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

Лабораторная работа №4
По дисциплине: «ОМО»
Тема : “Введение в нейронные сети:
построение многослойного перцептрона”

Выполнил:
Студент 3-го курса
Группы АС-66
Цеван К.А.
Проверил:
Крощенко А.А.

Брест 2025

Цель: построить, обучить и оценить многослойный перцептрон (MLP) для решения задачи классификации

Вариант 12

Вариант 12 Детекция аномалий в сети
• KDD Cup 1999

Основы машинного обучения, 2025, Крощенко А.А.
• Задача: классифицировать подключения на "нормальные" и "атаки" (бинарная классификация).
• Архитектура:
o входной слой;
o один скрытый слой с 64 нейронами (ReLU);
o выходной слой с 1 нейроном (Sigmoid).
• Эксперимент: уменьшите количество нейронов до 16. Насколько сильно упала точность и как изменилось время обучения?

```
# -*- coding: utf-8 -*-

import os
import time
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
import torch.optim as optim

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.metrics import accuracy_score, f1_score, roc_curve, auc, confusion_matrix

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

current_dir = os.path.dirname(os.path.abspath(__file__))
dataset_path = os.path.join(current_dir, "kddcup.data_10_percent_corrected")

df = pd.read_csv(dataset_path, header=None)
df.columns = [f"col_{i}" for i in range(df.shape[1])]
df["target"] = (df["col_41"] != "normal.").astype(int)

X = df.drop(columns=["target", "col_41"])
y = df["target"]
```

```
numeric_features = X.select_dtypes(include=["int64", "float64"]).columns.tolist()
categorical_features = X.select_dtypes(include=["object"]).columns.tolist()

preprocess = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), numeric_features),
        ("cat", OneHotEncoder(handle_unknown="ignore"), categorical_features)
    ]
)

X_train_raw, X_test_raw, y_train_np, y_test_np = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

X_train = preprocess.fit_transform(X_train_raw)
X_test = preprocess.transform(X_test_raw)

if hasattr(X_train, "toarray"):
    X_train = X_train.toarray()
    X_test = X_test.toarray()

X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train_np.values, dtype=torch.float32).view(-1, 1)
y_test_tensor = torch.tensor(y_test_np.values, dtype=torch.float32).view(-1, 1)

input_dim = X_train_tensor.shape[1]

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
X_train_tensor = X_train_tensor.to(device)
X_test_tensor = X_test_tensor.to(device)
y_train_tensor = y_train_tensor.to(device)
y_test_tensor = y_test_tensor.to(device)

class MLP64(nn.Module):
    def __init__(self, input_dim):
        super().__init__()
        self.fc1 = nn.Linear(input_dim, 64)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(64, 1)
    def forward(self, x):
        return self.fc2(self.relu(self.fc1(x)))

class MLP16(nn.Module):
    def __init__(self, input_dim):
        super().__init__()
        self.fc1 = nn.Linear(input_dim, 16)
```

```
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(16, 1)
    def forward(self, x):
        return self.fc2(self.relu(self.fc1(x)))

def train_and_evaluate(ModelClass, input_dim, n_epochs=10, lr=0.001):
    model = ModelClass(input_dim).to(device)
    criterion = nn.BCEWithLogitsLoss()
