`

Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Отчет по лабораторной работе 3

Специальность ИИ-23

Выполнил:

Макаревич Н.Р.

Студент группы ИИ-23

Проверил:

Андренко К. В.
Преподаватель-стажёр
Кафедры ИИТ,
«___» ___________ 2025 г.

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.

Код программы:

```
# -*- coding: utf-8 -*-
"""iad_3.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1NbhINh_nwV_Emt9h_uLQDu0YH4HbFS5x
"""

!pip install ucimlrepo --quiet
!pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cpu -q
```

`

```
!pip install scikit-learn pandas numpy matplotlib seaborn --quiet


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


RND = 42
random.seed(RND)
np.random.seed(RND)
torch.manual_seed(RND)
device = torch.device("cpu")


from ucimlrepo import fetch_ucirepo


ds = fetch_ucirepo(id=863)
X = ds.data.features
y = ds.data.targets
```

`

```
if not isinstance(X, pd.DataFrame):
    X = pd.DataFrame(X)
if isinstance(y, (pd.DataFrame, pd.Series)):
    y = np.array(y).reshape(-1,)
else:
    y = np.array(y)

le = LabelEncoder()
y_enc = le.fit_transform(y)
class_names = le.classes_
n_classes = len(class_names)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X.values.astype(float))

X_train, X_temp, y_train, y_temp = train_test_split(
    X_scaled, y_enc, test_size=0.4, random_state=RND, stratify=y_enc
)
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, random_state=RND, stratify=y_temp
)

def to_loader(X, y, batch_size=32, shuffle=True):
    X_t = torch.tensor(X, dtype=torch.float32)
    y_t = torch.tensor(y, dtype=torch.long)
```

`

```
    ds = TensorDataset(X_t, y_t)
    return DataLoader(ds, batch_size=batch_size, shuffle=shuffle)

train_loader = to_loader(X_train, y_train, batch_size=32, shuffle=True)
val_loader = to_loader(X_val, y_val, batch_size=64, shuffle=False)
test_loader = to_loader(X_test, y_test, batch_size=64, shuffle=False)

def evaluate_model(model, loader):
    model.eval()
    ys = []
    ys_pred = []
    with torch.no_grad():
        for xb, yb in loader:
            xb = xb.to(device)
            out = model(xb)
            preds = out.argmax(dim=1).cpu().numpy()
            ys_pred.extend(preds.tolist())
            ys.extend(yb.numpy().tolist())
    return np.array(ys), np.array(ys_pred)

def print_metrics(y_true, y_pred, labels=None):
    print("Accuracy:", accuracy_score(y_true, y_pred))
    print("Macro F1: ", f1_score(y_true, y_pred, average='macro'))
    print("\nClassification report:\n", classification_report(y_true, y_pred, 
target_names=labels))
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(6,5))
```

`

```
    sns.heatmap(cm, annot=True, fmt='d', xticklabels=labels, yticklabels=labels,
cmap='Blues')
    plt.xlabel("Predicted")
    plt.ylabel("True")
    plt.show()


input_dim = X_train.shape[1]
hidden_sizes = [64, 32, 16]
dropout = 0.2


class FCNet(nn.Module):
    def __init__(self, input_dim, hidden_sizes, n_classes, dropout=0.2):
        super().__init__()
        layers = []
        prev = input_dim
        for h in hidden_sizes:
            layers.append(nn.Linear(prev, h))
            layers.append(nn.ReLU())
            layers.append(nn.Dropout(dropout))
            prev = h
        layers.append(nn.Linear(prev, n_classes))
        self.net = nn.Sequential(*layers)
    def forward(self, x):
        return self.net(x)


model_plain = FCNet(input_dim, hidden_sizes, n_classes, dropout=dropout).to(device)
print(model_plain)
```

