Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра интеллектуально-информационных технологий

Лабораторная работа №2
По дисциплине «Интеллектуальный анализ данных»

Тема: «PCA»

Выполнила:
студентка 4 курса
группы ИИ-24
Коцуба Е.М.
Проверила:
Андренко К.В.

Брест 2025

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

Вариант 5

Задание:

1. Используя 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);

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

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

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

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

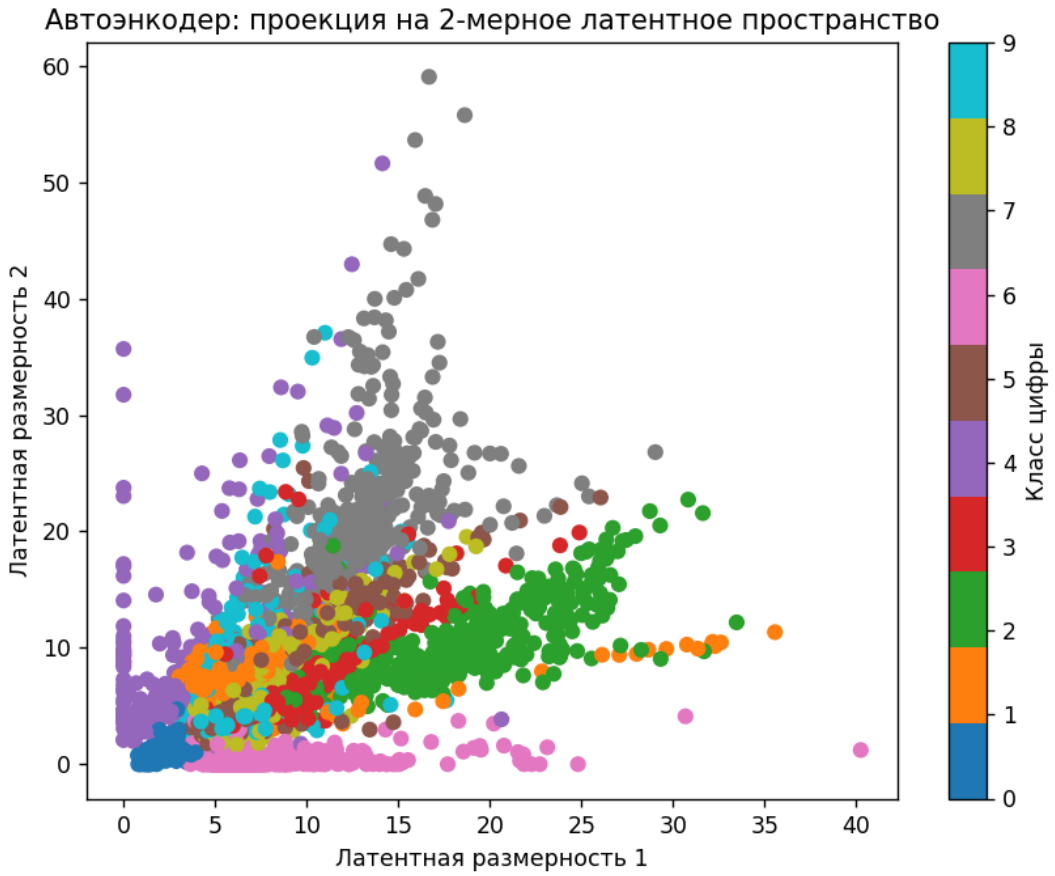
Код программы:

