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-24
Капуза Н.А.
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.

№	Выборка	Тип задачи	Целевая переменная
4	https://archive.ics.uci.edu/dataset/925/infrared+thermography+temperature+dataset	регрессия	aveOralF/aveOralM

Ход работы:

Код программы:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_absolute_percentage_error, mean_absolute_error, r2_score
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

import tensorflow as tf
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Input, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping

# Глобальные параметры
```

```
RANDOM_STATE = 42
np.random.seed(RANDOM_STATE)
tf.random.set_seed(RANDOM_STATE)


# ФУНКЦИЯ ДЛЯ ПОСТРОЕНИЯ ГРАФИКОВ КРИВЫХ ОБУЧЕНИЯ
def plot_learning_curves(history_no_pretrain, history_with_pretrain, metric, title, metric_name):
    """
    Строит кривые обучения для моделей с/без предобучения.

    :param history_no_pretrain: история обучения модели без предобучения
    :param history_with_pretrain: история обучения модели с предобучением
    :param metric: название метрики в объекте history (e.g., 'loss', 'accuracy')
    :param title: заголовок графика
    :param metric_name: название для оси Y
    """
    plt.figure(figsize=(12, 8))

    # Модель без предобучения (синие линии)
    plt.plot(history_no_pretrain.history[metric], label=f'Train ({metric_name}) - Без предобучения',
color='blue', linestyle='-')
    plt.plot(history_no_pretrain.history[f'val_{metric}'], label=f'Validation ({metric_name}) - Без
предобучения', color='cyan', linestyle='--')

    # Модель с предобучением (красные линии)
    plt.plot(history_with_pretrain.history[metric], label=f'Train ({metric_name}) - С предобучением',
color='red', linestyle='-')
    plt.plot(history_with_pretrain.history[f'val_{metric}'], label=f'Validation ({metric_name}) - С
предобучением', color='magenta', linestyle='--')

    plt.title(title, fontsize=16)
    plt.xlabel('Эпоха', fontsize=12)
    plt.ylabel(metric_name, fontsize=12)
    plt.legend(loc='best')
    plt.grid(True)
    plt.show()


# ЧАСТЬ 1: ЗАДАЧА РЕГРЕССИИ (INFRARED THERMOGRAPHY DATASET)

print("--- Часть 1: Задача регрессии (Infrared Thermography Temperature) ---")

# 1.1 Загрузка и предобработка данных
try:
    df_thermo = pd.read_csv('FLIR_groups1and2.csv')
except FileNotFoundError:
    print("Файл 'FLIR_groups1and2.csv' не найден. Пожалуйста, скачайте его и поместите в
папку с проектом.")
    df_thermo = pd.DataFrame()

