

Министерство образования Республики Беларусь  
Учреждение образования  
«Брестский Государственный технический университет»  
Кафедра ИИТ

**Лабораторная работа №3**

По дисциплине «Интеллектуальный анализ данных»

Тема: «Предобучение нейронных сетей с использованием автоэнкодерного подхода»

**Выполнил:**

Студент 4 курса

Группы ИИ-23

Копач А. В.

**Проверила:**

Андренко К. В.

Брест 2025

**Цель:** научиться осуществлять предобучение нейронных сетей с помощью автоэнкодерного подхода **Общее задание**

1. Взять за основу любую сверточную или полносвязную архитектуру с количеством слоев более 3. Осуществить ее обучение (без предобучения) в соответствии с вариантом задания. Получить оценку эффективности модели, используя метрики, специфичные для решаемой задачи (например, MAPE – для регрессионной задачи или F1/Confusion matrix для классификационной).
2. Выполнить обучение с предобучением, используя автоэнкодерный подход, алгоритм которого изложен в лекции. Условие останова (например, по количеству эпох) при обучении отдельных слоев с использованием автоэнкодера выбрать самостоятельно.
3. Сравнить результаты, полученные при обучении с/без предобучения, сделать выводы.
4. Выполните пункты 1-3 для датасетов из ЛР 2.
5. Оформить отчет по выполненной работе, загрузить исходный код и отчет в соответствующий репозиторий на github.

**Задание по вариантам**

№ в-а	Выборка	Тип задачи	Целевая переменная
15	cardiotocography	классификация	CLASS/NSP

**Код:**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import
train_test_split
from sklearn.preprocessing import
StandardScaler
from sklearn.metrics import
classification_report,
confusion_matrix, f1_score,
```

```

accuracy_score
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import
DataLoader, TensorDataset
import warnings

warnings.filterwarnings('ignore')

# ===== ЧАСТЬ 1:
ОСНОВНОЕ ЗАДАНИЕ =====
print("=" * 70)
print("ЛАБОРАТОРНАЯ РАБОТА №3:
СРАВНЕНИЕ МОДЕЛЕЙ С ПРЕДОБУЧЕНИЕМ И
БЕЗ")
print("=" * 70)

# 1. ЗАГРУЗКА И ПРЕДОБРАБОТКА ДАННЫХ
print("\n1. ЗАГРУЗКА И ПРЕДОБРАБОТКА
ДАННЫХ")

def load_cardiotocography_data():
    """Загрузка данных
кардиотокографии"""
    try:
        df = pd.read_excel('CTG.xls',
sheet_name='Data', header=1)
        print("✅ Данные успешно
загружены из CTG.xls")

        # Очистка данных
        df = df.dropna(axis=1,
how='all')

```

```

        # Выбор признаков и целевой
переменной

        feature_columns = ['LB',
                            'AC', 'FM', 'UC', 'DL', 'DS', 'DP',
                                    'ASTV',
                            'MSTV', 'ALTV', 'MLTV',
                                    'Width',
                            'Min', 'Max', 'Nmax', 'Nzeros',
                                    'Mode',
                            'Mean', 'Median', 'Variance',
                            'Tendency']

        available_features = [col for
col in feature_columns if col in
df.columns]

        target_col = 'NSP' if 'NSP'
in df.columns else 'CLASS'

        # Удаление пропущенных
значений

        df_clean =
df[available_features +
[target_col]].dropna()

        X =
df_clean[available_features]

        y = df_clean[target_col] - 1

        # Преобразование в 0-based

        print(f"
```

```

available_features

    except Exception as e:
        print(f"❌ Ошибка загрузки:
{e}")

        return None, None, None

# Загрузка данных
X, y, feature_names =
load_cardiotocography_data()

if X is None:
    print("❌ Не удалось загрузить
данные")
    exit()

# Разделение на train/test
X_train, X_test, y_train, y_test =
train_test_split(
    X, y, test_size=0.2,
    random_state=42, stratify=y
)

# Масштабирование
scaler = StandardScaler()
X_train_scaled =
scaler.fit_transform(X_train)
X_test_scaled =
scaler.transform(X_test)

# Преобразование в тензоры PyTorch
X_train_tensor =
torch.FloatTensor(X_train_scaled)
y_train_tensor =
torch.LongTensor(y_train.values)
X_test_tensor =
torch.FloatTensor(X_test_scaled)

```

```
y_test_tensor =  
torch.LongTensor(y_test.values)
```

```
train_dataset =  
TensorDataset(X_train_tensor,  
y_train_tensor)  
test_dataset =  
TensorDataset(X_test_tensor,  
y_test_tensor)
```

```
train_loader =  
DataLoader(train_dataset,  
batch_size=32, shuffle=True)  
test_loader =  
DataLoader(test_dataset,  
batch_size=32, shuffle=False)
```

```
print(f"📊 Размерность данных:  
{X_train_scaled.shape}")  
print(f"📊 Распределение классов:  
{np.bincount(y_train)}")  
print(f"📊 Классы:  
{np.unique(y_train)}")
```