    optimizer = optim.Adam(model.parameters(), lr=lr)
    loss_history = []
    start_time = time.time()

    for epoch in range(n_epochs):
        model.train()
        logits = model(X_train_tensor)
        loss = criterion(logits, y_train_tensor)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        loss_history.append(loss.item())
        if (epoch + 1) % 2 == 0:
            print(f"Epoch {epoch+1}/{n_epochs} Loss: {loss.item():.4f}")

    train_time = time.time() - start_time

    model.eval()
    with torch.no_grad():
        logits_test = model(X_test_tensor)
        probs_test = torch.sigmoid(logits_test)
        preds = (probs_test > 0.5).float()

    y_true = y_test_tensor.cpu().numpy().ravel()
    y_pred = preds.cpu().numpy().ravel()
    y_prob = probs_test.cpu().numpy().ravel()

    acc = accuracy_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)

    plt.figure(figsize=(7, 4))
    plt.plot(loss_history)
    plt.title(f"Loss — {ModelClass.__name__}")
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.grid(True)
    plt.show()

    fpr, tpr, _ = roc_curve(y_true, y_prob)
```

```
    roc_auc = auc(fpr, tpr)

    plt.figure(figsize=(6, 6))
    plt.plot(fpr, tpr, label=f"AUC={roc_auc:.4f}")
    plt.plot([0, 1], [0, 1], '--', color="gray")
    plt.title(f"ROC — {ModelClass.__name__}")
    plt.xlabel("FPR")
    plt.ylabel("TPR")
    plt.legend()
    plt.grid(True)
    plt.show()

    cm = confusion_matrix(y_true, y_pred)

    plt.figure(figsize=(5, 4))
    plt.imshow(cm, cmap="Blues")
    plt.title(f"Confusion Matrix — {ModelClass.__name__}")
    plt.colorbar()
    plt.xlabel("Predicted")
    plt.ylabel("True")
    for i in range(2):
        for j in range(2):
            plt.text(j, i, cm[i, j], ha="center", va="center", color="red")
    plt.show()

    return acc, f1, train_time


print("\n==== 64 neurons ====")
acc64, f164, time64 = train_and_evaluate(MLP64, input_dim)
print(f"Accuracy={acc64:.4f}, F1={f164:.4f}, Time={time64:.2f}")

print("\n==== 16 neurons ====")
acc16, f116, time16 = train_and_evaluate(MLP16, input_dim)
print(f"Accuracy={acc16:.4f}, F1={f116:.4f}, Time={time16:.2f}")

print("\n==== Comparison ====")
print(f"ΔAccuracy={acc16 - acc64:.4f}")
print(f"ΔTime={time16 - time64:.2f}")

# ======== GRAPH: MODEL COMPARISON ========

models = ["MLP64", "MLP16"]
acc_values = [acc64, acc16]
f1_values = [f164, f116]
time_values = [time64, time16]
```

```
x = np.arange(len(models))
w = 0.25

plt.figure(figsize=(9, 5))
plt.bar(x - w, acc_values, width=w, label="Accuracy")
plt.bar(x, f1_values, width=w, label="F1-score")
plt.bar(x + w, time_values, width=w, label="Time")

plt.xticks(x, models)
plt.title("Сравнение моделей (64 vs 16 нейронов)")
plt.ylabel("Значение")
plt.legend()
plt.grid(True)
plt.show()
```

```
==== 64 neurons ====
Epoch 2/10 Loss: 0.7064
Epoch 4/10 Loss: 0.6778
Epoch 6/10 Loss: 0.6509
Epoch 8/10 Loss: 0.6255
Epoch 10/10 Loss: 0.6009
Accuracy=0.9854, F1=0.9909, Time=1.38

==== 16 neurons ====
Epoch 2/10 Loss: 0.6966
Epoch 4/10 Loss: 0.6874
Epoch 6/10 Loss: 0.6788
Epoch 8/10 Loss: 0.6708
Epoch 10/10 Loss: 0.6630
Accuracy=0.4068, F1=0.4281, Time=0.52

==== Comparison ====
ΔAccuracy=-0.5786
ΔTime=-0.85
```

