Министерство образования Республики Беларусь
Учреждение образования
“Брестский государственный технический университет”
Кафедра интеллектуально-информационных технологий

Интеллектуальный анализ данных
Лабораторная работа №3
Предобучение нейронных сетей с использованием автоэнкодерного подхода

Выполнил:
студентка 4 курса
группы ИИ-24
Крупич Д. Д.
Проверила:
Андренко К. В.

Брест-2025

Цель работы: 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 (Wisconsin Diagnostic Breast Cancer (WDBC), класс – 2 признак).
5. Оформить отчет по выполненной работе, загрузить исходный код и отчет в соответствующий репозиторий на github.

№	Выборка	Тип задачи	Целевая переменная
7	https://archive.ics.uci.edu/dataset/503/hepatitis+c+virus+hcv+for+egyptian+patients	классификация	Baselinehistological staging

Код программы:

```
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 LabelEncoder, StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, f1_score, accuracy_score
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
import warnings
warnings.filterwarnings('ignore')

np.random.seed(42)
torch.manual_seed(42)
if torch.cuda.is_available():
```

```
    torch.cuda.manual_seed(42)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Используется устройство: {device}")

def load_mushroom_data():
    """Загрузка и предобработка датасета Mushroom"""
    print("\n" + "="*50)
    print("ЗАГРУЗКА ДАТАСЕТА MUSHROOM")
    print("="*50)

    # Загрузка данных
    url = "https://archive.ics.uci.edu/ml/machine-learning-databases/mushroom/agaricus-lepiota.data"
    columns = ['class', 'cap-shape', 'cap-surface', 'cap-color', 'bruises', 'odor',
               'gill-attachment', 'gill-spacing', 'gill-size', 'gill-color',
               'stalk-shape', 'stalk-root', 'stalk-surface-above-ring',
               'stalk-surface-below-ring', 'stalk-color-above-ring',
               'stalk-color-below-ring', 'veil-type', 'veil-color',
               'ring-number', 'ring-type', 'spore-print-color',
               'population', 'habitat']

    df = pd.read_csv(url, names=columns)
    print(f"Размер датасета: {df.shape}")
    print(f"Количество признаков: {len(columns)-1}")
    print(f"Распределение классов:\n{df['class'].value_counts()}")

    # Кодирование признаков
    X = df.drop('class', axis=1)
    y = df['class']

    # Label encoding для всех признаков
    le_dict = {}
    for col in X.columns:
        le = LabelEncoder()
        X[col] = le.fit_transform(X[col].astype(str))
        le_dict[col] = le

    # Кодирование целевой переменной (e=edible, p=poisonous)
    le_y = LabelEncoder()
    y = le_y.fit_transform(y)

    print(f"Классы: {le_y.classes_} -> {np.unique(y)}")

    return X.values, y

def load_hcv_data():
    print("\n" + "="*50)
    print("ЗАГРУЗКА ДАТАСЕТА HCV")
    print("="*50)

    try:
        print("Попытка загрузки датасета...")
        from ucimlrepo import fetch_ucirepo

        hcv_data = fetch_ucirepo(id=571)

        # Получение данных
        X = hcv_data.data.features
        y = hcv_data.data.targets

        # Объединение и удаление пропусков
        df = pd.concat([X, y], axis=1)
        df = df.dropna()
```

```
    # Разделение обратно
    y_col = y.columns[0]
    X = df.drop(columns=[y_col])
    y = df[y_col].values


except ImportError:
    print("Попытка альтернативного метода загрузки...")

    try:
        import requests
        import io
        import zipfile

        url = "https://archive.ics.uci.edu/static/public/571/hcv+data.zip"
        print(f"Загрузка с {url}...")

        response = requests.get(url, timeout=30)
        response.raise_for_status()

        z = zipfile.ZipFile(io.BytesIO(response.content))
        df = pd.read_csv(z.open('hcvdat0.csv'))

        # Обработка данных
        df = df.dropna()

        # Удаляем ID и берём Category как целевую переменную
        if 'X' in df.columns:
            df = df.drop(columns=['X'])

        y = df['Category'].values
        X = df.drop(columns=['Category']).values

        print("✓ Альтернативная загрузка успешна")

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

        print("Попытка загрузки через прямую ссылку на CSV...")
        try:
            url_csv = "https://archive.ics.uci.edu/ml/machine-learning-databases/00503/hcvdat0.csv"
            df = pd.read_csv(url_csv)
            df = df.dropna()

            if 'X' in df.columns:
                df = df.drop(columns=['X'])

            y = df['Category'].values
            X = df.drop(columns=['Category']).values

            print("✓ Загрузка CSV успешна")
        except Exception as e2:
            print(f"✗ Все методы загрузки не удались: {e2}")
            raise RuntimeError("Не удалось загрузить HCV датасет. Проверьте подключение к интернету.")

except Exception as e:
    print(f"✗ Неожиданная ошибка: {e}")
    raise

print(f"Размер датасета: {X.shape}")
print(f"Количество признаков: {X.shape[1]}")
```

```
    print(f"Количество классов: {len(np.unique(y))}")
    print(f"Распределение классов:\n{pd.Series(y).value_counts()}")
    print("\nПроверка типов данных...")

    if isinstance(X, np.ndarray):
        X = pd.DataFrame(X)

    categorical_columns = X.select_dtypes(include=['object']).columns

    if len(categorical_columns) > 0:
        print(f"Обнаружено {len(categorical_columns)} категориальных признаков")
        print("Выполняется кодирование...")

    for col in categorical_columns:
        le = LabelEncoder()
        X[col] = le.fit_transform(X[col].astype(str))
        print(f"  ✓ {col}: закодирован")

    X = X.astype(float).values
    print(f"✓ Все признаки в числовом формате: {X.dtype}")


    # Кодирование целевой переменной
    le_y = LabelEncoder()
    y = le_y.fit_transform(y)

    print(f"Классы закодированы: {le_y.classes_[:5]}... -> {np.unique(y)}")

    return X, y

class ImprovedAutoencoder(nn.Module):
    def __init__(self, input_dim, hidden_dim):
        super(ImprovedAutoencoder, self).__init__()

        self.encoder = nn.Sequential(
            nn.Linear(input_dim, hidden_dim),
            nn.BatchNorm1d(hidden_dim),
            nn.ReLU(),
            nn.Dropout(0.1)
        )

        self.decoder = nn.Sequential(
            nn.Linear(hidden_dim, input_dim)
        )

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

class ImprovedDeepNN(nn.Module):
    def __init__(self, input_dim, hidden_dims, output_dim, dropout_rate=0.3):
        super(ImprovedDeepNN, self).__init__()