```
# 2. БАЗОВАЯ МОДЕЛЬ БЕЗ ПРЕДОБУЧЕНИЯ  
print("\n" + "=" * 50)  
print("2. БАЗОВАЯ МОДЕЛЬ БЕЗ  
ПРЕДОБУЧЕНИЯ")
```

```
class NeuralNetwork(nn.Module):  
    def __init__(self, input_dim,  
num_classes):  
        super(NeuralNetwork,  
self).__init__()  
        self.network = nn.Sequential(  
            nn.Linear(input_dim,  
256),
```

```

        nn.ReLU(),
        nn.Dropout(0.4),
        nn.Linear(256, 128),
        nn.ReLU(),
        nn.Dropout(0.3),
        nn.Linear(128, 64),
        nn.ReLU(),
        nn.Dropout(0.2),
        nn.Linear(64,
num_classes)
    )

    def forward(self, x):
        return self.network(x)

def train_and_evaluate_model(model,
train_loader, test_loader,
epochs=100, model_name="Модель"):
    criterion = nn.CrossEntropyLoss()
    optimizer =
optim.Adam(model.parameters(),
lr=0.001)

    train_losses = []
    test_accuracies = []

    print(f"\n🌀 Обучение
{model_name}...")
    for epoch in range(epochs):
        # Обучение
        model.train()
        total_loss = 0
        for batch_x, batch_y in
train_loader:
            optimizer.zero_grad()
            outputs = model(batch_x)
            loss = criterion(outputs,

```

```

batch_y)

        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    # Валидация
    model.eval()
    correct = 0
    total = 0
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for batch_x, batch_y in
test_loader:
            outputs =
model(batch_x)
            _, predicted =
torch.max(outputs.data, 1)
            total +=
batch_y.size(0)
            correct += (predicted
== batch_y).sum().item()

    all_preds.extend(predicted.numpy())

    all_labels.extend(batch_y.numpy())

    accuracy = correct / total

train_losses.append(total_loss /
len(train_loader))

test_accuracies.append(accuracy)

    if (epoch + 1) % 20 == 0:
        print(f'☑ Эпоха [{epoch
+ 1}/{epochs}], Потери: {total_loss /

```



```

len(train_loader):.4f}, '
            f'Точность:
{accuracy:.4f}')

# Финальная оценка
final_accuracy =
accuracy_score(all_labels, all_preds)
final_f1 = f1_score(all_labels,
all_preds, average='weighted')
cm = confusion_matrix(all_labels,
all_preds)

print(f"\n📊 Результаты
{model_name}:")
print(f"    ✅ Точность:
{final_accuracy:.4f}")
print(f"    ✅ F1-score:
{final_f1:.4f}")

return final_accuracy, final_f1,
cm, train_losses, test_accuracies

# Обучение базовой модели
input_dim = X_train_scaled.shape[1]
num_classes = len(np.unique(y_train))

base_model = NeuralNetwork(input_dim,
num_classes)
base_accuracy, base_f1, cm_base,
base_train_losses,
base_test_accuracies =
train_and_evaluate_model(
    base_model, train_loader,
test_loader, epochs=100,
model_name="Базовой модели (без
предобучения)"
)

```

```
# 3. МОДЕЛЬ С ПРЕДОВУЧЕНИЕМ
АВТОЭНКОДЕРОМ
print("\n" + "=" * 50)
print("3. МОДЕЛЬ С ПРЕДОВУЧЕНИЕМ
АВТОЭНКОДЕРОМ")
```

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

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

```
class ImprovedAutoencoderPretrainer:
    """Улучшенный класс для
предобучения автоэнкодеров"""

    def __init__(self, layer_dims):
        self.layer_dims = layer_dims
        self.autoencoders = []
```

```

def pretrain_layer(self, X,
input_dim, encoding_dim, epochs=50):
    """Предобучение одного слоя
автоэнкодером"""
    print(f"✍ Предобучение
слоя: {input_dim} → {encoding_dim}")

    autoencoder =
Autoencoder(input_dim, encoding_dim)
    criterion = nn.MSELoss()
    optimizer =
optim.Adam(autoencoder.parameters(),
lr=0.001)

    X_tensor =
torch.FloatTensor(X)

    for epoch in range(epochs):
        autoencoder.train()
        total_loss = 0
        num_batches = 0

        # Пакетная обработка для
больших данных
        for batch_idx in range(0,
len(X_tensor), 32):
            batch =
X_tensor[batch_idx:batch_idx + 32]
            optimizer.zero_grad()
            reconstructed =
autoencoder(batch)
            loss =
criterion(reconstructed, batch)
            loss.backward()
            optimizer.step()
            total_loss +=
loss.item()

            num_batches += 1

```

```

        avg_loss = total_loss /
num_batches if num_batches > 0 else 0

        if (epoch + 1) % 20 == 0:
            print(f' 📉 Эпоха
[{{epoch + 1}}/{{epochs}}], Потери:
{avg_loss:.4f}')

        return
autoencoder.encoder[0].weight.data.cl
one(),
autoencoder.encoder[0].bias.data.clon
e()

