Федеральное государственное автономное

образовательное учреждение высшего образования

«Пермский государственный национальный исследовательский университет» (ПГНИУ)

Региональный институт непрерывного образования (РИНО ПГНИУ)

Цифровая кафедра

Лабораторная работа на тему: «Автоэнкодер»

по курсу профессиональной переподготовки «Анализ данных и машинное обучение»

Студент 4 курса группы БАС-21:

Богомаз Виктор Артемович

Пермь, 2024

# Оглавление

[Оглавление 2](#_Toc185794805)

[Цель работы 3](#_Toc185794806)

[Реализация работы 4](#_Toc185794807)

[Анализ результатов 11](#_Toc185794808)

[Выводы 16](#_Toc185794809)

[Список литературы 17](#_Toc185794810)

# Цель работы

Целью данной лабораторной работы является разработка и обучение автоэнкодера с использованием набора данных UNSW-NB15. Этот набор данных включает записи как о нормальном, так и о вредоносном интернет-трафике. Обучение автоэнкодера будет проводиться исключительно на записях нормального трафика, поскольку они составляют большую часть данных. Для проверки эффективности модели будет использоваться весь набор данных, охватывающий оба типа трафика.

# Реализация работы

Работу буду реализовывать на языке программирования Python.

Добавим необходимые библиотеки.

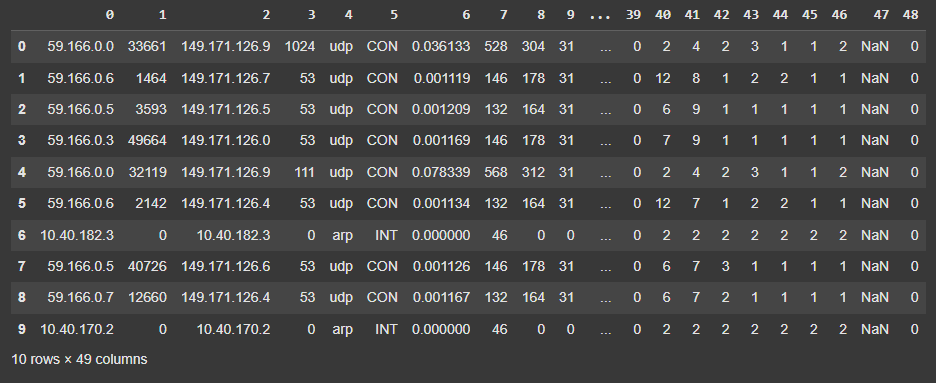
|  |
| --- |
| import pandas as pd  import numpy as np  import pickle  from torch.autograd import Variable  import torch  import torch.nn as nn  import torch.nn.functional as F  from torch.utils.data import DataLoader  import torch.utils.data as data\_utils  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import StandardScaler  from scipy import stats  import matplotlib.pyplot as plt  import seaborn as sns  from pylab import rcParams  from sklearn.metrics import (confusion\_matrix, precision\_recall\_curve, auc,                               roc\_curve, recall\_score, classification\_report, f1\_score,                               precision\_recall\_fscore\_support)  sns.set(style='whitegrid', palette='muted', font\_scale=1.5)  rcParams['figure.figsize'] = 14, 8  RANDOM\_SEED = 42 |

Загрузим наш датасет и найдем его.

|  |
| --- |
| import kagglehub  # Download latest version  path = kagglehub.dataset\_download("mrwellsdavid/unsw-nb15")  print("Path to dataset files:", path)  import os  # Получение списка файлов в папке  files = os.listdir(path)  print("Files in dataset:", files)  # Выбираем файл "UNSW\_NB15\_1.csv"  csv\_file = [f for f in files if f.endswith("UNSW-NB15\_1.csv")][0]  file\_path = os.path.join(path, csv\_file)  print("Using file:", file\_path)  # Загрузка данных в DataFrame  df = pd.read\_csv(file\_path)  print("Dataset shape:", df.shape)  df.columns = range(df.shape[1])  # Заменяем названия столбцов на числа от 0 до n-1  print("Columns:", df.columns) |

Целевая переменная в 48 столбце. 0 – интернет трафик нормальный, 1 – интернет трафик вредоносный.