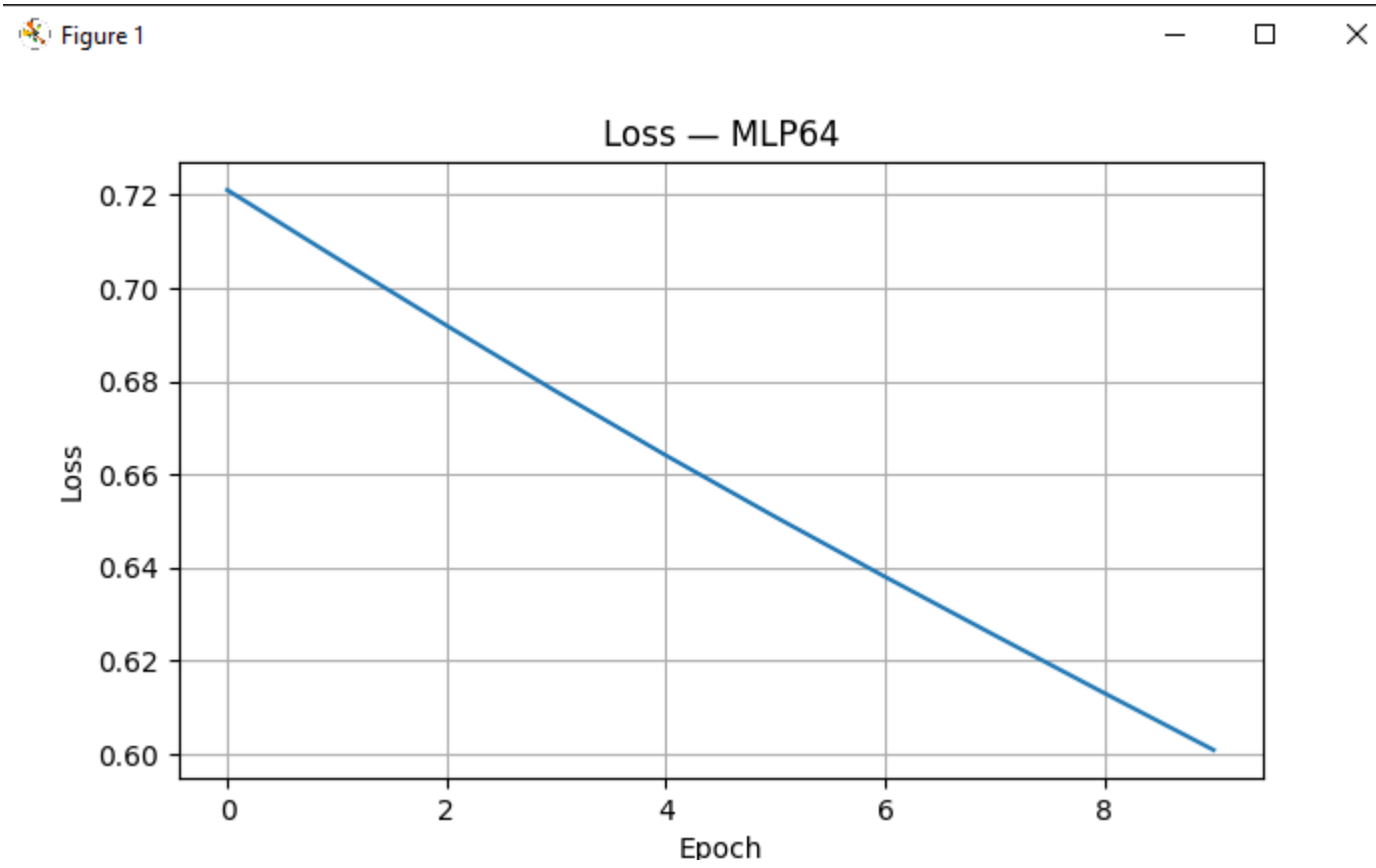


Figure 1
Loss — MLP64
Loss
Epoch
0.72
0.70
0.68
0.66
0.64
0.62
0.60
0
2
4
6
8

