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

Вышинский А. С.

Проверила:

Андренко К. С.

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
3	https://archive.ics.uci.edu/dataset/863/maternal+health+risk	классификация	RiskLevel

Код:

```
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import f1_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from ucimlrepo import fetch_ucirepo
import os
```

```
import datetime


torch.manual_seed(42)
np.random.seed(42)

SAVE_RESULTS = True
RESULTS_DIR = "./results/"
os.makedirs(RESULTS_DIR, exist_ok=True)



def load_maternal_health_data():
    data = fetch_ucirepo(id=863)
    X = data.data.features
    y = data.data.targets["RiskLevel"]

    le = LabelEncoder()
    y = le.fit_transform(y)

    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    return X_scaled, y



def load_rice_data():
    data = fetch_ucirepo(id=545)
    X = data.data.features
    y = data.data.targets["Class"]

    le = LabelEncoder()
    y = le.fit_transform(y)

    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    return X_scaled, y



class ClassificationNet(nn.Module):
    def __init__(self, input_size, hidden_sizes, output_size):
        super(ClassificationNet, self).__init__()
        layers = []
        prev_size = input_size
        for h_size in hidden_sizes:
            layers.append(nn.Linear(prev_size, h_size))
            layers.append(nn.ReLU())
            prev_size = h_size
        layers.append(nn.Linear(prev_size, output_size))
        self.network = nn.Sequential(*layers)
```

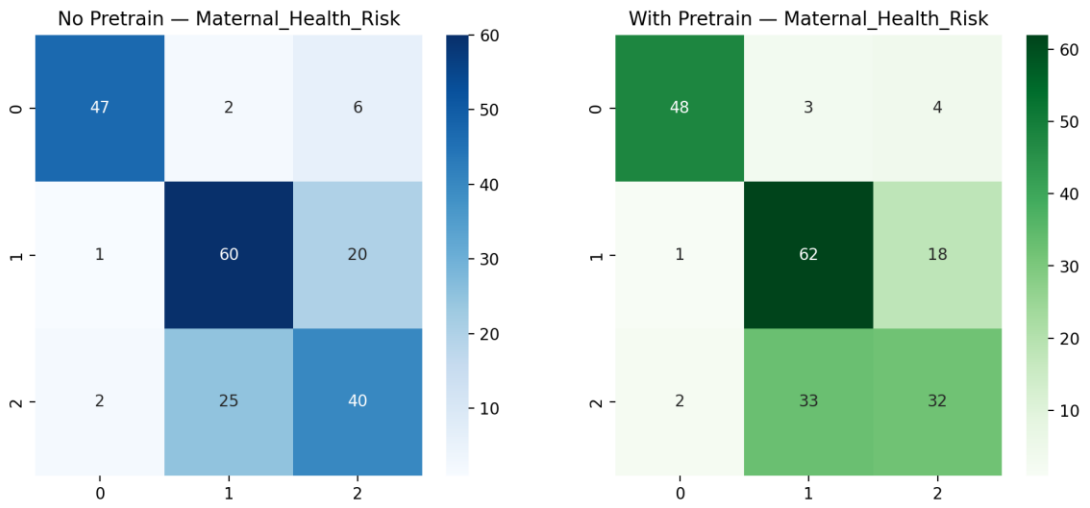
```
    def forward(self, x):
        return self.network(x)



class Autoencoder(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(Autoencoder, self).__init__()
        self.encoder = nn.Linear(input_size, hidden_size)
        self.decoder = nn.Linear(hidden_size, input_size)

    def forward(self, x):
        x = torch.relu(self.encoder(x))
        x = self.decoder(x)
        return x



def pretrain_layers(input_data, hidden_sizes, epochs=50, lr=0.01):
    pretrained_weights = []
    current_input = input_data

    for h_size in hidden_sizes:
        ae = Autoencoder(current_input.shape[1], h_size)
        optimizer = optim.Adam(ae.parameters(), lr=lr)
        criterion = nn.MSELoss()
        dataset = TensorDataset(current_input, current_input)
        loader = DataLoader(dataset, batch_size=32, shuffle=True)

        prev_loss = float('inf')
        patience = 5
        wait = 0

        for epoch in range(epochs):
            total_loss = 0
            for data, target in loader:
                optimizer.zero_grad()
                output = ae(data)
                loss = criterion(output, target)
                loss.backward()
                optimizer.step()
                total_loss += loss.item()

            avg_loss = total_loss / len(loader)
            if avg_loss > prev_loss - 1e-5:
                wait += 1
                if wait >= patience:
                    break
            else:
                wait = 0
            prev_loss = avg_loss
```