        layers = []
        prev_dim = input_dim

        for i, hidden_dim in enumerate(hidden_dims):
            layers.append(nn.Linear(prev_dim, hidden_dim))
            layers.append(nn.BatchNorm1d(hidden_dim))
            layers.append(nn.ReLU())
            layers.append(nn.Dropout(dropout_rate))
            prev_dim = hidden_dim
```

```
        # Выходной слой
        layers.append(nn.Linear(prev_dim, output_dim))

        self.network = nn.Sequential(*layers)

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

def train_autoencoder(autoencoder, train_loader, epochs=50, lr=0.001, patience=10):
    """
    Обучение автоэнкодера с early stopping

    Параметры:
        autoencoder: модель автоэнкодера
        train_loader: DataLoader с обучающими данными
        epochs: максимальное количество эпох
        lr: learning rate
        patience: количество эпох без улучшения для early stopping
    """
    criterion = nn.MSELoss()
    optimizer = optim.Adam(autoencoder.parameters(), lr=lr, weight_decay=1e-5)

    autoencoder.train()
    best_loss = float('inf')
    patience_counter = 0

    for epoch in range(epochs):
        total_loss = 0
        for batch_x, _ in train_loader:
            batch_x = batch_x.to(device)

            optimizer.zero_grad()
            decoded, _ = autoencoder(batch_x)
            loss = criterion(decoded, batch_x)
            loss.backward()
            optimizer.step()

            total_loss += loss.item()

        avg_loss = total_loss / len(train_loader)

        # Early stopping
        if avg_loss < best_loss:
            best_loss = avg_loss
            patience_counter = 0
        else:
            patience_counter += 1

        if patience_counter >= patience:
            print(f"  Early stopping на эпохе {epoch+1}, Best Loss: {best_loss:.6f}")
            break

        if (epoch + 1) % 10 == 0:
            print(f"  Эпоха {epoch+1}/{epochs}, Loss: {avg_loss:.6f}")

    return autoencoder

def pretrain_layers(X_train, hidden_dims, epochs_per_layer=50):
    """
    Послойное предобучение с использованием улучшенных автоэнкодеров

    Параметры:
```

```
        X_train: обучающие данные (torch.Tensor)
        hidden_dims: список размерностей скрытых слоев
        epochs_per_layer: количество эпох для обучения каждого автоэнкодера

    Возвращает:
        pretrained_weights: список весов для Linear слоев
        pretrained_bn: список параметров для BatchNorm слоев
    """
    print("\n" + "="*50)
    print("ПОСЛОЙНОЕ ПРЕДОБУЧЕНИЕ")
    print("="*50)

    pretrained_weights = []
    pretrained_bn = []
    current_input = X_train.clone()

    for i, hidden_dim in enumerate(hidden_dims):
        print(f"\nПредобучение слоя {i+1}/{len(hidden_dims)}: {current_input.shape[1]} -> {hidden_dim}")

        # Создание автоэнкодера для текущего слоя
        autoencoder = ImprovedAutoencoder(current_input.shape[1], hidden_dim).to(device)

        # Создание DataLoader
        dataset = TensorDataset(current_input, torch.zeros(current_input.shape[0]))
        loader = DataLoader(dataset, batch_size=128, shuffle=True)

        # Обучение автоэнкодера
        autoencoder = train_autoencoder(autoencoder, loader, epochs=epochs_per_layer)

        # Сохранение весов энкодера (Linear слой)
        pretrained_weights.append({
            'weight': autoencoder.encoder[0].weight.data.clone(),
            'bias': autoencoder.encoder[0].bias.data.clone()
        })

        # Сохранение параметров Batch Normalization
        pretrained_bn.append({
            'weight': autoencoder.encoder[1].weight.data.clone(),
            'bias': autoencoder.encoder[1].bias.data.clone(),
            'running_mean': autoencoder.encoder[1].running_mean.clone(),
            'running_var': autoencoder.encoder[1].running_var.clone()
        })

        # Получение выхода энкодера для следующего слоя
        autoencoder.eval()
        with torch.no_grad():
            _, current_input = autoencoder(current_input.to(device))
            current_input = current_input.cpu()

    print(f"\n✓ Предобучение {len(hidden_dims)} слоев завершено")

    return pretrained_weights, pretrained_bn

def initialize_with_pretrained_weights(model, pretrained_weights, pretrained_bn):
    """
    Инициализация модели предобученными весами

    Параметры:
        model: модель для инициализации
        pretrained_weights: веса Linear слоев
        pretrained_bn: параметры BatchNorm слоев
    """
    layer_idx = 0
```

```
    bn_idx = 0

    for module in model.network:
        if isinstance(module, nn.Linear) and layer_idx < len(pretrained_weights):
            # Копируем веса Linear слоя
            module.weight.data = pretrained_weights[layer_idx]['weight'].clone()
            module.bias.data = pretrained_weights[layer_idx]['bias'].clone()
            layer_idx += 1

        elif isinstance(module, nn.BatchNorm1d) and bn_idx < len(pretrained_bn):
            # Копируем параметры BatchNorm слоя
            module.weight.data = pretrained_bn[bn_idx]['weight'].clone()
            module.bias.data = pretrained_bn[bn_idx]['bias'].clone()
            module.running_mean = pretrained_bn[bn_idx]['running_mean'].clone()
            module.running_var = pretrained_bn[bn_idx]['running_var'].clone()
            bn_idx += 1

    print(f"✓ Инициализировано {layer_idx} Linear слоев и {bn_idx} BatchNorm слоев")

def train_model(model, train_loader, val_loader, epochs=150, lr=0.001, patience=15):
    """
    Обучение модели с early stopping и learning rate scheduler

    Параметры:
        model: модель для обучения
        train_loader: DataLoader с обучающими данными
        val_loader: DataLoader с валидационными данными
        epochs: максимальное количество эпох
        lr: начальный learning rate
        patience: количество эпох без улучшения для early stopping
    """
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=lr, weight_decay=1e-5)

    scheduler = optim.lr_scheduler.ReduceLROnPlateau(
        optimizer, mode='max', factor=0.5, patience=5
    )

    train_losses = []
    val_accuracies = []
    best_val_acc = 0
    patience_counter = 0

    for epoch in range(epochs):
        # ========== ОБУЧЕНИЕ ==========
        model.train()
        total_loss = 0

        for batch_x, batch_y in train_loader:
            batch_x, batch_y = batch_x.to(device), batch_y.to(device)

            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
```

