визуализация t-SNE
fig = plt.figure(figsize=(20, 8))

# 1. 2D t-SNE
plt.subplot(1, 2, 1)
scatter_tsne_2d = plt.scatter(X_tsne_2d[:, 0], X_tsne_2d[:, 1],
                              c=y, cmap='tab10', alpha=0.7, s=30)
plt.colorbar(scatter_tsne_2d, label='Цифра')
plt.title('t-SNE - 2D проекция', fontsize=14, fontweight='bold')
plt.xlabel('t-SNE компонента 1')
plt.ylabel('t-SNE компонента 2')
plt.grid(True, alpha=0.3)

# 2. 3D t-SNE
ax = plt.subplot(1, 2, 2, projection='3d')
scatter_tsne_3d = ax.scatter(X_tsne_3d[:, 0], X_tsne_3d[:, 1], X_tsne_3d[:, 2],
                             c=y, cmap='tab10', alpha=0.7, s=30)
plt.colorbar(scatter_tsne_3d, label='Цифра')
ax.set_title('t-SNE - 3D проекция', fontsize=14, fontweight='bold')
ax.set_xlabel('t-SNE компонента 1')
ax.set_ylabel('t-SNE компонента 2')
ax.set_zlabel('t-SNE компонента 3')

plt.tight_layout()
plt.show()
```

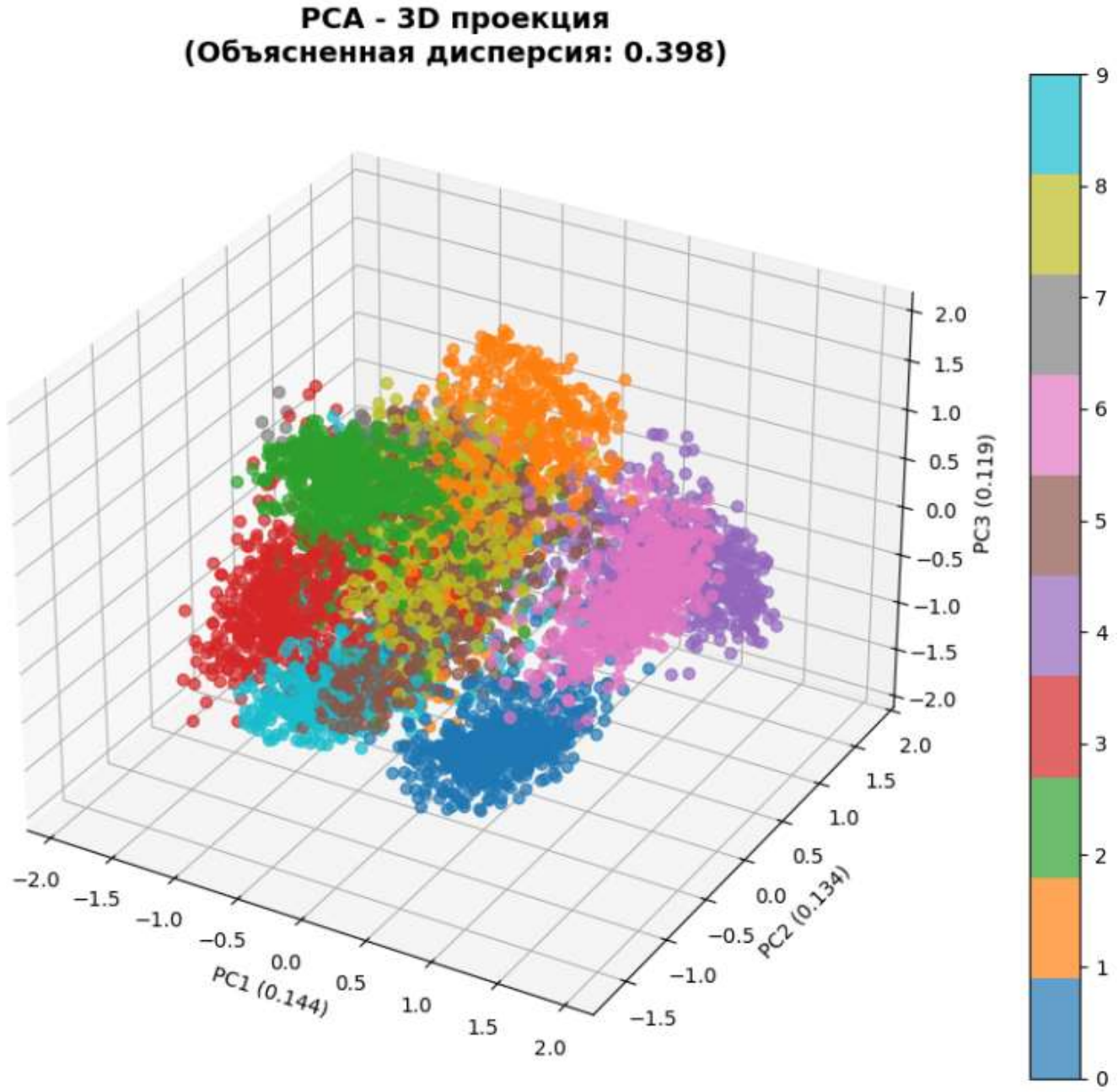

t-SNE - 2D проекция
t-SNE компонента 1
t-SNE компонента 2
−80
−60
−40
−20
0
20
40
60
80
9
8
7
6
5
4
3
2
1
0