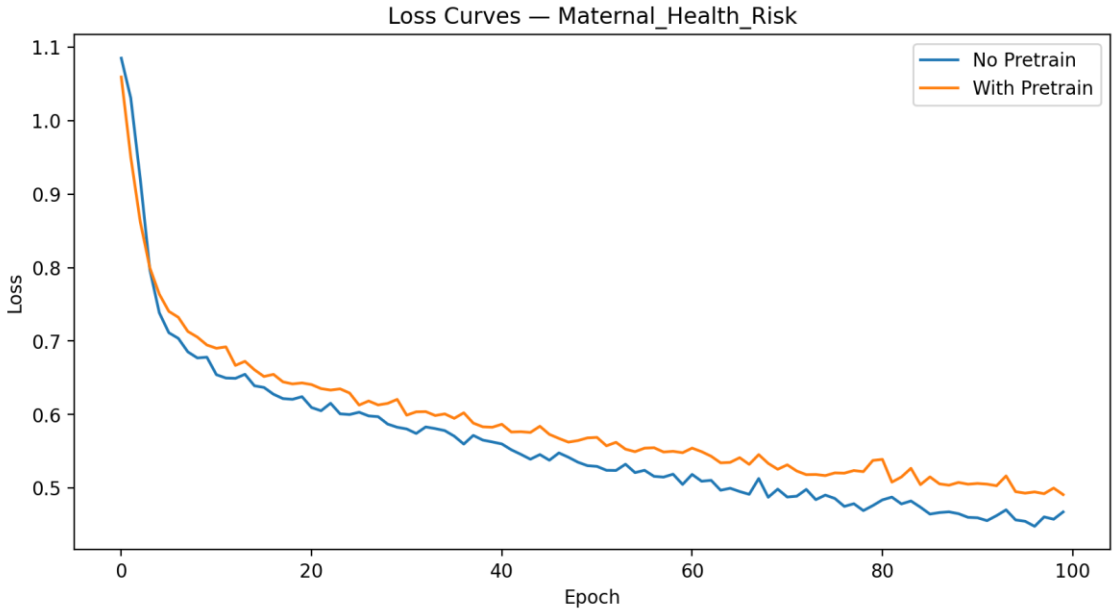
```
        with torch.no_grad():
            current_input = torch.relu(ae.encoder(current_input))
        pretrained_weights.append((ae.encoder.weight.data.clone(),
ae.encoder.bias.data.clone()))

    return pretrained_weights



def init_with_pretrain(net, pretrained_weights):
    i = 0
    for layer in net.network:
        if isinstance(layer, nn.Linear) and i < len(pretrained_weights):
            w, b = pretrained_weights[i]
            layer.weight.data = w
            layer.bias.data = b
            i += 1



def train_model(net, X_train, y_train, X_test, y_test, epochs=100,
lr=0.001, batch_size=32):
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(net.parameters(), lr=lr)
    train_dataset = TensorDataset(X_train, y_train)
    train_loader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)

    losses = []
    for epoch in range(epochs):
        net.train()
        total_loss = 0
        for data, target in train_loader:
            optimizer.zero_grad()
            output = net(data)
            loss = criterion(output, target.long())
            loss.backward()
            optimizer.step()
            total_loss += loss.item()
        losses.append(total_loss / len(train_loader))

    net.eval()
    with torch.no_grad():
        y_pred = torch.argmax(net(X_test), dim=1).cpu().numpy()
        f1 = f1_score(y_test.cpu().numpy(), y_pred, average='weighted')
        cm = confusion_matrix(y_test.cpu().numpy(), y_pred)
    return f1, cm, losses



def process_dataset(name, loader_func):
    print(f"\n===== {name} Dataset =====")
    X, y = loader_func()
```