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

        val_acc = 100 * correct / total
        avg_loss = total_loss / len(train_loader)

        train_losses.append(avg_loss)
        val_accuracies.append(val_acc)

        scheduler.step(val_acc)

        if val_acc > best_val_acc:
            best_val_acc = val_acc
            patience_counter = 0
        else:
            patience_counter += 1

        if patience_counter >= patience:
            print(f"Early stopping на эпохе {epoch+1}. Лучшая точность: {best_val_acc:.2f}%")
            break

        if (epoch + 1) % 20 == 0:
            current_lr = optimizer.param_groups[0]['lr']
            print(f"Эпоха {epoch+1}/{epochs}, Loss: {avg_loss:.4f}, Val Acc: {val_acc:.2f}%, LR: {current_lr:.6f}")

    return train_losses, val_accuracies

def evaluate_model(model, test_loader):
    """
    Оценка модели на тестовой выборке

    Параметры:
        model: обученная модель
        test_loader: DataLoader с тестовыми данными

    Возвращает:
        dict с метриками и предсказаниями
    """
    model.eval()
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for batch_x, batch_y in test_loader:
            batch_x = batch_x.to(device)
            outputs = model(batch_x)
            _, predicted = torch.max(outputs.data, 1)
            all_preds.extend(predicted.cpu().numpy())
            all_labels.extend(batch_y.numpy())