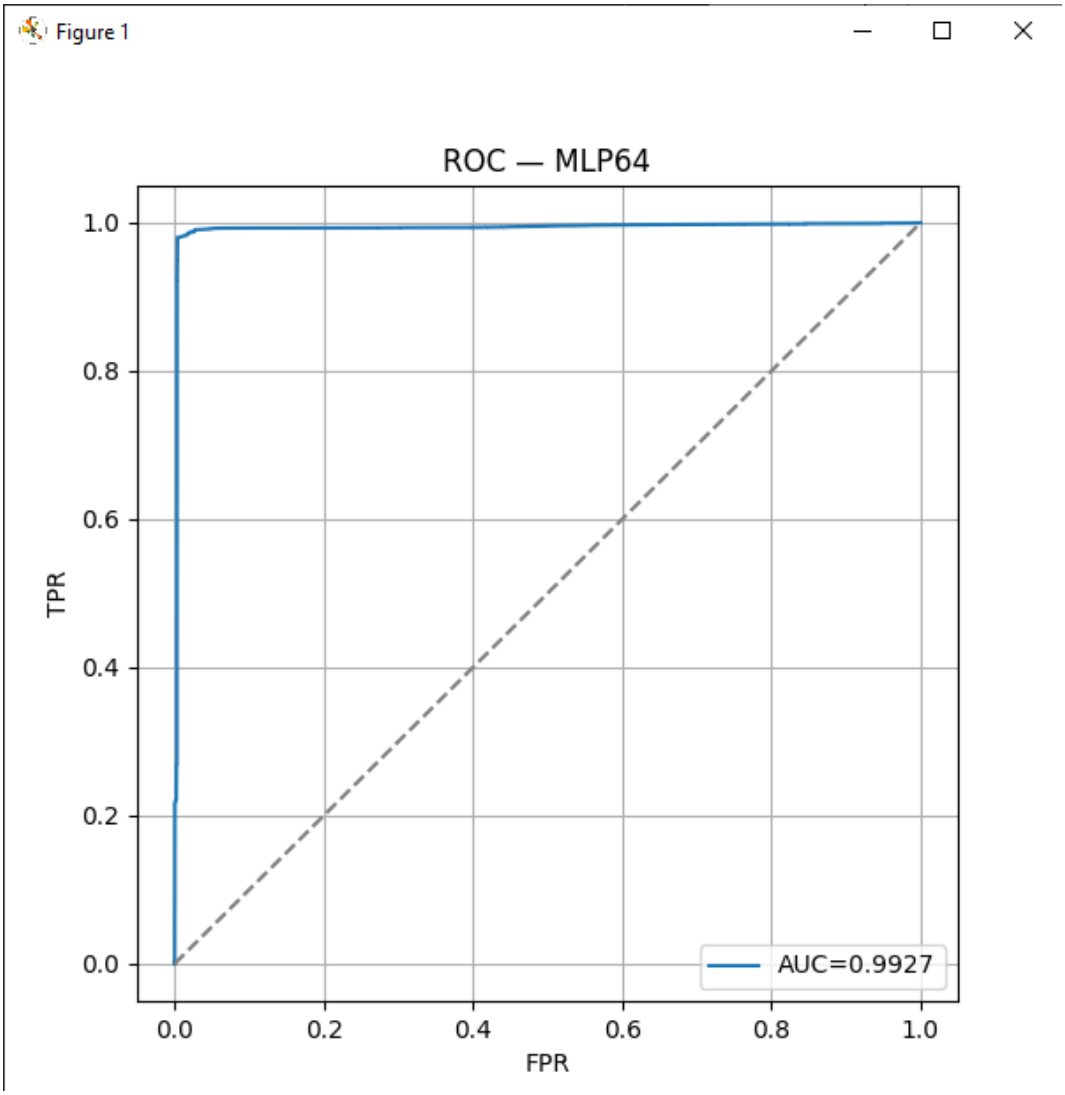


Figure 1
ROC — MLP64
TPR
FPR
AUC=0.9927
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