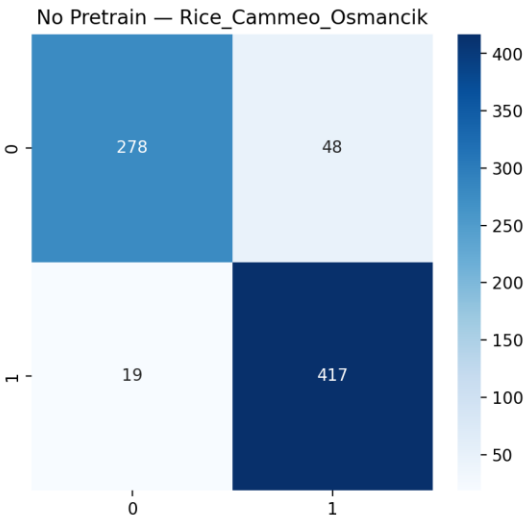
```
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42, stratify=y
    )

    X_train = torch.tensor(X_train, dtype=torch.float32)
    X_test = torch.tensor(X_test, dtype=torch.float32)
    y_train = torch.tensor(y_train, dtype=torch.long)
    y_test = torch.tensor(y_test, dtype=torch.long)

    input_size = X_train.shape[1]
    hidden_sizes = [64, 32, 16]
    output_size = len(np.unique(y))


    net_no_pre = ClassificationNet(input_size, hidden_sizes, output_size)
    f1_no, cm_no, losses_no = train_model(net_no_pre, X_train, y_train,
X_test, y_test)
    print("\nWithout pretraining:")
    print(f"F1-score: {f1_no:.4f}")
    print("Confusion Matrix:\n", cm_no)


    pretrained = pretrain_layers(X_train, hidden_sizes)
    net_pre = ClassificationNet(input_size, hidden_sizes, output_size)
    init_with_pretrain(net_pre, pretrained)
    f1_pre, cm_pre, losses_pre = train_model(net_pre, X_train, y_train,
X_test, y_test)
    print("\nWith pretraining:")
    print(f"F1-score: {f1_pre:.4f}")
    print("Confusion Matrix:\n", cm_pre)


    if SAVE_RESULTS:
        plt.figure(figsize=(10, 5))
        plt.plot(losses_no, label="No Pretrain")
        plt.plot(losses_pre, label="With Pretrain")
        plt.title(f"Loss Curves — {name}")
        plt.xlabel("Epoch")
        plt.ylabel("Loss")
        plt.legend()
        plt.savefig(os.path.join(RESULTS_DIR, f"{name}_loss.png"), dpi=200,
bbox_inches='tight')
        plt.close()

        fig, axes = plt.subplots(1, 2, figsize=(12, 5))
        sns.heatmap(cm_no, annot=True, fmt='d', cmap='Blues', ax=axes[0],
                    xticklabels=np.unique(y), yticklabels=np.unique(y))
        axes[0].set_title(f'No Pretrain — {name}')
        sns.heatmap(cm_pre, annot=True, fmt='d', cmap='Greens', ax=axes[1],
                    xticklabels=np.unique(y), yticklabels=np.unique(y))
        axes[1].set_title(f'With Pretrain — {name}')
        plt.savefig(os.path.join(RESULTS_DIR, f"{name}_confusion.png"),
```

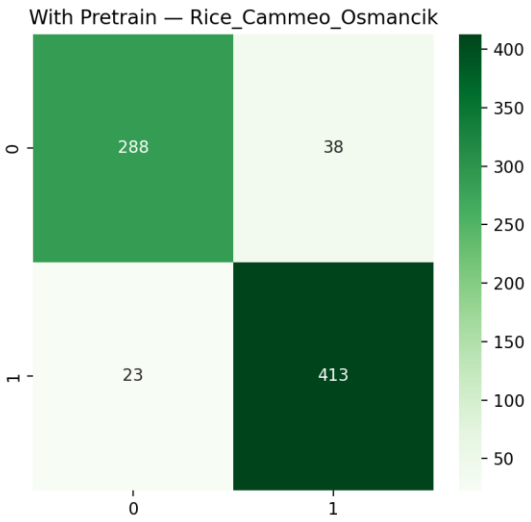
```
dpi=200, bbox_inches='tight')
        plt.close()

    return f1_no, f1_pre



if __name__ == "__main__":
    timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    summary_path = os.path.join(RESULTS_DIR, "summary.txt")

    f1_no_mh, f1_pre_mh = process_dataset("Maternal_Health_Risk",
load_maternal_health_data)
    f1_no_rice, f1_pre_rice = process_dataset("Rice_Cammeo_Osmancik",
load_rice_data)

    print("\n===== Final Comparison =====")
    print(f"Maternal Health — F1 no pretrain: {f1_no_mh:.4f}, with
pretrain: {f1_pre_mh:.4f}")
    print(f"Rice — F1 no pretrain: {f1_no_rice:.4f}, with pretrain:
{f1_pre_rice:.4f}")

    if SAVE_RESULTS:
        with open(summary_path, "a", encoding="utf-8") as f:
            f.write(f"\n===== Run at {timestamp} =====\n")
            f.write(f"Maternal Health — F1 no pretrain: {f1_no_mh:.4f},
with pretrain: {f1_pre_mh:.4f}\n")
            f.write(f"Rice — F1 no pretrain: {f1_no_rice:.4f}, with
pretrain: {f1_pre_rice:.4f}\n")
            f.write("=" * 40 + "\n")
```