Выведем содержимое датасета

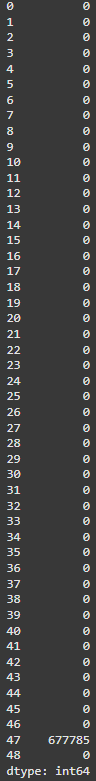


|  |
| --- |
| df.head(10) |

1. Первые 10 строк датасета

Посмотрим присутствие пропусков по столбцам

|  |
| --- |
| # Подсчёт количества пустых ячеек в каждом столбце  missing\_values = df.isna().sum()  # Вывод результата  print(missing\_values) |



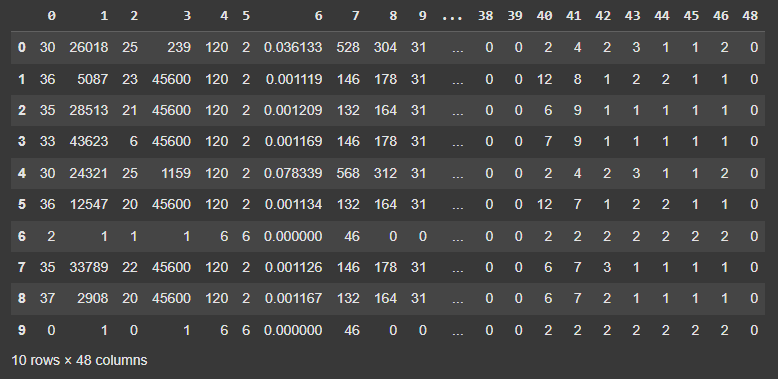
1. Количество пропусков в каждом столбце

Принимаю решение удалить 47 столбец так 97% ячеек содержат пропуск.

Далее необходимо перевести категориальные столбцы в числовые.

|  |
| --- |
| from sklearn.preprocessing import LabelEncoder  label\_encoder = LabelEncoder()  # Список категориальных столбцов  categorical\_columns = [0, 1, 2, 3, 4, 5, 13]  for col in categorical\_columns:      df[col] = label\_encoder.fit\_transform(df[col]) |

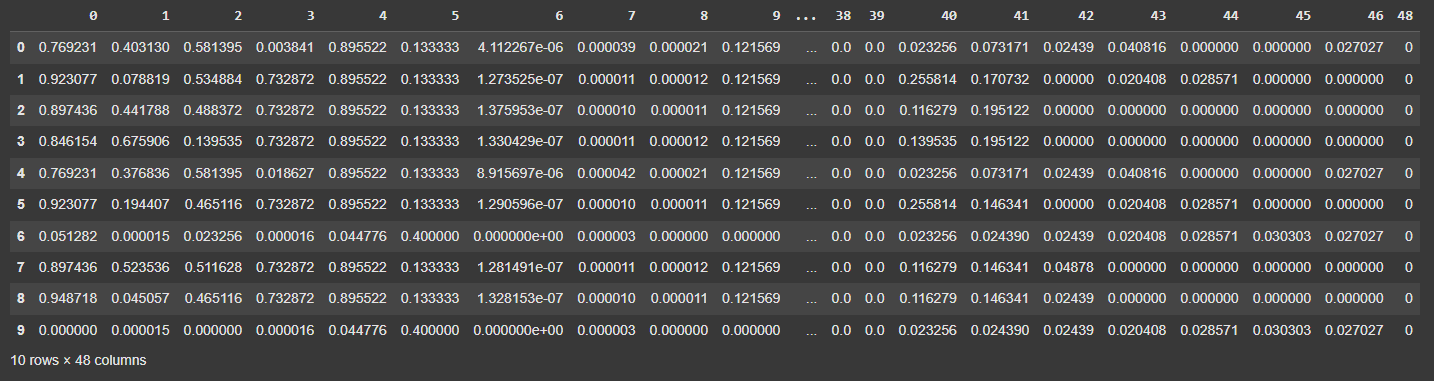
Выведем наш набор данных после преобразований