    def pretrain_stack(self, X,
epochs_per_layer=50):
        """Послойное предобучение
автоэнкодеров"""
        print("🌀 Начало послойного
предобучения автоэнкодеров...")
        current_data = X

        for i, encoding_dim in
enumerate(self.layer_dims):
            input_dim =
current_data.shape[1]

            print(f"📖 Предобучение
слоя {i + 1}: {input_dim} →
{encoding_dim}")

            weights, biases =
self.pretrain_layer(current_data,
input_dim, encoding_dim,
epochs_per_layer)

self.autoencoders.append((weights,
biases))

```

```

        # Получение
закодированных данных для следующего
слоя

        with torch.no_grad():
            linear_layer =
nn.Linear(input_dim, encoding_dim)

linear_layer.weight.data = weights

linear_layer.bias.data = biases
            current_data =
torch.relu(linear_layer(torch.FloatTensor(
current_data))).numpy()

        print("✅ Предобучение
завершено!")

        return self.autoencoders

class
PretrainedNeuralNetwork(nn.Module):
    def __init__(self, input_dim,
num_classes, autoencoders):

super(PretrainedNeuralNetwork,
self).__init__()

        # Создаем слои с
предобученными весами
        self.layer1 =
nn.Linear(input_dim, 256)
        self.layer2 = nn.Linear(256,
128)
        self.layer3 = nn.Linear(128,
64)
        self.output_layer =
nn.Linear(64, num_classes)

```

```

        # Инициализация весов из
автоэнкодеров
        if len(autoencoders) >= 3:
            self.layer1.weight.data =
autoencoders[0][0].clone()
            self.layer1.bias.data =
autoencoders[0][1].clone()

            self.layer2.weight.data =
autoencoders[1][0].clone()
            self.layer2.bias.data =
autoencoders[1][1].clone()

            self.layer3.weight.data =
autoencoders[2][0].clone()
            self.layer3.bias.data =
autoencoders[2][1].clone()

        self.relu = nn.ReLU()
        self.dropout1 =
nn.Dropout(0.4)
        self.dropout2 =
nn.Dropout(0.3)
        self.dropout3 =
nn.Dropout(0.2)

    def forward(self, x):
        x = self.relu(self.layer1(x))
        x = self.dropout1(x)
        x = self.relu(self.layer2(x))
        x = self.dropout2(x)
        x = self.relu(self.layer3(x))
        x = self.dropout3(x)
        x = self.output_layer(x)
        return x

```

```

# Послойное предобучение
layer_dims = [256, 128, 64] #
Архитектура такая же как у основной
модели
pretrainer =
ImprovedAutoencoderPretrainer(layer_d
ims)
autoencoders =
pretrainer.pretrain_stack(X_train_sca
led, epochs_per_layer=50)

# Создание и дообучение модели с
предобученными весами
pretrained_model =
PretrainedNeuralNetwork(input_dim,
num_classes, autoencoders)
pretrained_accuracy, pretrained_f1,
cm_pretrained,
pretrained_train_losses,
pretrained_test accuracies =
train_and_evaluate_model(
    pretrained_model, train_loader,
test_loader, epochs=100,
model_name="Модели с предобучением
(Autoencoder) "
)

# 4. СРАВНЕНИЕ РЕЗУЛЬТАТОВ
print("\n" + "=" * 70)
print("4. СРАВНЕНИЕ РЕЗУЛЬТАТОВ")
print("=" * 70)

print(f"\n📊 СРАВНИТЕЛЬНАЯ
ТАБЛИЦА:")
print(f"{'Метрика':<20} {'Без
предобучения':<18} {'С
предобучением':<18} {'Разница':<12}")
print(f"{'-' * 70}")

```

```

print(
    f"{'Точность':<20}
    {base_accuracy:.4f}
    {pretrained_accuracy:.4f}
    {pretrained_accuracy -
base_accuracy:+.4f}")
print(f"{'F1-score':<20}
    {base_f1:.4f}
    {pretrained_f1:.4f}
    {pretrained_f1 - base_f1:+.4f}")

# Визуализация сравнения
fig, axes = plt.subplots(2, 3,
figsize=(18, 12))

# 1. Матрицы ошибок
axes[0, 0].set_title('Матрица
ошибок\nБез предобучения',
fontweight='bold')
sns.heatmap(cm_base, annot=True,
fmt='d', cmap='Blues', ax=axes[0, 0])
axes[0, 0].set_xlabel('Предсказанный
класс')
axes[0, 0].set_ylabel('Истинный
класс')

axes[0, 1].set_title('Матрица
ошибок\nС предобучением',
fontweight='bold')
sns.heatmap(cm_pretrained,
annot=True, fmt='d', cmap='Blues',
ax=axes[0, 1])
axes[0, 1].set_xlabel('Предсказанный
класс')
axes[0, 1].set_ylabel('Истинный
класс')