    # Вычисление метрик
    accuracy = accuracy_score(all_labels, all_preds)
    f1_macro = f1_score(all_labels, all_preds, average='macro', zero_division=0)
    f1_weighted = f1_score(all_labels, all_preds, average='weighted', zero_division=0)

    print(f"\n{'='*50}")
    print(f"РЕЗУЛЬТАТЫ НА ТЕСТОВОЙ ВЫБОРКЕ")
```

```
    print(f"{'='*50}")
    print(f"Accuracy:          {accuracy:.4f} ({accuracy*100:.2f}%)")
    print(f"F1-score (macro):  {f1_macro:.4f}")
    print(f"F1-score (weighted): {f1_weighted:.4f}")

    # Матрица ошибок
    cm = confusion_matrix(all_labels, all_preds)

    return {
        'accuracy': accuracy,
        'f1_macro': f1_macro,
        'f1_weighted': f1_weighted,
        'confusion_matrix': cm,
        'predictions': all_preds,
        'labels': all_labels
    }

def run_experiment(X, y, dataset_name, hidden_dims=[128, 64, 32]):
    """
    Запуск полного эксперимента для датасета

    Параметры:
        X: матрица признаков
        y: целевая переменная
        dataset_name: название датасета для визуализации
        hidden_dims: архитектура скрытых слоев
    """
    print("\n" + "="*70)
    print(f"ЭКСПЕРИМЕНТ: {dataset_name}")
    print("="*70)
    print(f"Архитектура: {hidden_dims}")

    X_temp, X_test, y_temp, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42, stratify=y
    )
    X_train, X_val, y_train, y_val = train_test_split(
        X_temp, y_temp, test_size=0.2, random_state=42, stratify=y_temp
    )

    print(f"\nРазделение данных:")
    print(f"  Train: {X_train.shape[0]} ({X_train.shape[0]/len(X)*100:.1f}%)")
    print(f"  Val:   {X_val.shape[0]} ({X_val.shape[0]/len(X)*100:.1f}%)")
    print(f"  Test:  {X_test.shape[0]} ({X_test.shape[0]/len(X)*100:.1f}%)")

    # Нормализация данных
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_val = scaler.transform(X_val)
    X_test = scaler.transform(X_test)

    print(f"\nПосле нормализации:")
    print(f"  Mean: {X_train.mean():.4f}, Std: {X_train.std():.4f}")
    print(f"  Min: {X_train.min():.4f}, Max: {X_train.max():.4f}")

    # Конвертация в PyTorch тензоры
    X_train_tensor = torch.FloatTensor(X_train)
    y_train_tensor = torch.LongTensor(y_train)
    X_val_tensor = torch.FloatTensor(X_val)
    y_val_tensor = torch.LongTensor(y_val)
    X_test_tensor = torch.FloatTensor(X_test)
    y_test_tensor = torch.LongTensor(y_test)