`

```
def train_classifier(model, train_loader, val_loader, epochs=100, lr=1e-3, weight_decay=1e-5, patience=10):
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=lr, weight_decay=weight_decay)
    best_val_loss = float('inf')
    best_state = None
    patience_cnt = 0
    history = {'train_loss':[], 'val_loss':[]}
    for ep in range(1, epochs+1):
        model.train()
        total_loss = 0.0
        for xb, yb in train_loader:
            xb, yb = xb.to(device), yb.to(device)
            optimizer.zero_grad()
            out = model(xb)
            loss = criterion(out, yb)
            loss.backward()
            optimizer.step()
            total_loss += loss.item() * xb.size(0)
        avg_train_loss = total_loss / len(train_loader.dataset)
        model.eval()
        total_val_loss = 0.0
        with torch.no_grad():
            for xb, yb in val_loader:
                xb, yb = xb.to(device), yb.to(device)
                out = model(xb)
```

```
`
                loss = criterion(out, yb)
                total_val_loss += loss.item() * xb.size(0)
        avg_val_loss = total_val_loss / len(val_loader.dataset)
        history['train_loss'].append(avg_train_loss)
        history['val_loss'].append(avg_val_loss)

        if avg_val_loss < best_val_loss - 1e-5:
            best_val_loss = avg_val_loss
            best_state = {k:v.cpu() for k,v in model.state_dict().items()}
            patience_cnt = 0
        else:
            patience_cnt += 1
        if patience_cnt >= patience:
            print(f"Early stopping at epoch {ep}")
            break
        if ep % 10 == 0 or ep==1:
            print(f"Epoch {ep} train_loss={avg_train_loss:.4f} val_loss={avg_val_loss:.4f}")

    if best_state is not None:
        model.load_state_dict(best_state)
    return model, history


model_plain_trained, hist_plain = train_classifier(model_plain, train_loader, val_loader,
epochs=200, lr=1e-3, patience=15)


y_true, y_pred = evaluate_model(model_plain_trained, test_loader)
print_metrics(y_true, y_pred, labels=class_names)
```

`

```
ae_epochs = 80
ae_lr = 1e-3
ae_batch = 32

class SimpleAE(nn.Module):
    def __init__(self, input_dim, bottleneck_dim):
        super().__init__()
        self.enc = nn.Sequential(nn.Linear(input_dim, bottleneck_dim), nn.ReLU())
        self.dec = nn.Sequential(nn.Linear(bottleneck_dim, input_dim))
    def forward(self, x):
        z = self.enc(x)
        xr = self.dec(z)
        return xr, z

def train_autoencoder(ae, data_X, epochs=50, batch_size=32, lr=1e-3, verbose=False):
    ae = ae.to(device)
    opt = optim.Adam(ae.parameters(), lr=lr)
    criterion = nn.MSELoss()
    loader = to_loader(data_X, np.zeros(len(data_X)), batch_size=batch_size, shuffle=True)
    for ep in range(1, epochs+1):
        ae.train()
        total = 0.0
        for xb, _ in loader:
            xb = xb.to(device)
            xr, _ = ae(xb)
```

`

```
        loss = criterion(xr, xb)
        opt.zero_grad()
        loss.backward()
        opt.step()
        total += loss.item() * xb.size(0)
    if verbose and (ep==1 or ep%20==0 or ep==epochs):
        print(f"AE epoch {ep} loss {total/len(data_X):.6f}")
return ae

ae_models = []
activations_train = X_train.copy()
activations_val = X_val.copy()
activations_test = X_test.copy()

trained_encoders = []

for i, h in enumerate(hidden_sizes):
    ae = SimpleAE(input_dim=activations_train.shape[1], bottleneck_dim=h)
    ae = train_autoencoder(ae, activations_train, epochs=ae_epochs, batch_size=ae_batch, lr=ae_lr, verbose=True)
    ae_models.append(ae)
    ae = ae.to(device)
    ae.eval()
    with torch.no_grad():
        train_z = ae.enc(torch.tensor(activations_train, dtype=torch.float32).to(device)).cpu().numpy()
```