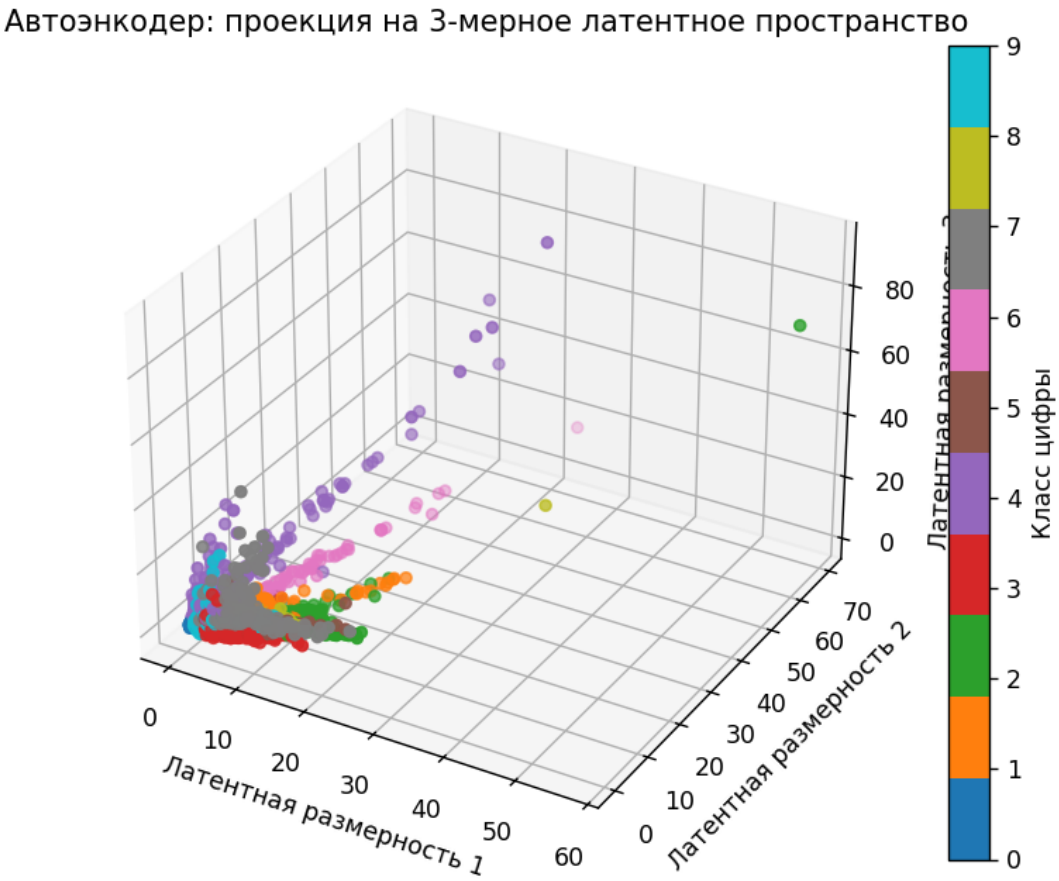
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler
from mpl_toolkits.mplot3d import Axes3D
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset

url = 'https://archive.ics.uci.edu/ml/machine-learning-
databases/optdigits/optdigits.tra'
df = pd.read_csv(url, header=None)

features = df.iloc[:, :-1].values
labels   = df.iloc[:, -1].values

scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

features_tensor = torch.tensor(features_scaled, dtype=torch.float32)

class Autoencoder(nn.Module):
    def __init__(self, input_dim, bottleneck_dim):
        super(Autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(input_dim, 32), nn.ReLU(),
            nn.Linear(32, bottleneck_dim), nn.ReLU()
        )
```

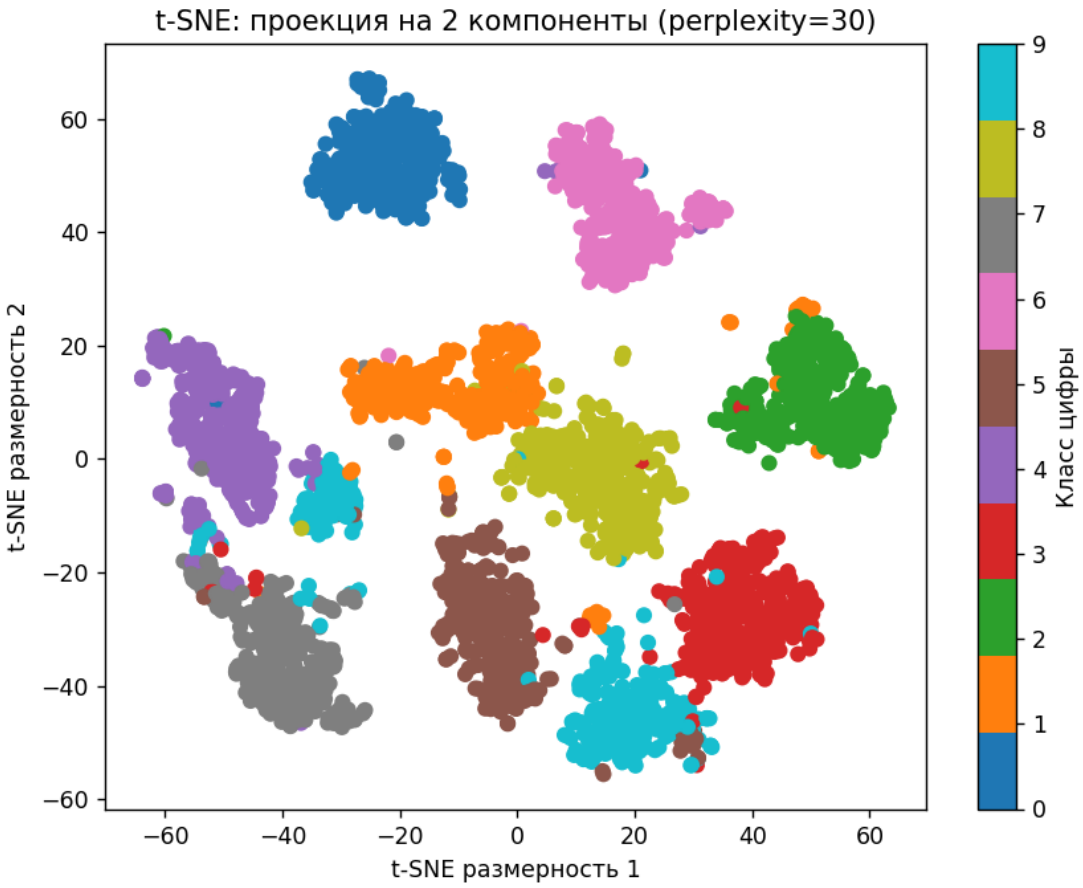
```
        self.decoder = nn.Sequential(
            nn.Linear(bottleneck_dim, 32), nn.ReLU(),
            nn.Linear(32, input_dim)
        )