    # Создание DataLoader
```

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

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=64, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)

input_dim = X_train.shape[1]
output_dim = len(np.unique(y))

print(f"\nПараметры модели:")
print(f"  Input dim: {input_dim}")
print(f"  Hidden dims: {hidden_dims}")
print(f"  Output dim: {output_dim}")

print("\n" + "="*70)
print("ЭКСПЕРИМЕНТ 1: ОБУЧЕНИЕ БЕЗ ПРЕДОБУЧЕНИЯ")
print("="*70)

model_no_pretrain = ImprovedDeepNN(input_dim, hidden_dims, output_dim).to(device)

print(f"Количество параметров: {sum(p.numel() for p in model_no_pretrain.parameters()):,}")

train_losses_no_pretrain, val_acc_no_pretrain = train_model(
    model_no_pretrain, train_loader, val_loader, epochs=150, lr=0.001
)

print("\nОценка на тестовой выборке:")
results_no_pretrain = evaluate_model(model_no_pretrain, test_loader)

print("\n" + "="*70)
print("ЭКСПЕРИМЕНТ 2: ОБУЧЕНИЕ С АВТОЭНКОДЕРНЫМ ПРЕДОБУЧЕНИЕМ")
print("="*70)

# Предобучение слоев
pretrained_weights, pretrained_bn = pretrain_layers(
    X_train_tensor, hidden_dims, epochs_per_layer=50
)

# Создание модели и инициализация предобученными весами
model_pretrain = ImprovedDeepNN(input_dim, hidden_dims, output_dim).to(device)

print("\nИнициализация предобученными весами...")
initialize_with_pretrained_weights(model_pretrain, pretrained_weights, pretrained_bn)

print("\n" + "-"*70)
print("ДООБУЧЕНИЕ МОДЕЛИ")
print("-"*70)

train_losses_pretrain, val_acc_pretrain = train_model(
    model_pretrain, train_loader, val_loader, epochs=150, lr=0.001
)

print("\nОценка на тестовой выборке:")
results_pretrain = evaluate_model(model_pretrain, test_loader)

visualize_results(
    results_no_pretrain, results_pretrain,
    train_losses_no_pretrain, train_losses_pretrain,
    val_acc_no_pretrain, val_acc_pretrain,
    dataset_name
)
```

```
    return results_no_pretrain, results_pretrain

def visualize_results(results_no_pretrain, results_pretrain,
                      train_losses_no_pretrain, train_losses_pretrain,
                      val_acc_no_pretrain, val_acc_pretrain,
                      dataset_name):
    """Визуализация результатов экспериментов"""

    fig, axes = plt.subplots(2, 2, figsize=(16, 12))
    fig.suptitle(f'Результаты для {dataset_name}', fontsize=18, fontweight='bold', y=0.995)

    # График 1: Потери при обучении
    ax1 = axes[0, 0]
    ax1.plot(train_losses_no_pretrain, label='Без предобучения',
             linewidth=2.5, alpha=0.8, color='#FF6B6B')
    ax1.plot(train_losses_pretrain, label='С предобучением',
             linewidth=2.5, alpha=0.8, color='#4ECDC4')
    ax1.set_xlabel('Эпоха', fontsize=13, fontweight='bold')
    ax1.set_ylabel('Loss', fontsize=13, fontweight='bold')
    ax1.set_title('Динамика потерь при обучении', fontsize=14, fontweight='bold', pad=10)
    ax1.legend(fontsize=12, loc='upper right')
    ax1.grid(True, alpha=0.3, linestyle='--')
    ax1.set_xlim(0, max(len(train_losses_no_pretrain), len(train_losses_pretrain)))

    # График 2: Точность на валидации
    ax2 = axes[0, 1]
    ax2.plot(val_acc_no_pretrain, label='Без предобучения',
             linewidth=2.5, alpha=0.8, color='#FF6B6B')
    ax2.plot(val_acc_pretrain, label='С предобучением',
             linewidth=2.5, alpha=0.8, color='#4ECDC4')
    ax2.set_xlabel('Эпоха', fontsize=13, fontweight='bold')
    ax2.set_ylabel('Accuracy (%)', fontsize=13, fontweight='bold')
    ax2.set_title('Точность на валидационной выборке', fontsize=14, fontweight='bold', pad=10)
    ax2.legend(fontsize=12, loc='lower right')
    ax2.grid(True, alpha=0.3, linestyle='--')
    ax2.set_xlim(0, max(len(val_acc_no_pretrain), len(val_acc_pretrain)))

    # График 3: Матрица ошибок (без предобучения)
    ax3 = axes[1, 0]
    sns.heatmap(results_no_pretrain['confusion_matrix'], annot=True, fmt='d',
                cmap='Blues', ax=ax3, cbar_kws={'label': 'Количество'},
                square=True, linewidths=1, linecolor='gray')
    ax3.set_title('Матрица ошибок: БЕЗ предобучения', fontsize=14, fontweight='bold', pad=10)
    ax3.set_ylabel('Истинные метки', fontsize=12, fontweight='bold')
    ax3.set_xlabel('Предсказанные метки', fontsize=12, fontweight='bold')

    # График 4: Матрица ошибок (с предобучением)
    ax4 = axes[1, 1]
    sns.heatmap(results_pretrain['confusion_matrix'], annot=True, fmt='d',
                cmap='Greens', ax=ax4, cbar_kws={'label': 'Количество'},
                square=True, linewidths=1, linecolor='gray')