`

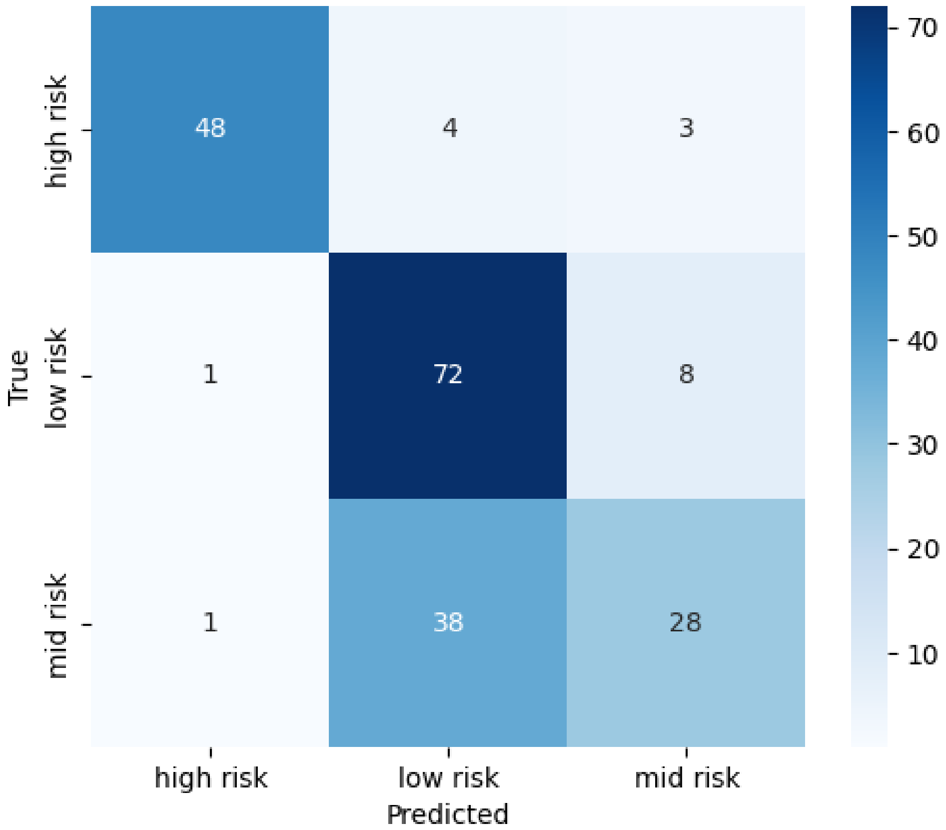
```
        val_z = ae.enc(torch.tensor(activations_val,
dtype=torch.float32).to(device)).cpu().numpy()
        test_z = ae.enc(torch.tensor(activations_test,
dtype=torch.float32).to(device)).cpu().numpy()


    linear_layer = ae.enc[0]

    enc_linear = nn.Linear(linear_layer.in_features, linear_layer.out_features)
    enc_linear.weight.data = linear_layer.weight.data.cpu().clone()
    enc_linear.bias.data = linear_layer.bias.data.cpu().clone()
    trained_encoders.append(nn.Sequential(enc_linear, nn.ReLU()))

    activations_train = train_z
    activations_val = val_z
    activations_test = test_z

print("\nPretraining finished. Number of pretrained encoders:", len(trained_encoders))

class FCNetFromEncoders(nn.Module):
    def __init__(self, encoders, n_classes, dropout=0.2):
        super().__init__()
        layers = []
        for enc in encoders:
            layers.append(enc[0])
            layers.append(nn.ReLU())
            layers.append(nn.Dropout(dropout))
```