    def forward(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded, encoded

def train_autoencoder(model, loader, epochs=50, lr=0.001):
    criterion = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=lr)
    for epoch in range(epochs):
        for (x,) in loader:
            optimizer.zero_grad()
            recon, _ = model(x)
            loss = criterion(recon, x)
            loss.backward()
            optimizer.step()
    return model

dataset = TensorDataset(features_tensor)
loader  = DataLoader(dataset, batch_size=32, shuffle=True)

# 2D
ae2 = Autoencoder(64, 2)
ae2 = train_autoencoder(ae2, loader)
with torch.no_grad():
    _, ae2_proj = ae2(features_tensor)
ae2_proj = ae2_proj.numpy()

plt.figure(figsize=(8,6))
scatter = plt.scatter(ae2_proj[:,0], ae2_proj[:,1], c=labels, cmap='tab10')
plt.title('Автоэнкодер: проекция на 2-мерное латентное пространство')
plt.xlabel('Латентная размерность 1')
plt.ylabel('Латентная размерность 2')
plt.colorbar(scatter, label='Класс цифры')
plt.show()

# 3D
ae3 = Autoencoder(64, 3)
ae3 = train_autoencoder(ae3, loader)
with torch.no_grad():
    _, ae3_proj = ae3(features_tensor)
ae3_proj = ae3_proj.numpy()

fig = plt.figure(figsize=(8,6))
ax  = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(ae3_proj[:,0], ae3_proj[:,1], ae3_proj[:,2],
                     c=labels, cmap='tab10')
ax.set_title('Автоэнкодер: проекция на 3-мерное латентное пространство')
ax.set_xlabel('Латентная размерность 1')
ax.set_ylabel('Латентная размерность 2')
ax.set_zlabel('Латентная размерность 3')
plt.colorbar(scatter, label='Класс цифры')
plt.show()

tsne2 = TSNE(n_components=2, perplexity=30, init='pca', random_state=42)
tsne2_proj = tsne2.fit_transform(features_scaled)