    ax4.set_title('Матрица ошибок: С предобучением', fontsize=14, fontweight='bold', pad=10)
    ax4.set_ylabel('Истинные метки', fontsize=12, fontweight='bold')
    ax4.set_xlabel('Предсказанные метки', fontsize=12, fontweight='bold')

    plt.tight_layout()
    filename = f'{dataset_name.replace(" ", "_")}_results.png'
    plt.savefig(filename, dpi=300, bbox_inches='tight', facecolor='white')
    print(f"\n✓ График сохранен: {filename}")
    plt.show()

    print("\n" + "="*70)
```

```
print("СРАВНИТЕЛЬНЫЙ АНАЛИЗ МЕТРИК")
print("="*70)

comparison_df = pd.DataFrame({
    'Метрика': ['Accuracy', 'F1-score (macro)', 'F1-score (weighted)'],
    'Без предобучения': [
        results_no_pretrain['accuracy'],
        results_no_pretrain['f1_macro'],
        results_no_pretrain['f1_weighted']
    ],
    'С предобучением': [
        results_pretrain['accuracy'],
        results_pretrain['f1_macro'],
        results_pretrain['f1_weighted']
    ]
})

comparison_df['Разница'] = (
    comparison_df['С предобучением'] - comparison_df['Без предобучения']
)
comparison_df['Улучшение (%)'] = (
    comparison_df['Разница'] / comparison_df['Без предобучения'] * 100
).round(2)

print(comparison_df.to_string(index=False))

print("\n" + "="*70)
print("ДОПОЛНИТЕЛЬНЫЙ АНАЛИЗ СХОДИМОСТИ")
print("="*70)

max_val_acc_no_pretrain = max(val_acc_no_pretrain)
max_val_acc_pretrain = max(val_acc_pretrain)

print(f"\nМаксимальная валидационная точность:")
print(f"  Без предобучения: {max_val_acc_no_pretrain:.2f}%")
print(f"  С предобучением:  {max_val_acc_pretrain:.2f}%")
print(f"  Разница: {max_val_acc_pretrain - max_val_acc_no_pretrain:+.2f}%")

# Анализ скорости сходимости (эпохи до 95% от макс. точности)
threshold_no_pretrain = 0.95 * max_val_acc_no_pretrain
threshold_pretrain = 0.95 * max_val_acc_pretrain

epochs_to_95_no_pretrain = next(
    (i for i, x in enumerate(val_acc_no_pretrain) if x >= threshold_no_pretrain),
    len(val_acc_no_pretrain)
)
epochs_to_95_pretrain = next(
    (i for i, x in enumerate(val_acc_pretrain) if x >= threshold_pretrain),
    len(val_acc_pretrain)
)

print(f"\nСкорость сходимости (эпохи до 95% от макс. точности):")
print(f"  Без предобучения: {epochs_to_95_no_pretrain} эпох")
print(f"  С предобучением:  {epochs_to_95_pretrain} эпох")

if epochs_to_95_no_pretrain > 0:
    speedup = (epochs_to_95_no_pretrain - epochs_to_95_pretrain) / epochs_to_95_no_pretrain * 100
    print(f"  Ускорение: {speedup:.1f}%")

print(f"\nФинальное количество эпох обучения:")
print(f"  Без предобучения: {len(val_acc_no_pretrain)} эпох")
print(f"  С предобучением:  {len(val_acc_pretrain)} эпох")
```

```
    # Анализ стабильности
    std_no_pretrain = np.std(val_acc_no_pretrain[-10:])  # std последних 10 эпох
    std_pretrain = np.std(val_acc_pretrain[-10:])

    print(f"\nСтабильность (std последних 10 эпох):")
    print(f"  Без предобучения: {std_no_pretrain:.2f}%")
    print(f"  С предобучением:  {std_pretrain:.2f}%")

def main():
    """Главная функция для запуска всех экспериментов"""

    print("\n" + "="*70)
    print("Предобучение нейронных сетей с использованием автоэнкодерного подхода")
    print("="*70)
    print(f"\nПараметры эксперимента:")
    print(f"  Device: {device}")
    print(f"  PyTorch version: {torch.__version__}")
    print(f"  Random seed: 42")

    # Архитектура сети
    hidden_dims = [128, 64, 32]
    print(f"  Архитектура: {hidden_dims}")

    # ================================
    # ЭКСПЕРИМЕНТ 1: MUSHROOM DATASET
    # ================================
    print("\n" + "#"*70)
    print("# ЭКСПЕРИМЕНТ 1: MUSHROOM DATASET")
    print("#"*70)

    try:
        X_mushroom, y_mushroom = load_mushroom_data()
        results_mushroom = run_experiment(
            X_mushroom, y_mushroom,
            "Mushroom Dataset",
            hidden_dims=hidden_dims
        )
    except Exception as e:
        print(f"\n✗ Ошибка при работе с Mushroom dataset: {e}")
        import traceback
        traceback.print_exc()

    # ================================
    # ЭКСПЕРИМЕНТ 2: HCV DATASET
    # ================================
    print("\n" + "#"*70)
    print("# ЭКСПЕРИМЕНТ 2: HCV DATASET")
    print("#"*70)

    try:
        X_hcv, y_hcv = load_hcv_data()
        results_hcv = run_experiment(
            X_hcv, y_hcv,
            "HCV Dataset",
            hidden_dims=hidden_dims
        )
    except Exception as e:
        print(f"\n✗ Ошибка при работе с HCV dataset: {e}")
        import traceback
        traceback.print_exc()

if __name__ == "__main__":
    main()
```