if not df_thermo.empty:
```

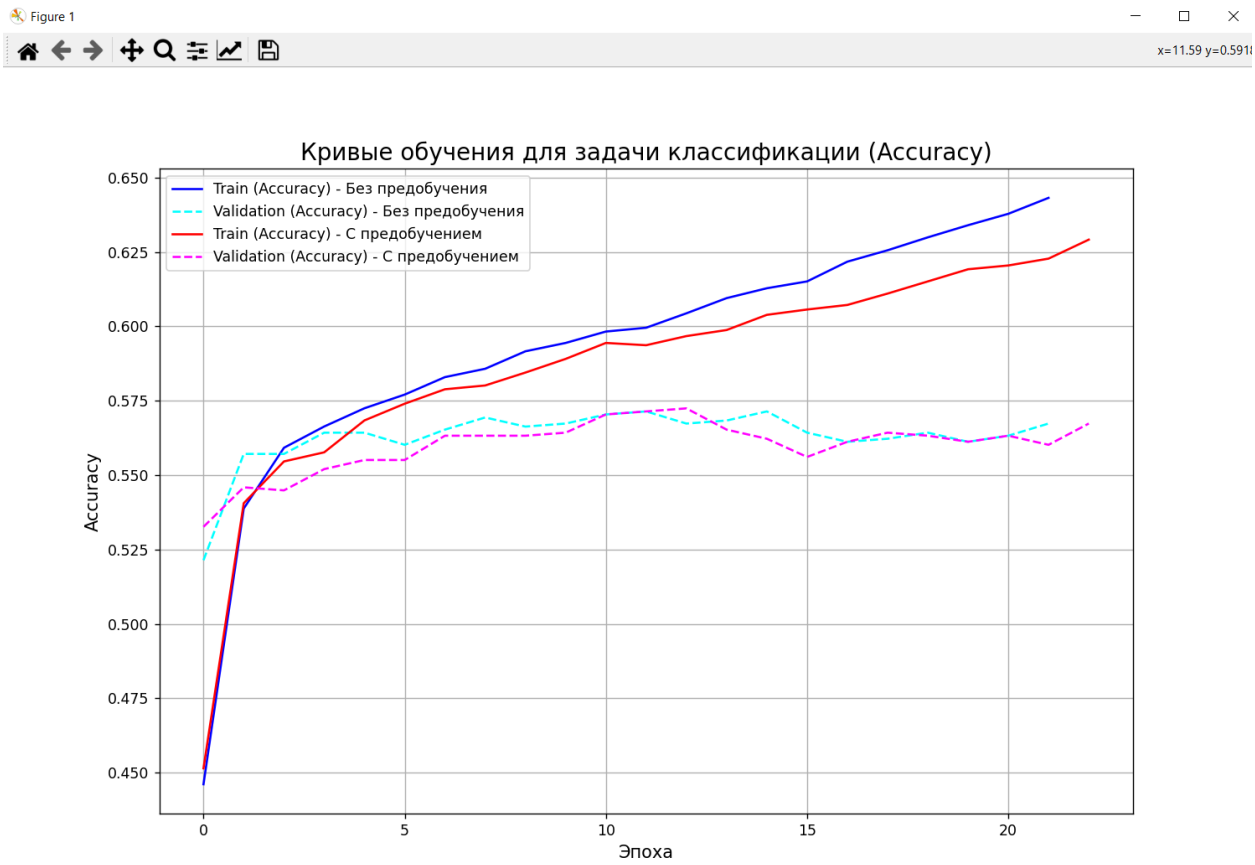
```
    df_thermo = df_thermo.drop(['subject_id', 'group'], axis=1)
    df_thermo = df_thermo.dropna(subset=['aveOralF', 'aveOralM'])
    X = df_thermo.drop(['aveOralF', 'aveOralM'], axis=1)
    y = df_thermo[['aveOralF', 'aveOralM']]
    categorical_features = ['gender', 'ethnicity']
    numerical_features = X.select_dtypes(include=np.number).columns.tolist()
    preprocessor = ColumnTransformer(transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)])
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=RANDOM_STATE)
    X_train_processed = preprocessor.fit_transform(X_train)
    X_test_processed = preprocessor.transform(X_test)
    print(f"Размерность обработанных обучающих данных: {X_train_processed.shape}")

    # 1.2 Обучение без предобучения
    print("\n--- 1.2 Обучение модели регрессии БЕЗ предобучения ---")
    def build_regression_model(input_shape):
        model = Sequential([
            Input(shape=(input_shape,)),
            Dense(128, activation='relu'),
            Dense(64, activation='relu'),
            Dense(32, activation='relu'),
            Dense(2)])
        model.compile(optimizer='adam', loss='mape', metrics=['mae'])
        return model
    model_no_pretrain = build_regression_model(X_train_processed.shape[1])
    history_no_pretrain = model_no_pretrain.fit(
        X_train_processed, y_train, validation_data=(X_test_processed, y_test),
        epochs=100, batch_size=32, verbose=0,
        callbacks=[EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)])
    print("\nОценка модели без предобучения:")
    y_pred_no_pretrain = model_no_pretrain.predict(X_test_processed)
    mape_no_pretrain = mean_absolute_percentage_error(y_test, y_pred_no_pretrain)
    mae_no_pretrain = mean_absolute_error(y_test, y_pred_no_pretrain)
    r2_no_pretrain = r2_score(y_test, y_pred_no_pretrain)
    print(f"MAPE: {mape_no_pretrain:.4f}, MAE: {mae_no_pretrain:.4f}, R^2 Score:
{r2_no_pretrain:.4f}")

    # 1.3 Обучение с предобучением автоэнкодерами
    print("\n--- 1.3 Обучение модели регрессии С предобучением ---")
    #код предобучения автоэнкодеров
    input_layer1 = Input(shape=(X_train_processed.shape[1],))
    encoded1 = Dense(128, activation='relu')(input_layer1)
    autoencoder1 = Model(input_layer1, Dense(X_train_processed.shape[1])(encoded1))
    autoencoder1.compile(optimizer='adam', loss='mse')
    autoencoder1.fit(X_train_processed, X_train_processed, epochs=50, batch_size=64, shuffle=True,
verbose=0)
    encoder1 = Model(input_layer1, encoded1)
    layer1_pretrained_weights = encoder1.layers[1].get_weights()
    X_train_encoded1 = encoder1.predict(X_train_processed)
```