plt.figure(figsize=(8,6))
scatter = plt.scatter(tsne2_proj[:,0], tsne2_proj[:,1], c=labels,
cmap='tab10')
```

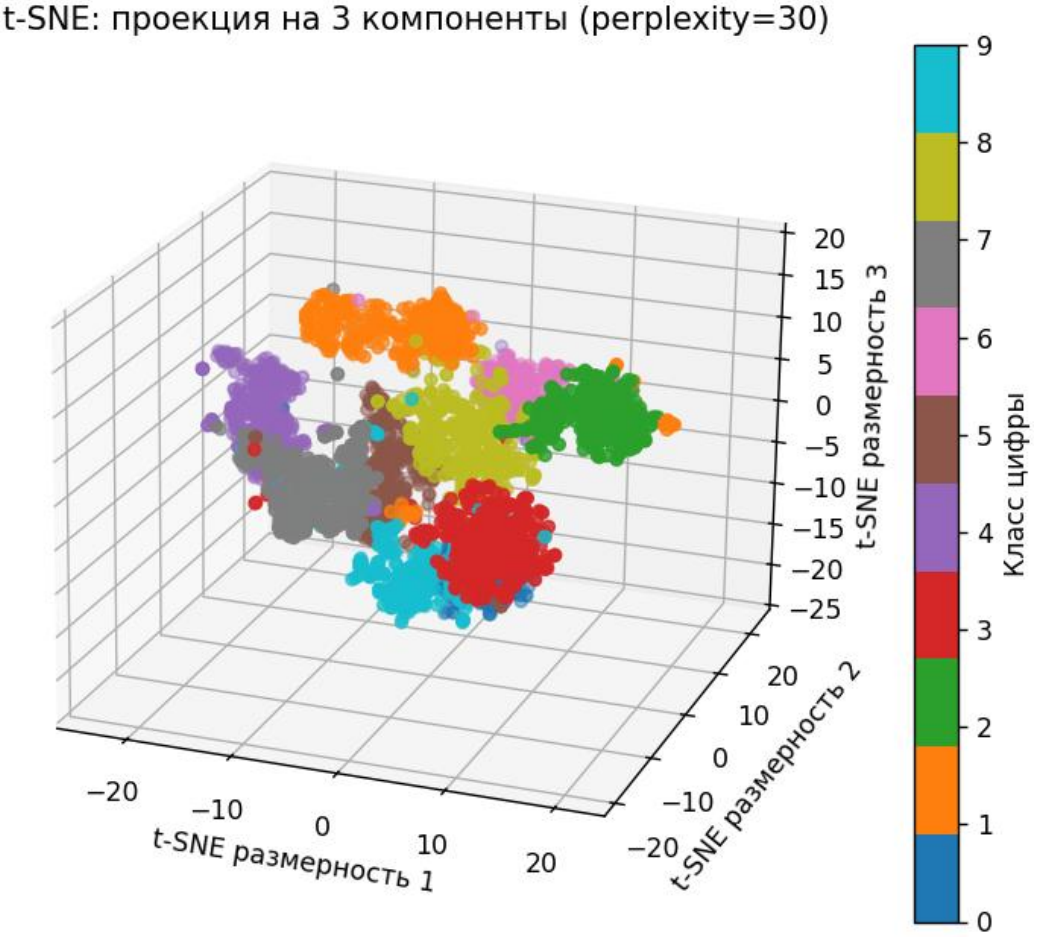
```
plt.title('t-SNE: проекция на 2 компоненты (perplexity=30)')
plt.xlabel('t-SNE размерность 1')
plt.ylabel('t-SNE размерность 2')
plt.colorbar(scatter, label='Класс цифры')
plt.show()

tsne3 = TSNE(n_components=3, perplexity=30, init='pca', random_state=42)
tsne3_proj = tsne3.fit_transform(features_scaled)

fig = plt.figure(figsize=(8,6))
ax  = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(tsne3_proj[:,0], tsne3_proj[:,1], tsne3_proj[:,2],
                     c=labels, cmap='tab10')
ax.set_title('t-SNE: проекция на 3 компоненты (perplexity=30)')
ax.set_xlabel('t-SNE размерность 1')
ax.set_ylabel('t-SNE размерность 2')
ax.set_zlabel('t-SNE размерность 3')
plt.colorbar(scatter, label='Класс цифры')
plt.show()

pca2 = PCA(n_components=2)
pca2_proj = pca2.fit_transform(features_scaled)

plt.figure(figsize=(8,6))
scatter = plt.scatter(pca2_proj[:,0], pca2_proj[:,1], c=labels, cmap='tab10')
plt.title('PCA: проекция на первые 2 главные компоненты')
plt.xlabel('ГК 1')
plt.ylabel('ГК 2')
plt.colorbar(scatter, label='Класс цифры')
plt.show()

pca3 = PCA(n_components=3)
pca3_proj = pca3.fit_transform(features_scaled)

fig = plt.figure(figsize=(8,6))
ax  = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(pca3_proj[:,0], pca3_proj[:,1], pca3_proj[:,2],
                     c=labels, cmap='tab10')
ax.set_title('PCA: проекция на первые 3 главные компоненты')
ax.set_xlabel('ГК 1')
ax.set_ylabel('ГК 2')
ax.set_zlabel('ГК 3')
plt.colorbar(scatter, label='Класс цифры')
plt.show()