`

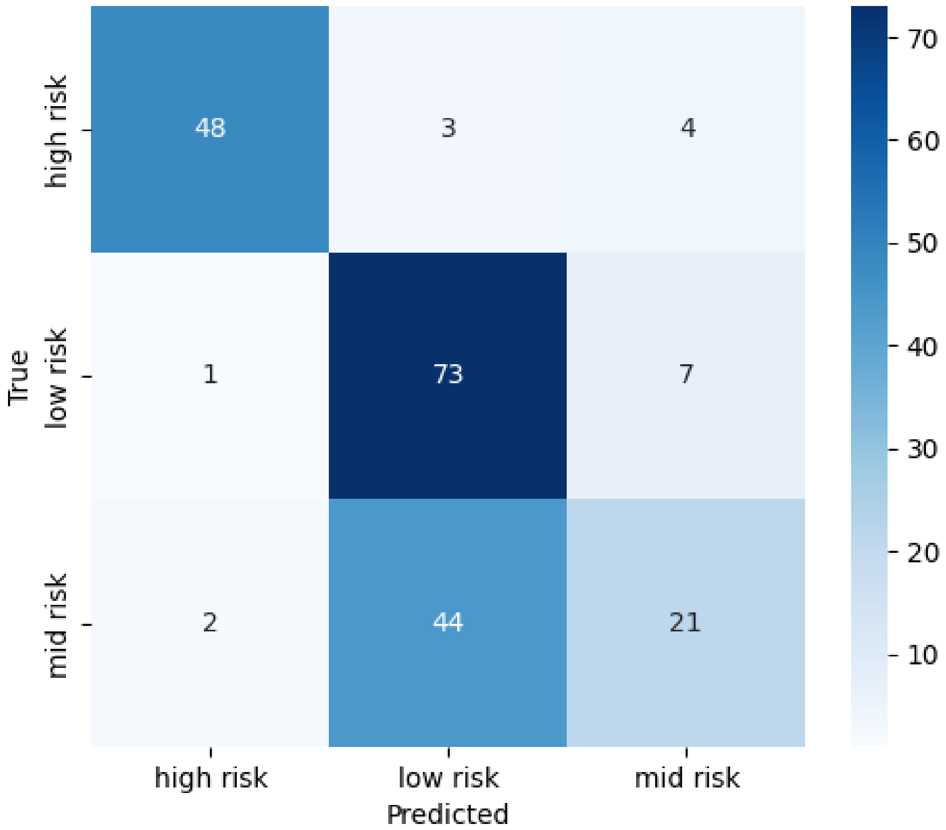
```
        last_dim = encoders[-1][0].out_features
        layers.append(nn.Linear(last_dim, n_classes))
        self.net = nn.Sequential(*layers)
    def forward(self, x):
        return self.net(x)


model_pretrained = FCNetFromEncoders(trained_encoders, n_classes, dropout=dropout).to(device)
print(model_pretrained)


model_pt_trained, hist_pt = train_classifier(model_pretrained, train_loader, val_loader, epochs=200, lr=1e-3, patience=15)


y_true_pt, y_pred_pt = evaluate_model(model_pt_trained, test_loader)
print_metrics(y_true_pt, y_pred_pt, labels=class_names)


print("Baseline (no pretraining):")
print("Accuracy:", accuracy_score(y_true, y_pred))
print("Macro F1:", f1_score(y_true, y_pred, average='macro'))
print()
print("With pretraining:")
print("Accuracy:", accuracy_score(y_true_pt, y_pred_pt))
print("Macro F1:", f1_score(y_true_pt, y_pred_pt, average='macro'))
```

Датасет 3 лаб работы:

`



high risk
low risk
mid risk
True
48
4
3
1
72
8
1
38
28
high risk
low risk
mid risk
Predicted
70
60
50
40
30
20
10