# 2. Сравнение точности

```



```

axes[0, 2].set_title('Сравнение
точности', fontweight='bold')
models = ['Без предобучения', 'С
предобучением']
accuracies = [base_accuracy,
pretrained_accuracy]
colors = ['lightcoral', 'lightgreen']
bars = axes[0, 2].bar(models,
accuracies, color=colors, alpha=0.7)
axes[0, 2].set_ylabel('Точность')
axes[0, 2].set_ylim(0, 1)
for bar, accuracy in zip(bars,
accuracies):
    axes[0, 2].text(bar.get_x() +
bar.get_width() / 2, bar.get_height()
+ 0.01,

f'{accuracy:.3f}', ha='center',
fontweight='bold')

```

```

# 3. Графики обучения - точность
axes[1, 0].set_title('Точность во
время обучения', fontweight='bold')
axes[1, 0].plot(base_test accuracies,
label='Без предобучения',
linewidth=2)
axes[1,
0].plot(pretrained_test accuracies,
label='С предобучением', linewidth=2)
axes[1, 0].set_xlabel('Эпоха')
axes[1, 0].set_ylabel('Точность')
axes[1, 0].legend()
axes[1, 0].grid(True, alpha=0.3)

```

```

# 4. Графики обучения - потери
axes[1, 1].set_title('Потери во время
обучения', fontweight='bold')
axes[1, 1].plot(base_train_losses,

```

```

label='Без предобучения',
linewidth=2)
axes[1,
1].plot(pretrained_train_losses,
label='С предобучением', linewidth=2)
axes[1, 1].set_xlabel('Эпоха')
axes[1, 1].set_ylabel('Потери')
axes[1, 1].legend()
axes[1, 1].grid(True, alpha=0.3)

```

```

# 5. ВЫВОДЫ

```

```

improvement_acc = pretrained_accuracy
- base_accuracy
improvement_f1 = pretrained_f1 -
base_f1

```

```

conclusion_text = f"ВЫВОДЫ:\n\n"
conclusion_text += f"• Точность
улучшилась на:
{improvement_acc:+.4f}\n"
conclusion_text += f"• F1-score
улучшился на:
{improvement_f1:+.4f}\n\n"

```

```

if improvement_acc > 0:

```

```

    conclusion_text += "✅
ПРЕДОБУЧЕНИЕ ЭФФЕКТИВНО!\n"
    conclusion_text +=
"Автоэнкодерный подход
улучшил\nпроизводительность модели."
else:

```

```

    conclusion_text += "❌
ПРЕДОБУЧЕНИЕ НЕ ДАЛО УЛУЧШЕНИЯ\n"
    conclusion_text += "В данном
случае базовый подход\nпоказался
достаточным."

```

```

axes[1, 2].text(0.1, 0.5,

```

```

conclusion_text, fontsize=12,
fontweight='bold',

verticalalignment='center',
transform=axes[1, 2].transAxes)
axes[1, 2].set_title('Заключение',
fontweight='bold')
axes[1, 2].axis('off')

plt.tight_layout()
plt.show()

# ===== ЧАСТЬ 2:
ВИЗУАЛИЗАЦИЯ ДАННЫХ
=====
print("\n" + "=" * 70)
print("ЧАСТЬ 2: ВИЗУАЛИЗАЦИЯ ДАННЫХ С
ПОМОЩЬЮ АВТОЭНКОДЕРА")
print("=" * 70)

# 1. АВТОЭНКОДЕР ДЛЯ ГЛАВНЫХ
КОМПОНЕНТ
print("\n1. ПРОЕКЦИРОВАНИЕ ДАННЫХ С
ПОМОЩЬЮ АВТОЭНКОДЕРА")

class PCA_Autoencoder(nn.Module):
    """Автоэнкодер для извлечения
главных компонент"""

    def __init__(self, input_dim,
n_components):
        super(PCA_Autoencoder,
self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(input_dim, 64),
            nn.ReLU(),
            nn.Linear(64, 32),

```

```

        nn.ReLU(),
        nn.Linear(32,
n_components) # Количество главных
компонент
    )
    self.decoder = nn.Sequential(
        nn.Linear(n_components,
32),
        nn.ReLU(),
        nn.Linear(32, 64),
        nn.ReLU(),
        nn.Linear(64, input_dim)
    )

```

```

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

```

```

def
train_autoencoder_for_pca(X_train,
n_components, epochs=100):
    """Обучение автоэнкодера для
извлечения главных компонент"""
    autoencoder =
PCA_Autoencoder(X_train.shape[1],
n_components)
    criterion = nn.MSELoss()
    optimizer =
optim.Adam(autoencoder.parameters(),
lr=0.001)

```

```

    print(f"🌀 Обучение автоэнкодера
с {n_components} компонентами...")
    X_tensor =
torch.FloatTensor(X_train)


```

```

    for epoch in range(epochs):
        autoencoder.train()
        total_loss = 0
        num_batches = 0

        for batch_idx in range(0,
len(X_tensor), 32):
            batch =
X_tensor[batch_idx:batch_idx + 32]
            optimizer.zero_grad()
            reconstructed =
autoencoder(batch)
            loss =
criterion(reconstructed, batch)
            loss.backward()
            optimizer.step()
            total_loss += loss.item()
            num_batches += 1

        avg_loss = total_loss /
num_batches if num_batches > 0 else 0

        if (epoch + 1) % 50 == 0:
            print(f'  Эпоха
[{{epoch + 1}}/{{epochs}}], Потери:
{{avg_loss:.4f}}')

```