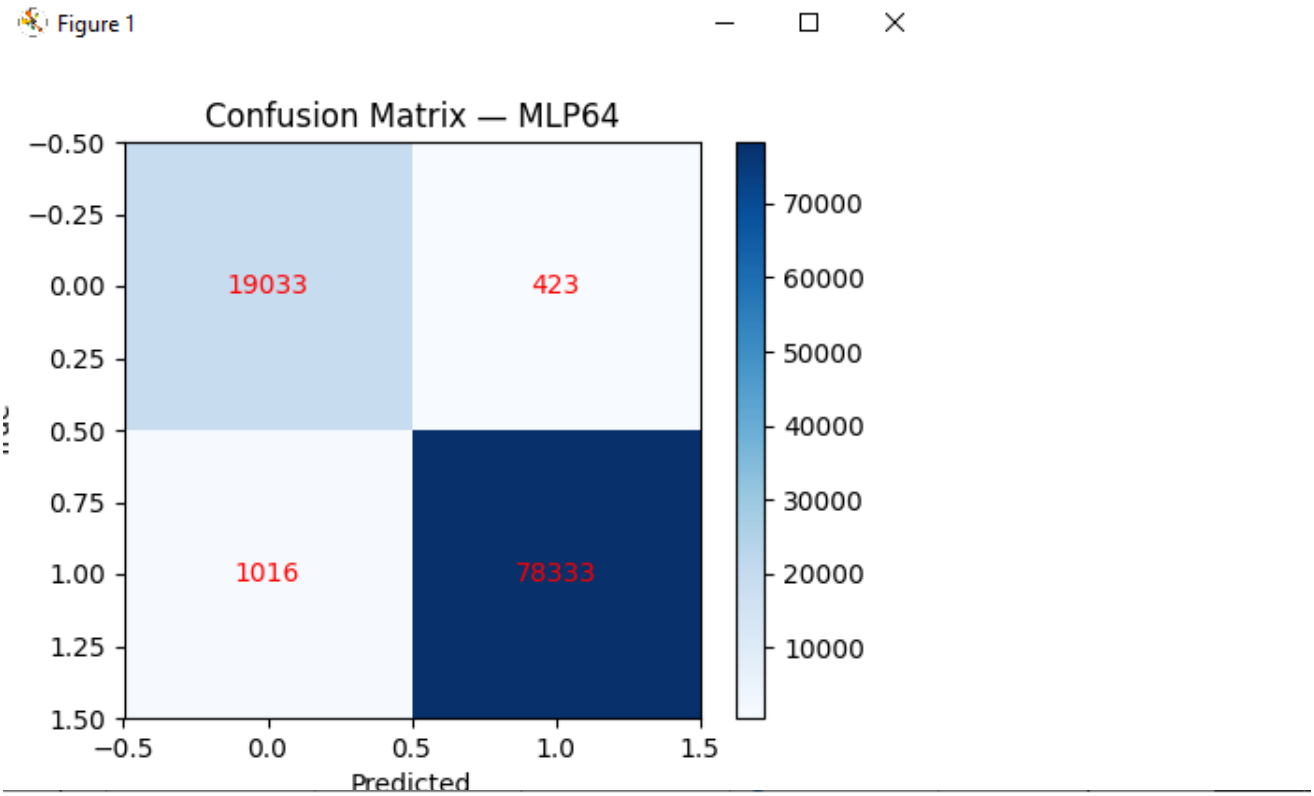


Figure 1
Confusion Matrix — MLP64
19033
423
1016
78333
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50
−0.5
0.0
0.5
1.0
1.5
Predicted
70000
60000
50000
40000
30000
20000
10000