Вывод программы:

```
Предобучение нейронных сетей с использованием автоэнкодерного подхода
================================================================================

Параметры эксперимента:
  Device: cuda
  PyTorch version: 2.8.0+cu128
  Random seed: 42
  Архитектура: [128, 64, 32]

################################################################################
# ЭКСПЕРИМЕНТ 1: MUSHROOM DATASET
################################################################################

================================================
ЗАГРУЗКА ДАТАСЕТА MUSHROOM
================================================
Размер датасета: (8124, 23)
Количество признаков: 22
Распределение классов:
class
e    4208
p    3916
Name: count, dtype: int64
Классы: ['e' 'p'] -> [0 1]

================================================================================
ЭКСПЕРИМЕНТ: Mushroom Dataset
================================================================================
Архитектура: [128, 64, 32]

Разделение данных:
  Train: 5199 (64.0%)
  Val:   1300 (16.0%)
  Test:  1625 (20.0%)

После нормализации:
  Mean: 0.0000, Std: 0.9770
  Min: -8.4699, Max: 4.4411
```

Параметры модели:
  Input dim: 22
  Hidden dims: [128, 64, 32]
  Output dim: 2

==========
====
ЭКСПЕРИМЕНТ 1: ОБУЧЕНИЕ БЕЗ ПРЕДОБУЧЕНИЯ
==========
====
Количество параметров: 13,794
Early stopping на эпохе 20. Лучшая точность: 100.00%

Оценка на тестовой выборке:

==========
РЕЗУЛЬТАТЫ НА ТЕСТОВОЙ ВЫБОРКЕ
==========
Accuracy: 1.0000 (100.00%)
F1-score (macro): 1.0000
F1-score (weighted): 1.0000

==========
====
ЭКСПЕРИМЕНТ 2: ОБУЧЕНИЕ С АВТОЭНКОДЕРНЫМ ПРЕДОБУЧЕНИЕМ
==========
====

==========
ПОСЛОЙНОЕ ПРЕДОБУЧЕНИЕ
==========

Предобучение слоя 1/3: 22 -> 128
  Эпоха 10/50, Loss: 0.048571
  Эпоха 20/50, Loss: 0.040887
  Эпоха 30/50, Loss: 0.036617
  Эпоха 40/50, Loss: 0.036570
  Эпоха 50/50, Loss: 0.036620

Предобучение слоя 2/3: 128 -> 64
  Эпоха 10/50, Loss: 0.065110

Эпоха 20/50, Loss: 0.052068
Эпоха 30/50, Loss: 0.049438
Эпоха 40/50, Loss: 0.047945
Эпоха 50/50, Loss: 0.047075

Предобучение слоя 3/3: 64 -> 32
Эпоха 10/50, Loss: 0.142895
Эпоха 20/50, Loss: 0.121342
Эпоха 30/50, Loss: 0.116687
Эпоха 40/50, Loss: 0.114908
Эпоха 50/50, Loss: 0.110266

✓ Предобучение 3 слоев завершено

Инициализация предобученными весами...
✓ Инициализировано 3 Linear слоев и 3 BatchNorm слоев

ДООБУЧЕНИЕ МОДЕЛИ

Early stopping на эпохе 18. Лучшая точность: 100.00%

Оценка на тестовой выборке:

==========
РЕЗУЛЬТАТЫ НА ТЕСТОВОЙ ВЫБОРКЕ
==========
Accuracy: 1.0000 (100.00%)
F1-score (macro): 1.0000
F1-score (weighted): 1.0000

✓ График сохранен: Mushroom_Dataset_results.png

==========
СРАВНИТЕЛЬНЫЙ АНАЛИЗ МЕТРИК
==========

Метрика	Без предобучения	С предобучением	Разница	Улучшение (%)
Accuracy	1.0	1.0	0.0	0.0
F1-score (macro)	1.0	1.0	0.0	0.0

F1-score (weighted) 1.0 1.0 0.0 0.0

==========
====
ДОПОЛНИТЕЛЬНЫЙ АНАЛИЗ СХОДИМОСТИ
==========
====

Максимальная валидационная точность:
  Без предобучения: 100.00%
  С предобучением: 100.00%
  Разница: +0.00%

Скорость сходимости (эпохи до 95% от макс. точности):
  Без предобучения: 0 эпох
  С предобучением: 0 эпох

Финальное количество эпох обучения:
  Без предобучения: 20 эпох
  С предобучением: 18 эпох

Стабильность (std последних 10 эпох):
  Без предобучения: 0.00%
  С предобучением: 0.00%