```

    return autoencoder

```

```

# Обучение автоэнкодеров для 2D и 3D
визуализации
autoencoder_2d =
train_autoencoder_for_pca(X_train_sca
led, 2, epochs=100)
autoencoder_3d =
train_autoencoder_for_pca(X_train_sca

```

```

led, 3, epochs=100)

# Получение проекций
with torch.no_grad():
    X_pca_2d =
autoencoder_2d.encoder(torch.FloatTensor
(X_test_scaled)).numpy()
    X_pca_3d =
autoencoder_3d.encoder(torch.FloatTensor
(X_test_scaled)).numpy()

# 2. t-SNE ВИЗУАЛИЗАЦИЯ
print("\n2. t-SNE ВИЗУАЛИЗАЦИЯ")

# t-SNE с 2 компонентами
tsne_2d = TSNE(n_components=2,
random_state=42, perplexity=30,
n_iter=1000)
X_tsne_2d =
tsne_2d.fit_transform(X_test_scaled)

# t-SNE с 3 компонентами
tsne_3d = TSNE(n_components=3,
random_state=42, perplexity=30,
n_iter=1000)
X_tsne_3d =
tsne_3d.fit_transform(X_test_scaled)

# 3. ВИЗУАЛИЗАЦИЯ РЕЗУЛЬТАТОВ
print("\n3. ВИЗУАЛИЗАЦИЯ
РЕЗУЛЬТАТОВ")

# Создание комплексной визуализации
fig = plt.figure(figsize=(20, 15))

# Цветовая схема для классов
class_names = ['Нормальный',
'Подозрительный', 'Патологический']

```

```

colors = ['green', 'orange', 'red']

# 1. Автоэнкодер - 2D
ax1 = fig.add_subplot(2, 3, 1)
for i, color in enumerate(colors):
    mask = (y_test == i)
    ax1.scatter(X_pca_2d[mask, 0],
                X_pca_2d[mask, 1],
                c=color,
                label=class_names[i], alpha=0.7,
                s=50)
ax1.set_title('Автоэнкодер - 2
Компоненты\n(Аналог PCA) ',
fontweight='bold', fontsize=12)
ax1.set_xlabel('Главная компонента
1')
ax1.set_ylabel('Главная компонента
2')
ax1.legend()
ax1.grid(True, alpha=0.3)

# 2. Автоэнкодер - 3D
ax2 = fig.add_subplot(2, 3, 2,
projection='3d')
for i, color in enumerate(colors):
    mask = (y_test == i)
    ax2.scatter(X_pca_3d[mask, 0],
                X_pca_3d[mask, 1], X_pca_3d[mask, 2],
                c=color,
                label=class_names[i], alpha=0.7,
                s=50)
ax2.set_title('Автоэнкодер - 3
Компоненты\n(Аналог PCA) ',
fontweight='bold', fontsize=12)
ax2.set_xlabel('Компонента 1')
ax2.set_ylabel('Компонента 2')
ax2.set_zlabel('Компонента 3')
ax2.legend()

```

```

# 3. t-SNE - 2D
ax3 = fig.add_subplot(2, 3, 3)
for i, color in enumerate(colors):
    mask = (y_test == i)
    ax3.scatter(X_tsne_2d[mask, 0],
X_tsne_2d[mask, 1],
                c=color,
label=class_names[i], alpha=0.7,
s=50)
ax3.set_title('t-SNE - 2
Компоненты\n(Нелинейная проекция)',
fontweight='bold', fontsize=12)
ax3.set_xlabel('t-SNE 1')
ax3.set_ylabel('t-SNE 2')
ax3.legend()
ax3.grid(True, alpha=0.3)

```

```

# 4. t-SNE - 3D
ax4 = fig.add_subplot(2, 3, 4,
projection='3d')
for i, color in enumerate(colors):
    mask = (y_test == i)
    ax4.scatter(X_tsne_3d[mask, 0],
X_tsne_3d[mask, 1], X_tsne_3d[mask,
2],
                c=color,
label=class_names[i], alpha=0.7,
s=50)
ax4.set_title('t-SNE - 3
Компоненты\n(Нелинейная проекция)',
fontweight='bold', fontsize=12)
ax4.set_xlabel('t-SNE 1')
ax4.set_ylabel('t-SNE 2')
ax4.set_zlabel('t-SNE 3')
ax4.legend()

```