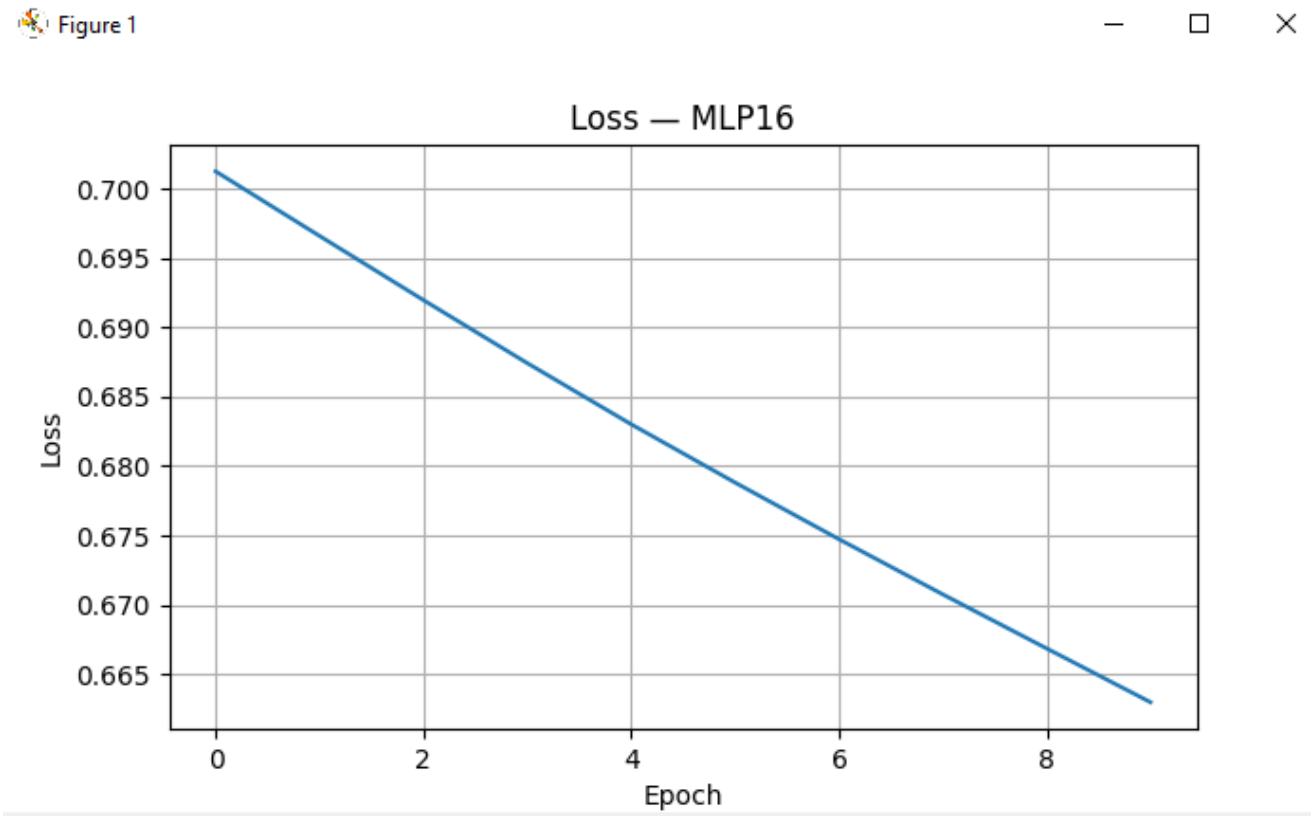
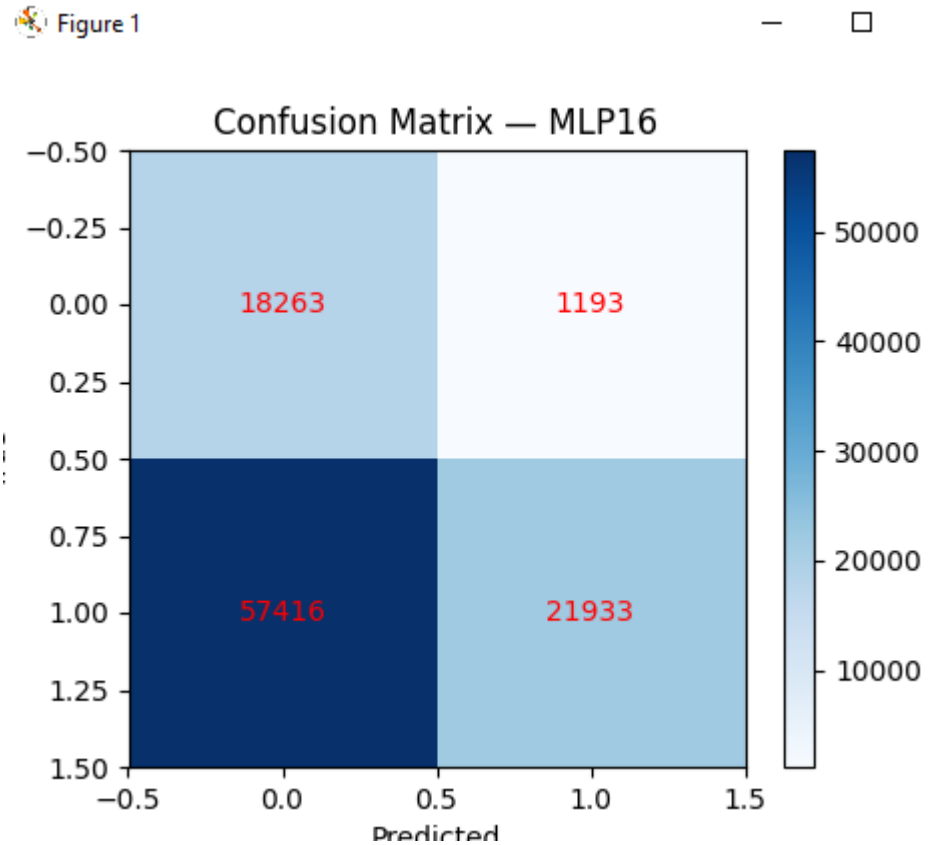
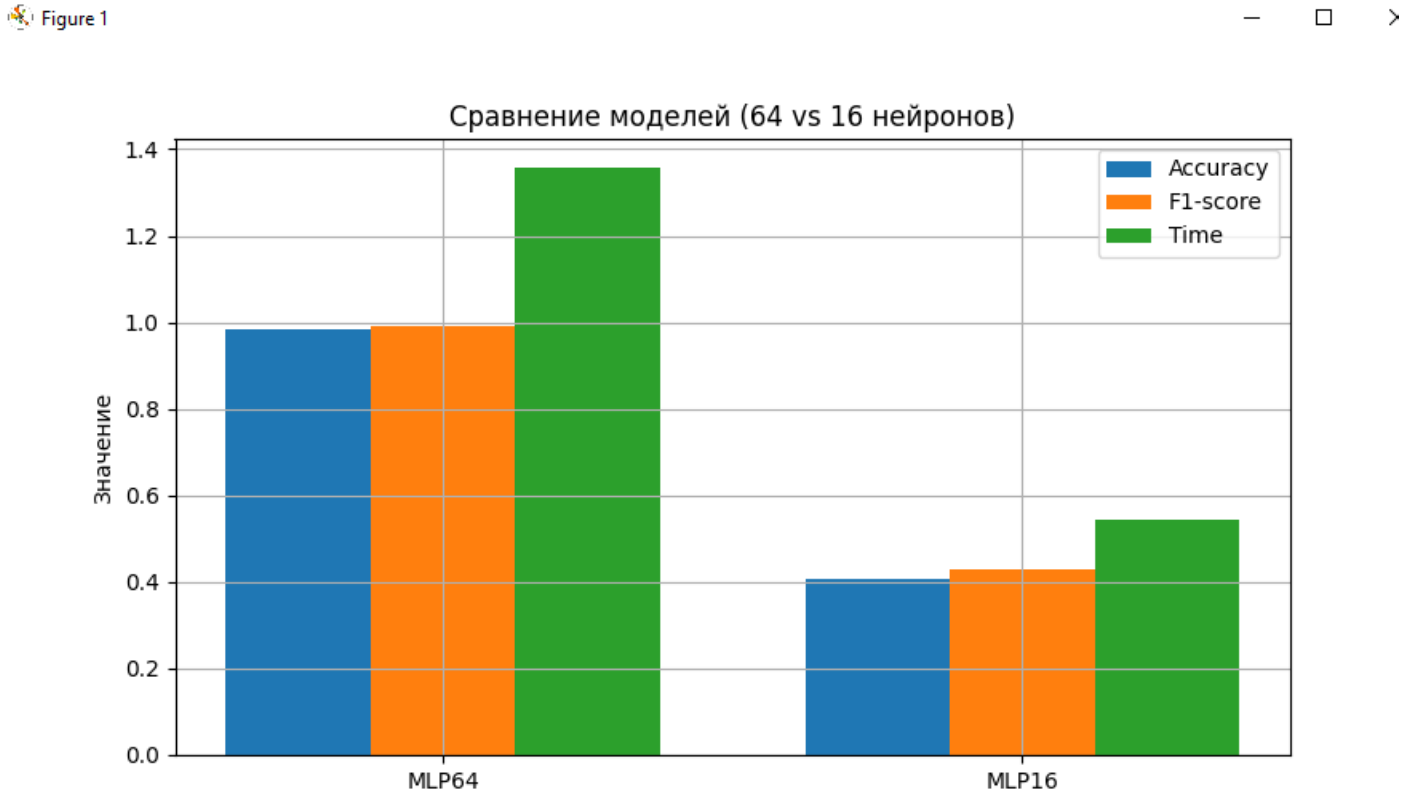


Figure 1
Loss — MLP16
0.700
0.695
0.690
0.685
0.680
0.675
0.670
0.665
Loss
0
2
4
6
8
Epoch





Вывод: построили, обучили и оценили многослойный перцептрон (MLP) для решения задачи классификациина практике сравнили работу несколько алгоритмов классификации.