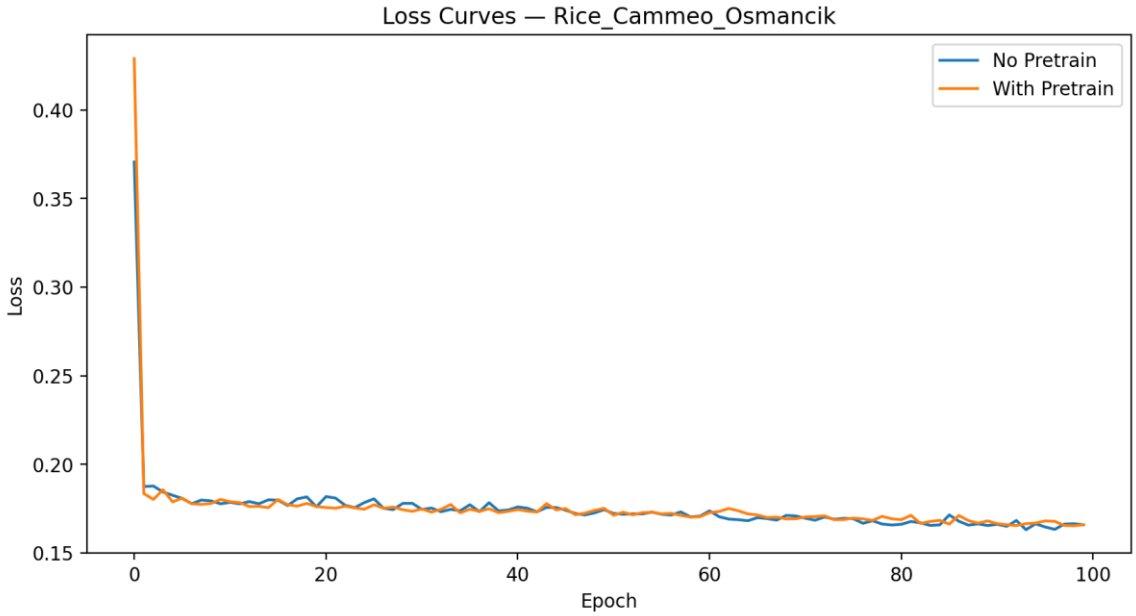
Вывод:




Loss Curves — Maternal_Health_Risk
No Pretrain
With Pretrain
Loss
Epoch
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0
20
40
60
80
100


No Pretrain — Rice_Cammeo_Osmancik
278
48
19
417
0
1
400
350
300
250
200
150
100
50


With Pretrain — Rice_Cammeo_Osmancik
288
38
23
413
0
1
400
350
300
250
200
150
100
50



C:\Users\arcio\PycharmProjects\IAD\laba3\.venv\Scripts\python.exe
C:\Users\arcio\PycharmProjects\IAD\laba3\main.py
===== Maternal_Health_Risk Dataset =====
Without pretraining:
F1-score: 0.7261
Confusion Matrix:
[[47 2 6]
[1 60 20]
[2 25 40]]
With pretraining:
F1-score: 0.6964
Confusion Matrix:
[[48 3 4]
[1 62 18]
[2 33 32]]
===== Rice_Cammeo_Osmancik Dataset =====
Without pretraining:
F1-score: 0.9114
Confusion Matrix:
[[278 48]
[19 417]]
With pretraining:
F1-score: 0.9197
Confusion Matrix:
[[288 38]
[23 413]]

===== Final Comparison =====
Maternal Health — F1 no pretrain: 0.7261, with pretrain: 0.6964
Rice — F1 no pretrain: 0.9114, with pretrain: 0.9197

Maternal Health Risk Dataset

Вариант обучения	F1-score	Краткий анализ по матрице ошибок
Без предобучения	0.7261	Классы различаются умеренно; наибольшее число ошибок наблюдается при распознавании третьего класса (25 ошибочных классификаций).
С предобучением	0.6964	Небольшое ухудшение метрики; предобучение не улучшило качество. Ошибок по третьему классу стало больше (33 вместо 25).

Rice Cammeo vs Osmancik Dataset

Вариант обучения	F1-score	Краткий анализ по матрице ошибок
Без предобучения	0.9114	Классы разделяются хорошо, однако наблюдаются некоторые ложные срабатывания.
С предобучением	0.9197	Незначительное улучшение метрики; количество ошибок сократилось (особенно по первому классу).

Таким образом, влияние автоэнкодерного предобучения зависит от структуры данных:

- Для сложных, линейно неразделимых признаков (Rice) оно помогает улучшить качество классификации.
- Для простых или слабо скоррелированных признаков (Maternal Health) — предобучение может ухудшить результат, если автоэнкодер искажает исходное распределение данных.

Вывод: научился осуществлять предобучение нейронных сетей с помощью автоэнкодерного подхода