1. Содержимое датасета после перевода всех столбцов в числовой тип

Мы видим большой перепад между числами, необходимо делать нормализацию.

|  |
| --- |
| from sklearn.preprocessing import MinMaxScaler  # Нормализация  # Список всех столбцов, кроме столбца 48  columns\_to\_normalize = [col for col in df.columns if col != 48]  scaler = MinMaxScaler()  df[columns\_to\_normalize] = scaler.fit\_transform(df[columns\_to\_normalize]) |

1. Содержимое датасета после нормализации

Так как столбцов 48, с помощью тепловой карты, выберем наиболее коррелируемые с целевой переменной.

|  |
| --- |
| correlation\_matrix = df.corr()  # Создание тепловой карты  plt.figure(figsize=(80, 60))  sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt='.2f')  plt.title("Корреляционная матрица")  plt.show() |

Я выбрал 0, 9, 10, 14, 23, 29, 32 и 36 столбец.

|  |
| --- |
| df = df[[0, 9, 10, 14, 23, 29, 32, 36, 48]] |

Автоэнкодер будем обучать на нормальном интернет трафике, но для проверки будут использоваться оба класса.

|  |
| --- |
| # Разделение данных  X\_train, X\_test = train\_test\_split(df, test\_size=0.2, random\_state=RANDOM\_SEED)  # Тренировочные данные  X\_train = X\_train[X\_train[48] == 0]  y\_train = X\_train[48]  X\_train = X\_train.drop([48], axis=1)  # Тестовые данные  y\_test = X\_test[48]  X\_test = X\_test.drop([48], axis=1)  # Преобразование данных в numpy для PyTorch  X\_train = X\_train.values  X\_test = X\_test.values  y\_test = y\_test.values  # Создание DataLoader  minibatch\_size = 50  train\_loader = data\_utils.DataLoader(torch.tensor(X\_train, dtype=torch.float32), batch\_size=minibatch\_size, shuffle=True)  test\_loader = data\_utils.DataLoader(torch.tensor(X\_test, dtype=torch.float32), batch\_size=1, shuffle=False) |

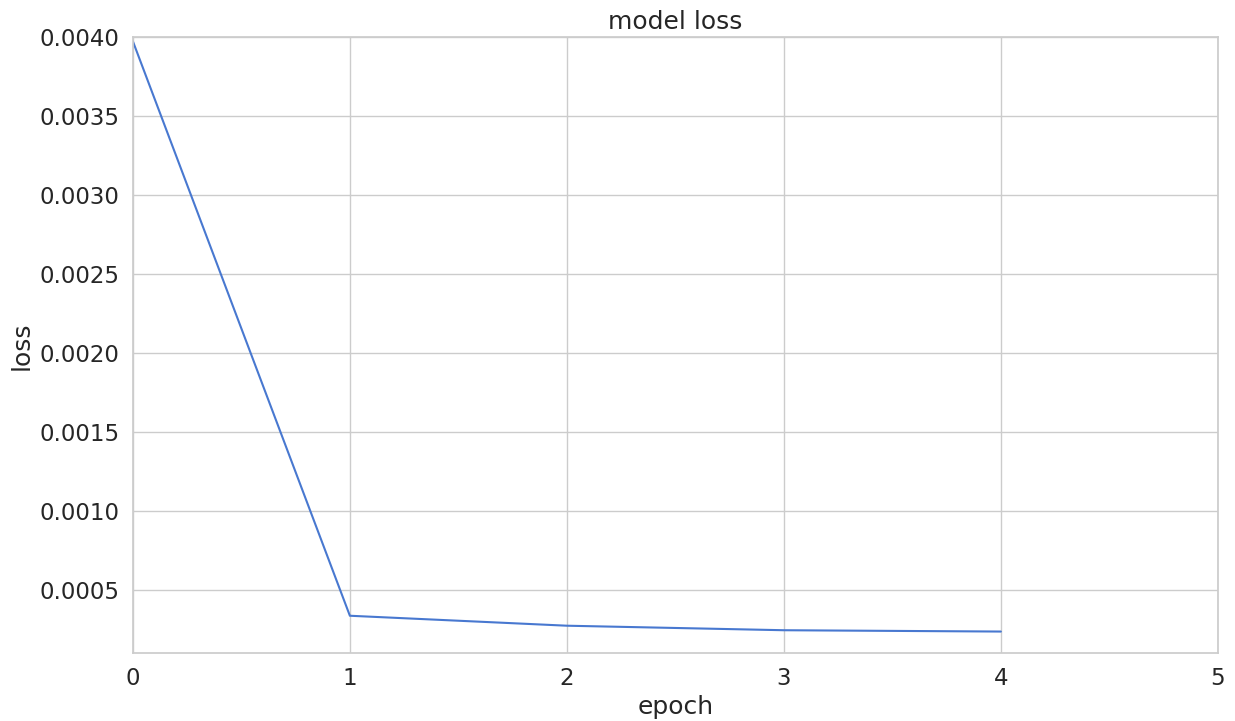
Далее определим модель автоэнкодера. У меня будет 3 слоя, на первом 16 нейронов, на втором 8, а на третьем 4.