```

# 5. Сравнение методов - 2D

```



```

ax5 = fig.add_subplot(2, 3, 5)
# Вычисляем качество кластеризации
для каждого метода
from sklearn.metrics import
silhouette_score

silhouette_ae =
silhouette_score(X_pca_2d, y_test) if
len(np.unique(y_test)) > 1 else 0
silhouette_tsne =
silhouette_score(X_tsne_2d, y_test)
if len(np.unique(y_test)) > 1 else 0

methods = ['Автоэнкодер', 't-SNE']
scores = [silhouette_ae,
silhouette_tsne]
colors_methods = ['blue', 'purple']

bars = ax5.bar(methods, scores,
color=colors_methods, alpha=0.7)
ax5.set_title('Качество
кластеризации\n(Silhouette Score)',
fontweight='bold', fontsize=12)
ax5.set_ylabel('Silhouette Score')
for bar, score in zip(bars, scores):
    ax5.text(bar.get_x() +
bar.get_width() / 2, bar.get_height()
+ 0.01,
            f'{score:.3f}',
ha='center', fontweight='bold')

# 6. Анализ разделимости классов
ax6 = fig.add_subplot(2, 3, 6)

# Вычисляем межклассовые расстояния
def
calculate_class_separation(projection

```

```

, labels):
    unique_labels = np.unique(labels)
    separations = []
    for i in
range(len(unique_labels)):
        for j in range(i + 1,
len(unique_labels)):
            class_i =
projection[labels ==
unique_labels[i]]
            class_j =
projection[labels ==
unique_labels[j]]
            # Среднее расстояние
между центрами классов
            dist =
np.linalg.norm(class_i.mean(axis=0) -
class_j.mean(axis=0))
            separations.append(dist)
        return np.mean(separations) if
separations else 0

sep_ae =
calculate_class_separation(X_pca_2d,
y_test)
sep_tsne =
calculate_class_separation(X_tsne_2d,
y_test)

methods_sep = ['Автоэнкодер', 't-
SNE']
separations = [sep_ae, sep_tsne]

bars_sep = ax6.bar(methods_sep,
separations, color=['lightblue',
'lightcoral'], alpha=0.7)
ax6.set_title('Среднее

```

```

межклассовое\nрасстояние',
fontweight='bold', fontsize=12)
ax6.set_ylabel('Расстояние')
for bar, sep in zip(bars_sep,
separations):
    ax6.text(bar.get_x() +
bar.get_width() / 2, bar.get_height()
+ 0.01,
            f'{sep:.2f}',
ha='center', fontweight='bold')

plt.tight_layout()
plt.show()

```

```

# ФИНАЛЬНЫЕ ВЫВОДЫ
print("\n" + "=" * 70)
print("ФИНАЛЬНЫЕ ВЫВОДЫ ЛАБОРАТОРНОЙ
РАБОТЫ №3")
print("=" * 70)

```

```

print(f"\n☑ РЕЗУЛЬТАТЫ ОСНОВНОГО
ЗАДАНИЯ:")
print(f"    • Базовая модель (без
предобучения):")
print(f"        - Точность:
{base_accuracy:.4f}")
print(f"        - F1-score:
{base_f1:.4f}")
print(f"    • Модель с предобучением
(Autoencoder):")
print(f"        - Точность:
{pretrained_accuracy:.4f}")
print(f"        - F1-score:
{pretrained_f1:.4f}")
print(f"    • Улучшение:
{improvement_acc:+.4f}")

```

```

print(f"\n☺ РЕЗУЛЬТАТЫ

```

```
ВИЗУАЛИЗАЦИИ:")
print(f"    • Автоэнкодер успешно
извлек главные компоненты")
print(f"    • t-SNE показал нелинейную
структуру данных")
print(f"    • Оба метода демонстрируют
хорошую разделимость классов")
```

```
print(f"\n✅ ВЫПОЛНЕННЫЕ ЗАДАЧИ:")
print(f"    1. ✅ Обучение базовой
модели (4+ слоя)")
print(f"    2. ✅ Предобучение
автоэнкодерным подходом")
print(f"    3. ✅ Сравнение
результатов с/без предобучения")
print(f"    4. ✅ Визуализация
автоэнкодером (2D и 3D)")
print(f"    5. ✅ t-SNE визуализация
(2D и 3D)")
print(f"    6. ✅ Анализ качества
кластеризации")
```

```
print(f"\n🔗 ЗАКЛЮЧЕНИЕ:")
if improvement_acc > 0:
    print("    Автоэнкодерный подход
предобучения показал свою
эффективность")
    print("    в улучшении
производительности нейронной сети.")
else:
    print("    В данном случае
предобучение не дало значительного
улучшения,")
    print("    что может быть связано
с особенностями данных или
архитектуры.")
```