```
    input_layer2 = Input(shape=(128,))
    encoded2 = Dense(64, activation='relu')(input_layer2)
    autoencoder2 = Model(input_layer2, Dense(128)(encoded2))
    autoencoder2.compile(optimizer='adam', loss='mse')
    autoencoder2.fit(X_train_encoded1, X_train_encoded1, epochs=50, batch_size=64, shuffle=True,
verbose=0)
    encoder2 = Model(input_layer2, encoded2)
    layer2_pretrained_weights = encoder2.layers[1].get_weights()
    X_train_encoded2 = encoder2.predict(X_train_encoded1)
    input_layer3 = Input(shape=(64,))
    encoded3 = Dense(32, activation='relu')(input_layer3)
    autoencoder3 = Model(input_layer3, Dense(64)(encoded3))
    autoencoder3.compile(optimizer='adam', loss='mse')
    autoencoder3.fit(X_train_encoded2, X_train_encoded2, epochs=50, batch_size=64, shuffle=True,
verbose=0)
    encoder3 = Model(input_layer3, encoded3)
    layer3_pretrained_weights = encoder3.layers[1].get_weights()
    model_with_pretrain = build_regression_model(X_train_processed.shape[1])
    model_with_pretrain.layers[0].set_weights(layer1_pretrained_weights)
    model_with_pretrain.layers[1].set_weights(layer2_pretrained_weights)
    model_with_pretrain.layers[2].set_weights(layer3_pretrained_weights)
    history_with_pretrain = model_with_pretrain.fit(
        X_train_processed, y_train, validation_data=(X_test_processed, y_test),
        epochs=100, batch_size=32, verbose=0,
        callbacks=[EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)])
    print("\nОценка модели с предобучением:")
    y_pred_with_pretrain = model_with_pretrain.predict(X_test_processed)
    mape_with_pretrain = mean_absolute_percentage_error(y_test, y_pred_with_pretrain)
    mae_with_pretrain = mean_absolute_error(y_test, y_pred_with_pretrain)
    r2_with_pretrain = r2_score(y_test, y_pred_with_pretrain)
    print(f"MAPE: {mape_with_pretrain:.4f}, MAE: {mae_with_pretrain:.4f}, R^2 Score:
{r2_with_pretrain:.4f}")

    # 1.4 Сравнение результатов и выводы
    print("\n--- 1.4 Сравнение результатов (Регрессия) ---")
    results_df = pd.DataFrame({'Metric': ['MAPE', 'MAE', 'R^2 Score'],
                               'Без предобучения': [mape_no_pretrain, mae_no_pretrain, r2_no_pretrain],
                               'С предобучением': [mape_with_pretrain, mae_with_pretrain,
r2_with_pretrain]})
    print(results_df)

    # 1.5 ПОСТРОЕНИЕ ГРАФИКА ДЛЯ РЕГРЕССИИ
    plot_learning_curves(history_no_pretrain, history_with_pretrain,
                         metric='loss',  # loss - это MAPE
                         title='Кривые обучения для задачи регрессии (MAPE)',
                         metric_name='MAPE Loss')


# ЧАСТЬ 2: ЗАДАЧА КЛАССИФИКАЦИИ (WINE QUALITY DATASET)

print("\n\n--- Часть 2: Задача классификации (Wine Quality) ---")
```

```
# 2.1 Загрузка и предобработка данных
try:
    df_wine = pd.read_csv('winequality-white.csv', sep=';')
except FileNotFoundError:
    print("Файл 'winequality-white.csv' не найден. Загрузка из сети...")
    url = "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-
white.csv"
    df_wine = pd.read_csv(url, sep=';')

X_wine = df_wine.drop('quality', axis=1)
y_wine = df_wine['quality'] - df_wine['quality'].min()
num_classes = y_wine.nunique()
scaler_wine = StandardScaler()
X_wine_scaled = scaler_wine.fit_transform(X_wine)
X_train_w, X_test_w, y_train_w, y_test_w = train_test_split(
    X_wine_scaled, y_wine, test_size=0.2, random_state=RANDOM_STATE, stratify=y_wine)
print(f"Размерность обучающих данных: {X_train_w.shape}, Количество классов:
{num_classes}")