```
# PCA проекции для сравнения с другими методами
print("\n" + "=" * 60)
print("PCA ПРОЕКЦИИ ДЛЯ СРАВНЕНИЯ")
print("=" * 60)

# PCA с 2 компонентами
pca_2d = PCA(n_components=2, random_state=42)
X_pca_2d = pca_2d.fit_transform(X_scaled)

# PCA с 3 компонентами
pca_3d = PCA(n_components=3, random_state=42)
X_pca_3d = pca_3d.fit_transform(X_scaled)

print(f"Объясненная дисперсия PCA 2D:
{pca_2d.explained_variance_ratio_.sum():.3f}")
```

```
print(f"Объясненная дисперсия PCA 3D:
{pca_3d.explained_variance_ratio_.sum():.3f}")
print(f"Объясненная дисперсия по компонентам (2D):
{pca_2d.explained_variance_ratio_}")

# Визуализация PCA
fig = plt.figure(figsize=(20, 8))

# 1. PCA 2D
plt.subplot(1, 2, 1)
scatter_pca_2d = plt.scatter(X_pca_2d[:, 0], X_pca_2d[:, 1],
                             c=y, cmap='tab10', alpha=0.7, s=30)
plt.colorbar(scatter_pca_2d, label='Цифра')
plt.title(f'PCA - 2D проекция\n(Объясненная дисперсия:
{pca_2d.explained_variance_ratio_.sum():.3f})',
          fontsize=14, fontweight='bold')
plt.xlabel(f'PC1 ({pca_2d.explained_variance_ratio_[0]:.3f})')
plt.ylabel(f'PC2 ({pca_2d.explained_variance_ratio_[1]:.3f})')
plt.grid(True, alpha=0.3)

# 2. PCA 3D
ax = plt.subplot(1, 2, 2, projection='3d')
scatter_pca_3d = ax.scatter(X_pca_3d[:, 0], X_pca_3d[:, 1], X_pca_3d[:, 2],
                            c=y, cmap='tab10', alpha=0.7, s=30)
plt.colorbar(scatter_pca_3d, label='Цифра')
ax.set_title(f'PCA - 3D проекция\n(Объясненная дисперсия:
{pca_3d.explained_variance_ratio_.sum():.3f})',
             fontsize=14, fontweight='bold')
ax.set_xlabel(f'PC1 ({pca_3d.explained_variance_ratio_[0]:.3f})')
ax.set_ylabel(f'PC2 ({pca_3d.explained_variance_ratio_[1]:.3f})')
ax.set_zlabel(f'PC3 ({pca_3d.explained_variance_ratio_[2]:.3f})')

plt.tight_layout()
plt.show()
```


PCA - 2D проекция
(Объясненная дисперсия: 0.278)
PC1 (0.144)
PC2 (0.134)
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
0
1
2
3
4
5
6
7
8
9



Вывод: научился применять автоэнкодеры для осуществления визуализации данных и их анализа