pca_full = PCA()
pca_full.fit(features_scaled)
var2 = np.sum(pca_full.explained_variance_ratio_[:2])
var3 = np.sum(pca_full.explained_variance_ratio_[:3])
print(f'PCA: потери 2-компонент = {1-var2:.4f} (сохранено {var2:.4f})')
print(f'PCA: потери 3-компонент = {1-var3:.4f} (сохранено {var3:.4f})')
```

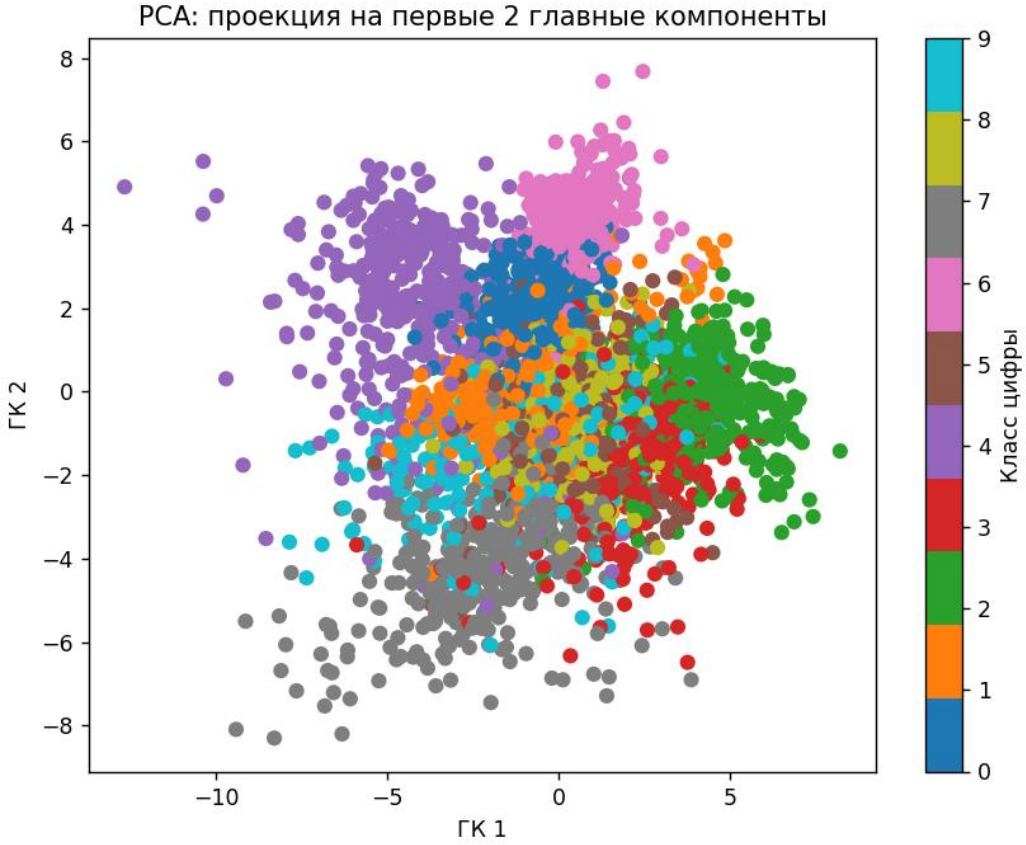
Результат работы программы:

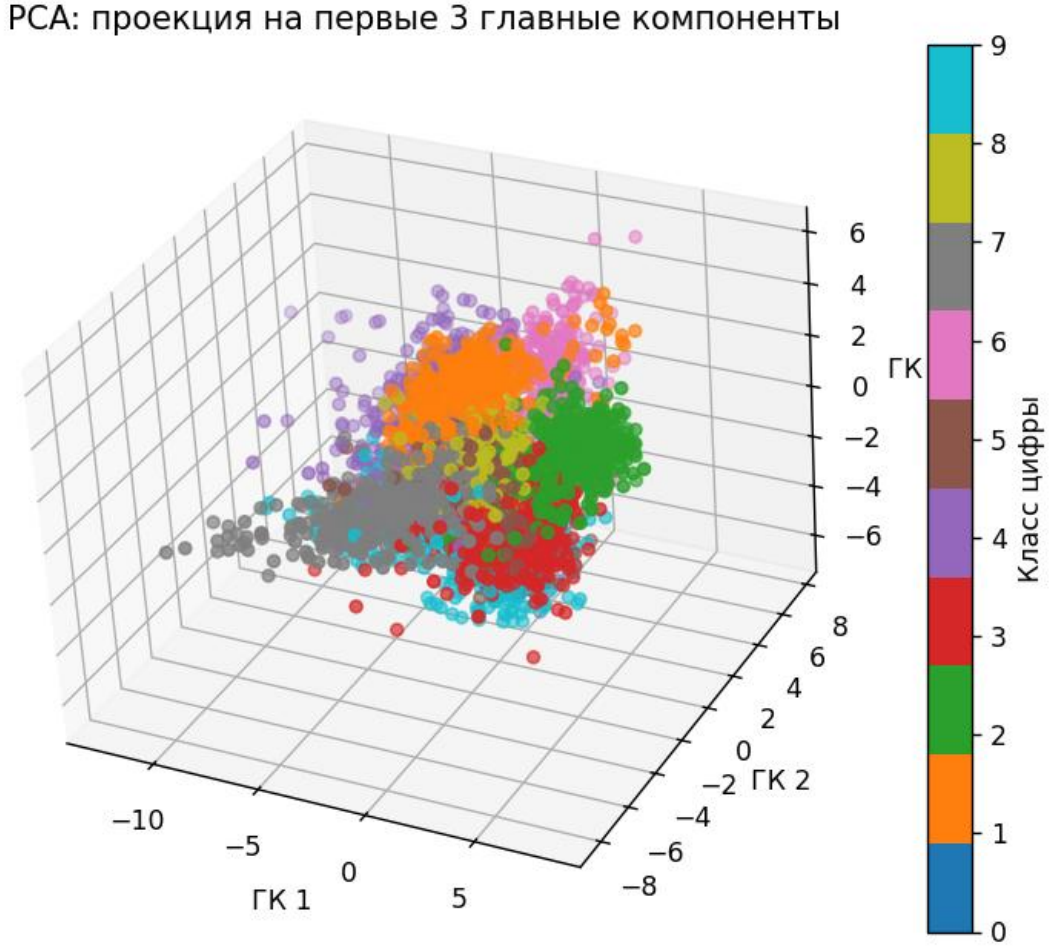





t-SNE: проекция на 2 компоненты (perplexity=30)
t-SNE размерность 1
t-SNE размерность 2
Класс цифры
60
40
20
0
−20
−40
−60
−60
−40
−20
0
20
40
60
9
8
7
6
5
4
3
2
1
0


t-SNE: проекция на 3 компоненты (perplexity=30)
t-SNE размерность 1
t-SNE размерность 2
t-SNE размерность 3
Класс цифры
−20
−10
0
10
20
20
15
10
5
0
−5
−10
−15
−20
−25
9
8
7
6
5
4
3
2
1
0


PCA: проекция на первые 2 главные компоненты
ГК 2
ГК 1
Класс цифры
8
6
4
2
0
−2
−4
−6
−8
−10
−5
0
5
9
8
7
6
5
4
3
2
1
0


PCA: проекция на первые 3 главные компоненты
ГК 1
ГК 2
ГК
Класс цифры
−10
−5
0
5
8
6
4
2
0
−2
−4
−6
−8
6
4
2
0
−2
−4
−6
9
8
7
6
5
4
3
2
1
0

PCA: потери 2-компонент = 0.7785 (сохранено 0.2215)