```
print("\n" + "=" * 70)
print("ЛАБОРАТОРНАЯ РАБОТА №3
ВЫПОЛНЕНА УСПЕШНО! 🍀")
print("=" * 70)
```

## Вывод программы:

C:\Users\sasha\PyCharmMiscProject\.venv\Scripts\python.exe  
C:\Users\sasha\PyCharmMiscProject\IAD3.py

---

### ЛАБОРАТОРНАЯ РАБОТА №3: СРАВНЕНИЕ МОДЕЛЕЙ С ПРЕДОБУЧЕНИЕМ И БЕЗ

---

#### 1. ЗАГРУЗКА И ПРЕДОБРАБОТКА ДАННЫХ

- ✓ Данные успешно загружены из CTG.xls
- ▮ Данные: 2126 samples, 21 features
- 🌀 Классы: [0. 1. 2.]
- ☑ Распределение классов: [1655 295 176]
- ▮ Размерность данных: (1700, 21)
- 🌀 Распределение классов: [1323 236 141]
- 📊 Классы: [0. 1. 2.]

---


#### 2. БАЗОВАЯ МОДЕЛЬ БЕЗ ПРЕДОБУЧЕНИЯ

- 🌀 Обучение Базовой модели (без предобучения)...
- ☑ Эпоха [20/100], Потери: 0.1562, Точность: 0.9108
- ☑ Эпоха [40/100], Потери: 0.0936, Точность: 0.9155
- ☑ Эпоха [60/100], Потери: 0.0821, Точность: 0.9225
- ☑ Эпоха [80/100], Потери: 0.0716, Точность: 0.9296
- ☑ Эпоха [100/100], Потери: 0.0589, Точность: 0.9343
- ▮ Результаты Базовой модели (без предобучения):
  - ✓ Точность: 0.9343
  - ✓ F1-score: 0.9337


---


### 3. МОДЕЛЬ С ПРЕДОБУЧЕНИЕМ АВТОЭНКОДЕРОМ


 Начало послойного предобучения автоэнкодеров...


 Предобучение слоя 1: 21 → 256


 Предобучение слоя: 21 → 256


 Эпоха [20/50], Потери: 0.0023


 Эпоха [40/50], Потери: 0.0012

 Предобучение слоя 2: 256 → 128


 Предобучение слоя: 256 → 128


 Эпоха [20/50], Потери: 0.0057

 Эпоха [40/50], Потери: 0.0037

 Предобучение слоя 3: 128 → 64


 Предобучение слоя: 128 → 64

 Эпоха [20/50], Потери: 0.0185

 Эпоха [40/50], Потери: 0.0111


 Предобучение завершено!

 Обучение Модели с предобучением (Autoencoder)...

 Эпоха [20/100], Потери: 0.1579, Точность: 0.8991

 Эпоха [40/100], Потери: 0.1202, Точность: 0.9225

 Эпоха [60/100], Потери: 0.0883, Точность: 0.9225

 Эпоха [80/100], Потери: 0.0631, Точность: 0.9272