# 2.2 Обучение без предобучения
print("\n--- 2.2 Обучение модели классификации БЕЗ предобучения ---")
def build_classification_model(input_shape, num_classes):
    model = Sequential([
        Input(shape=(input_shape,)),
        Dense(128, activation='relu'),
        Dense(64, activation='relu'),
        Dense(32, activation='relu'),
        Dense(num_classes, activation='softmax')])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model
model_cls_no_pretrain = build_classification_model(X_train_w.shape[1], num_classes)
history_cls_no_pretrain = model_cls_no_pretrain.fit(
    X_train_w, y_train_w, validation_data=(X_test_w, y_test_w),
    epochs=100, batch_size=64, verbose=0,
    callbacks=[EarlyStopping(monitor='val_accuracy', patience=10, restore_best_weights=True)])
print("\nОценка модели без предобучения:")
y_pred_cls_no_pretrain = np.argmax(model_cls_no_pretrain.predict(X_test_w), axis=1)
acc_no_pretrain = accuracy_score(y_test_w, y_pred_cls_no_pretrain)
print(f"Accuracy: {acc_no_pretrain:.4f}")
print("Classification Report:")
print(classification_report(y_test_w, y_pred_cls_no_pretrain, zero_division=0))

#  2.3 Обучение с предобучением автоэнкодерами
print("\n--- 2.3 Обучение модели классификации С предобучением ---")
# код предобучения автоэнкодеров
input_w1 = Input(shape=(X_train_w.shape[1],))
encoded_w1 = Dense(128, activation='relu')(input_w1)
autoencoder_w1 = Model(input_w1, Dense(X_train_w.shape[1])(encoded_w1))
autoencoder_w1.compile(optimizer='adam', loss='mse')
autoencoder_w1.fit(X_train_w, X_train_w, epochs=50, batch_size=128, shuffle=True, verbose=0)
```

```
encoder_w1 = Model(input_w1, encoded_w1)
layer1_w_weights = encoder_w1.layers[1].get_weights()
X_train_w_encoded1 = encoder_w1.predict(X_train_w)
input_w2 = Input(shape=(128,))
encoded_w2 = Dense(64, activation='relu')(input_w2)
autoencoder_w2 = Model(input_w2, Dense(128)(encoded_w2))
autoencoder_w2.compile(optimizer='adam', loss='mse')
autoencoder_w2.fit(X_train_w_encoded1, X_train_w_encoded1, epochs=50, batch_size=128,
shuffle=True, verbose=0)
encoder_w2 = Model(input_w2, encoded_w2)
layer2_w_weights = encoder_w2.layers[1].get_weights()
X_train_w_encoded2 = encoder_w2.predict(X_train_w_encoded1)
input_w3 = Input(shape=(64,))
encoded_w3 = Dense(32, activation='relu')(input_w3)
autoencoder_w3 = Model(input_w3, Dense(64)(encoded_w3))
autoencoder_w3.compile(optimizer='adam', loss='mse')
autoencoder_w3.fit(X_train_w_encoded2, X_train_w_encoded2, epochs=50, batch_size=128,
shuffle=True, verbose=0)
encoder_w3 = Model(input_w3, encoded_w3)
layer3_w_weights = encoder_w3.layers[1].get_weights()
model_cls_with_pretrain = build_classification_model(X_train_w.shape[1], num_classes)
model_cls_with_pretrain.layers[0].set_weights(layer1_w_weights)
model_cls_with_pretrain.layers[1].set_weights(layer2_w_weights)
model_cls_with_pretrain.layers[2].set_weights(layer3_w_weights)
history_cls_with_pretrain = model_cls_with_pretrain.fit(
   X_train_w, y_train_w, validation_data=(X_test_w, y_test_w),
   epochs=100, batch_size=64, verbose=0,
   callbacks=[EarlyStopping(monitor='val_accuracy', patience=10, restore_best_weights=True)])
print("\nОценка модели с предобучением:")
y_pred_cls_with_pretrain = np.argmax(model_cls_with_pretrain.predict(X_test_w), axis=1)
acc_with_pretrain = accuracy_score(y_test_w, y_pred_cls_with_pretrain)
print(f"Accuracy: {acc_with_pretrain:.4f}")
print("Classification Report:")
print(classification_report(y_test_w, y_pred_cls_with_pretrain, zero_division=0))

# 2.4 Сравнение результатов и выводы
print("\n--- 2.4 Сравнение результатов (Классификация) ---")
results_cls_df = pd.DataFrame({'Metric': ['Accuracy'],
                               'Без предобучения': [acc_no_pretrain],
                               'С предобучением': [acc_with_pretrain]})
print(results_cls_df)

# 2.5 ПОСТРОЕНИЕ ГРАФИКА ДЛЯ КЛАССИФИКАЦИИ
plot_learning_curves(history_cls_no_pretrain, history_cls_with_pretrain,
                     metric='accuracy',
                     title='Кривые обучения для задачи классификации (Accuracy)',
                     metric_name='Accuracy')
```

График:



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