##########
ЭКСПЕРИМЕНТ 2: HCV DATASET
##########

==========
ЗАГРУЗКА ДАТАСЕТА HCV
==========
Попытка загрузки через ucimlrepo...
✓ Загрузка через ucimlrepo успешна
Размер датасета: (589, 12)
Количество признаков: 12
Количество классов: 5
Распределение классов:
0=Blood Donor 526
3=Cirrhosis 24
1=Hepatitis 20
2=Fibrosis 12

```
0s=suspect Blood Donor      7
Name: count, dtype: int64

Проверка типов данных...
Обнаружено 1 категориальных признаков
Выполняется кодирование...
  ✓ Sex: закодирован
✓ Все признаки в числовом формате: float64
Классы закодированы: ['0=Blood Donor' '0s=suspect Blood Donor' '1=Hepatitis' '2=Fibrosis'
 '3=Cirrhosis']... -> [0 1 2 3 4]

================================================================
====
ЭКСПЕРИМЕНТ: HCV Dataset
================================================================
====
Архитектура: [128, 64, 32]

Разделение данных:
  Train: 376 (63.8%)
  Val:   95 (16.1%)
  Test:  118 (20.0%)

После нормализации:
  Mean: 0.0000, Std: 1.0000
  Min: -4.6312, Max: 13.2610

Параметры модели:
  Input dim: 12
  Hidden dims: [128, 64, 32]
  Output dim: 5

================================================================
====
ЭКСПЕРИМЕНТ 1: ОБУЧЕНИЕ БЕЗ ПРЕДОБУЧЕНИЯ
================================================================
====
Количество параметров: 12,613
Early stopping на эпохе 19. Лучшая точность: 95.79%

Оценка на тестовой выборке:
```

```
==================================================
РЕЗУЛЬТАТЫ НА ТЕСТОВОЙ ВЫБОРКЕ
==================================================
Accuracy:          0.9322 (93.22%)
F1-score (macro):  0.3927
F1-score (weighted): 0.8995

================================================================================
ЭКСПЕРИМЕНТ 2: ОБУЧЕНИЕ С АВТОЭНКОДЕРНЫМ ПРЕДОБУЧЕНИЕМ
================================================================================

==================================================
ПОСЛОЙНОЕ ПРЕДОБУЧЕНИЕ
==================================================

Предобучение слоя 1/3: 12 -> 128
  Эпоха 10/50, Loss: 0.380731
  Эпоха 20/50, Loss: 0.131464
  Эпоха 30/50, Loss: 0.077708
  Эпоха 40/50, Loss: 0.067872
  Эпоха 50/50, Loss: 0.069748

Предобучение слоя 2/3: 128 -> 64
  Эпоха 10/50, Loss: 0.290975
  Эпоха 20/50, Loss: 0.177111
  Эпоха 30/50, Loss: 0.123981
  Эпоха 40/50, Loss: 0.100545
  Эпоха 50/50, Loss: 0.084296

Предобучение слоя 3/3: 64 -> 32
  Эпоха 10/50, Loss: 0.456166
  Эпоха 20/50, Loss: 0.305462
  Эпоха 30/50, Loss: 0.233468
  Эпоха 40/50, Loss: 0.179297
  Эпоха 50/50, Loss: 0.157820

✓ Предобучение 3 слоев завершено

Инициализация предобученными весами...
```

✓ Инициализировано 3 Linear слоев и 3 BatchNorm слоев

ДООБУЧЕНИЕ МОДЕЛИ

Early stopping на эпохе 19. Лучшая точность: 94.74%

Оценка на тестовой выборке:

==========
РЕЗУЛЬТАТЫ НА ТЕСТОВОЙ ВЫБОРКЕ
==========
Accuracy: 0.9237 (92.37%)
F1-score (macro): 0.3704
F1-score (weighted): 0.8948

✓ График сохранен: HCV_Dataset_results.png

==========
СРАВНИТЕЛЬНЫЙ АНАЛИЗ МЕТРИК
==========

Метрика	Без предобучения	С предобучением	Разница	Улучшение (%)
Accuracy	0.932203	0.923729	-0.008475	-0.91
F1-score (macro)	0.392661	0.370438	-0.022222	-5.66
F1-score (weighted)	0.899549	0.894841	-0.004708	-0.52

==========
ДОПОЛНИТЕЛЬНЫЙ АНАЛИЗ СХОДИМОСТИ
==========

Максимальная валидационная точность:
Без предобучения: 95.79%
С предобучением: 94.74%
Разница: -1.05%

Скорость сходимости (эпохи до 95% от макс. точности):
Без предобучения: 2 эпох

С предобучением: 3 эпох
Ускорение: -50.0%

Финальное количество эпох обучения:
Без предобучения: 19 эпох
С предобучением: 19 эпох

Стабильность (std последних 10 эпох):
Без предобучения: 0.52%
С предобучением: 0.48%

📊 MUSHROOM DATASET:

✓ Обе модели достигли идеальной точности: 100% accuracy
✓ Предобучение сократило обучение: 20 → 18 эпох (экономия 10%)
⚠ Эффект минимален из-за простоты задачи

📊 HCV DATASET:

❌ Предобучение ухудшило результаты:
- Accuracy: 93.22% → 92.37% (снижение на 0.85%)
- F1-macro: 39.27% → 37.04% (снижение на 5.66%)

✓ Стабильность: незначительное улучшение (0.52% → 0.48% std)
⚠ Малый размер датасета (589 примеров) + дисбаланс классов

Вывод: научился осуществлять предобучение нейронных сетей с помощью автоэнкодерного подхода.