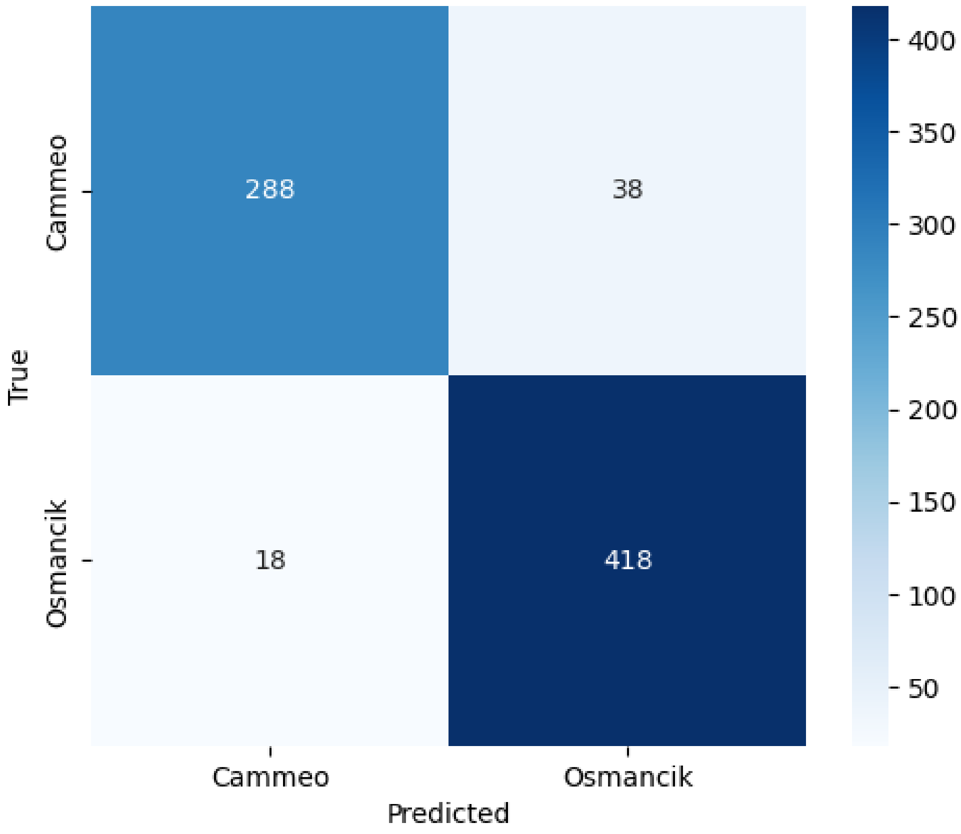


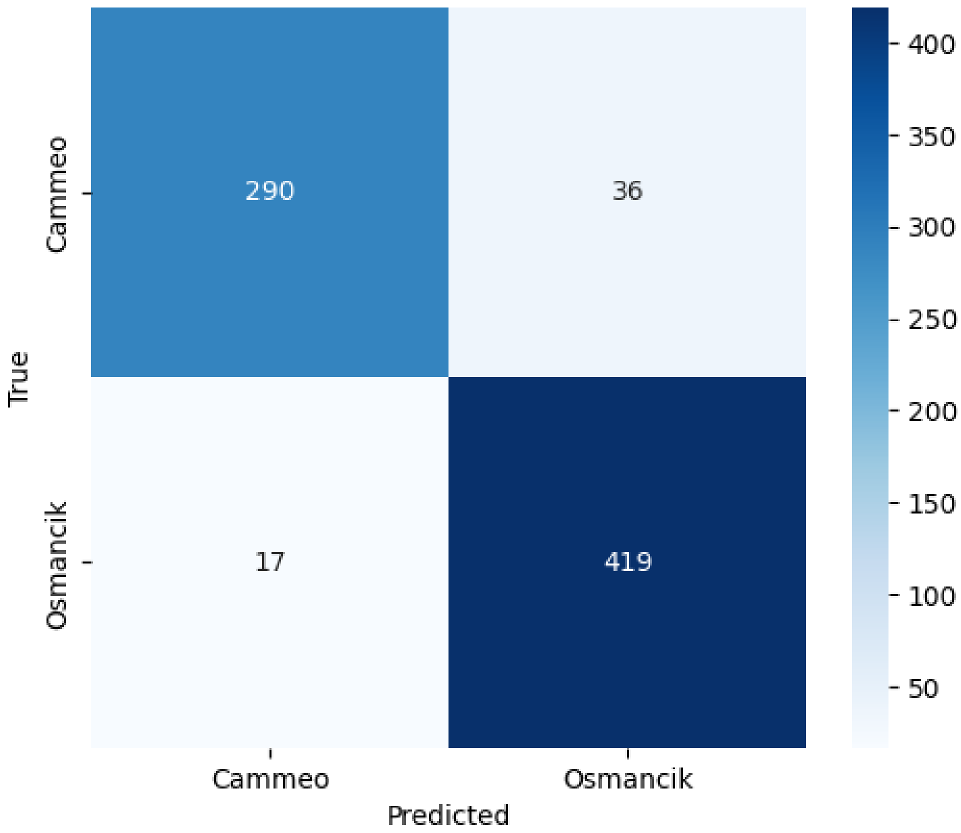
high risk
low risk
mid risk
True
48
3
4
1
73
7
2
44
21
high risk
low risk
mid risk
Predicted
70
60
50
40
30
20
10

`

```
Baseline (no pretraining):
Accuracy: 0.729064039408867
Macro F1: 0.7270163798465684

With pretraining:
Accuracy: 0.6995073891625616
Macro F1: 0.6854236536016316
```

Датасет 2 лаб работы:


288
38
18
418
Cammeo
Osmancik
True
Predicted
400
350
300
250
200
150
100
50


290
36
17
419
Cammeo
Osmancik
True
Predicted
400
350
300
250
200
150
100
50

`

```
Baseline (no pretraining):
Accuracy: 0.926509186351706
Macro F1: 0.9243060680024976

With pretraining:
Accuracy: 0.9304461942257218
Macro F1: 0.9283939979042665
```

Вывод: научился предобучать НС с помощью автоенкодеров.