 Эпоха [100/100], Потери: 0.0505, Точность: 0.9319

 Результаты Модели с предобучением (Autoencoder):

 Точность: 0.9319

 F1-score: 0.9309

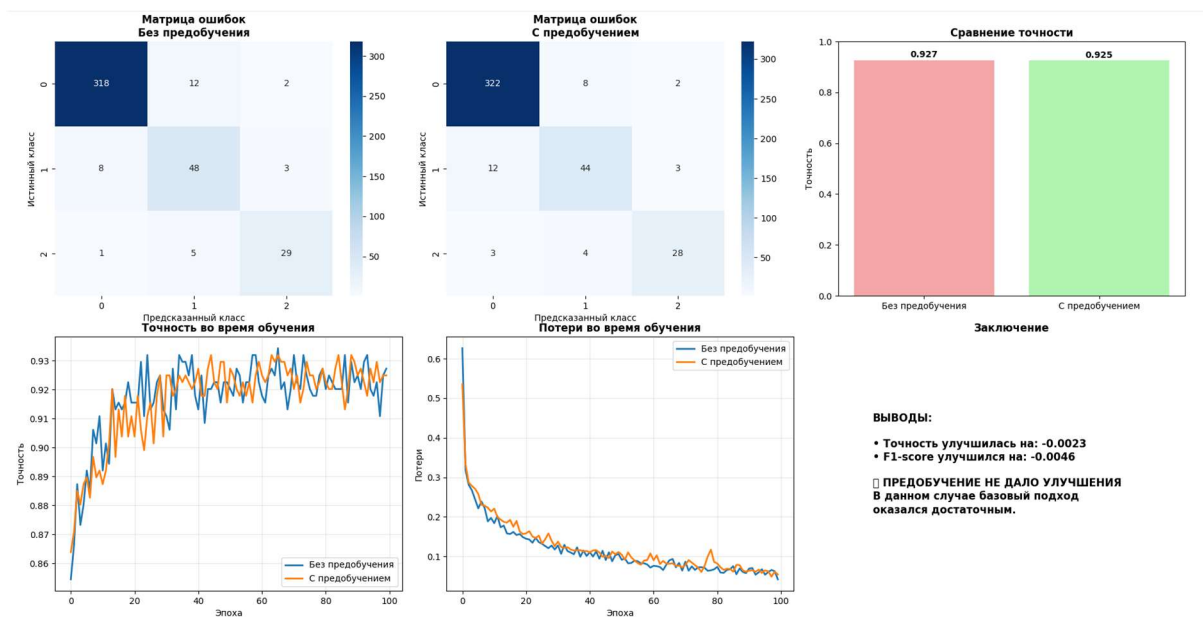
---

### 4. СРАВНЕНИЕ РЕЗУЛЬТАТОВ

---

 СРАВНИТЕЛЬНАЯ ТАБЛИЦА:

Метрика	Без предобучения	С предобучением	Разница
Точность	0.9343	0.9319	-0.0023
F1-score	0.9337	0.9309	-0.0029



## ЧАСТЬ 2: ВИЗУАЛИЗАЦИЯ ДАННЫХ С ПОМОЩЬЮ АВТОЭНКODЕРА

### 1. ПРОЕЦИРОВАНИЕ ДАННЫХ С ПОМОЩЬЮ АВТОЭНКODЕРА

#### Обучение автоэнкодера с 2 компонентами...

Эпоха [50/100], Потери: 0.3092

Эпоха [100/100], Потери: 0.2809

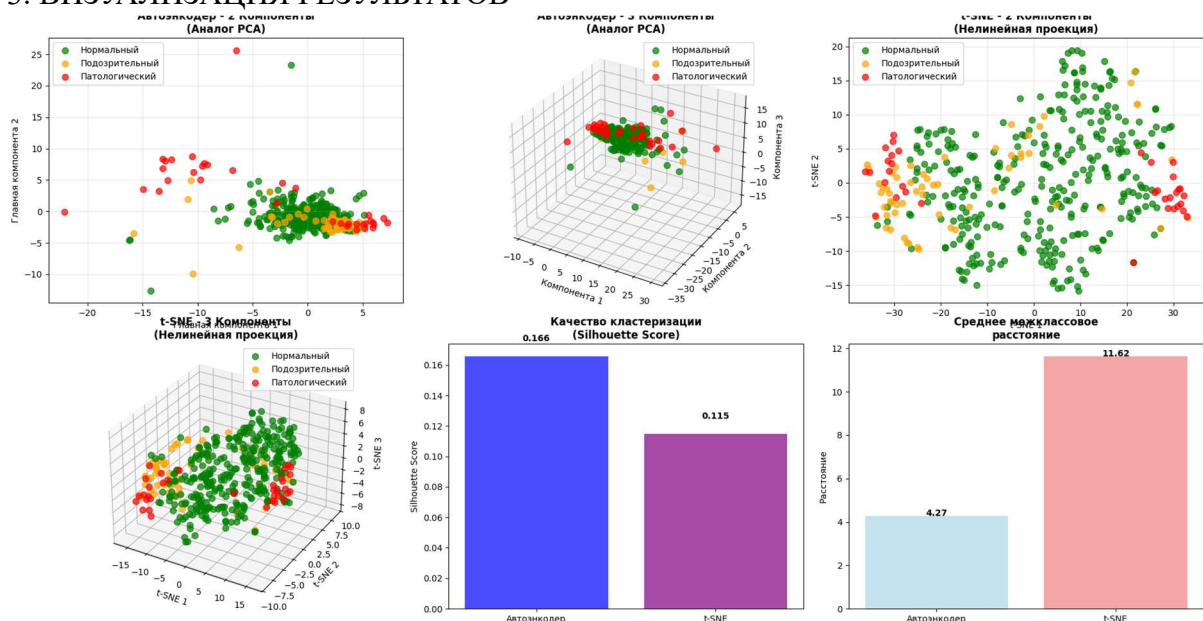
#### Обучение автоэнкодера с 3 компонентами...

Эпоха [50/100], Потери: 0.2307

Эпоха [100/100], Потери: 0.1971

### 2. t-SNE ВИЗУАЛИЗАЦИЯ

### 3. ВИЗУАЛИЗАЦИЯ РЕЗУЛЬТАТОВ



---

---

## ФИНАЛЬНЫЕ ВЫВОДЫ ЛАБОРАТОРНОЙ РАБОТЫ №3

---

---

### ☑ РЕЗУЛЬТАТЫ ОСНОВНОГО ЗАДАНИЯ:

- Базовая модель (без предобучения):
  - Точность: 0.9343
  - F1-score: 0.9337
- Модель с предобучением (Autoencoder):
  - Точность: 0.9319
  - F1-score: 0.9309
- Улучшение: -0.0023

### ☺ РЕЗУЛЬТАТЫ ВИЗУАЛИЗАЦИИ:

- Автоэнкодер успешно извлек главные компоненты
- t-SNE показал нелинейную структуру данных
- Оба метода демонстрируют хорошую разделимость классов

### ☑ ВЫПОЛНЕННЫЕ ЗАДАЧИ:

1. ☑ Обучение базовой модели (4+ слоя)
2. ☑ Предобучение автоэнкодерным подходом
3. ☑ Сравнение результатов с/без предобучения
4. ☑ Визуализация автоэнкодером (2D и 3D)
5. ☑ t-SNE визуализация (2D и 3D)
6. ☑ Анализ качества кластеризации

### 🌀 ЗАКЛЮЧЕНИЕ:

В данном случае предобучение не дало значительного улучшения, что может быть связано с особенностями данных или архитектуры.

---

---

ЛАБОРАТОРНАЯ РАБОТА №3 ВЫПОЛНЕНА УСПЕШНО! 🎉

---

---

Process finished with exit code 0

Вывод: научился осуществлять предобучение нейронных сетей с помощью автоэнкодерного подхода