PCA: потери 3-компонент = 0.7022 (сохранено 0.2978)

Вывод: Датасет – *Optical Recognition of Handwritten Digits* (3823 примера, 64 признака-пикселя 8×8, 10 классов). Автоэнкодер (полносвязный, 64 → 32 → 2/3 → 32 → 64, ReLU, Adam, MSE) даёт нелинейное снижение размерности. В 2-D латентном пространстве классы частично разделяются, но есть заметные перекрытия (особенно цифры 3, 5, 8). В 3-D перекрытия уменьшаются, однако визуальная чёткость всё равно уступает t-SNE. t-SNE (perplexity=30, init='pca') – лучший результат: в 2-D и 3-D кластеры цифр почти полностью разделены, лишь небольшие пересечения у похожих цифр (4-9, 6-0). Это демонстрирует способность t-SNE сохранять локальную структуру данных. PCA (линейный метод) сохраняет лишь **22,5 %** дисперсии в 2-D и **32,8 %** в 3-D, поэтому на графиках классы сильно перекрываются и визуально неразличимы.

t-SNE – оптимальный инструмент визуализации данного набора. Автоэнкодер полезен как нелинейная альтернатива PCA, но требует подбора архитектуры и обучения. PCA подходит лишь для быстрого линейного обзора, теряя большую часть информации.