|  |
| --- |
| # Определение автоэнкодера  class Autoencoder(nn.Module):      def \_\_init\_\_(self):          super(Autoencoder, self).\_\_init\_\_()          self.encoder = nn.Sequential(              nn.Linear(X\_train.shape[1], 16),              nn.ReLU(),              nn.Linear(16, 8),              nn.ReLU(),              nn.Linear(8, 4),  # Ближе к компактному представлению              nn.ReLU(),          )          self.decoder = nn.Sequential(              nn.Linear(4, 8),              nn.ReLU(),              nn.Linear(8, 16),              nn.ReLU(),              nn.Linear(16, X\_train.shape[1]),              nn.Sigmoid()  # Используем Sigmoid, если данные в диапазоне [0, 1]          )      def forward(self, x):          x = self.encoder(x)          x = self.decoder(x)          return x |

Инициализирую и обучаю модель, я буду использовать 5 эпох.

|  |
| --- |
| # Инициализация модели  model = Autoencoder()  criterion = nn.MSELoss() # Функция потерь. Вычисляет среднеквадратичную ошибку модели. Между предсказанными и реальными  optimizer = torch.optim.Adam(model.parameters(), lr=1e-3, weight\_decay=1e-5) # Оптимизирует параметры модели  # model.parameters() - передает веса и смещения которые будут обновляться во время обучения  # lr=1e-3 - скорость обучения  # weight\_decay=1e-05 - добавляет регуляризацию, чтобы уменьшить вероятность переобучения  # Обучение автоэнкодера  num\_epochs = 5  history = {'train\_loss': []}  for epoch in range(num\_epochs): # Каждая эпоха представляет собой проход по всем тренировачным данным      model.train()      epoch\_loss = 0      for data in train\_loader:          # Прямой проход          output = model(data)          loss = criterion(output, data) # Вычисляет функцию потерь          # Обратный проход          # Градиент - это векторы, которые указывают направление наибольшего изменения функции. В процессе обучения нейронных сетей градиенты используются для обновления параметров модели с целью минимизации ошибки.          # Градиент функции потерь показывает, как сильно функция изменяется в каждом направлении относительно параметров модели (весов).          optimizer.zero\_grad() # Обнуляет градиенты          loss.backward() # Вычисляет градиенты функции потерь относительно параметров модели          optimizer.step() # Обновляет параметры модели, используя вычисленные градиенты          epoch\_loss += loss.item()      mean\_loss = epoch\_loss / len(train\_loader)      history['train\_loss'].append(mean\_loss)      print(f'Epoch [{epoch + 1}/{num\_epochs}], Loss: {mean\_loss:.4f}') |

# Анализ результатов

Для начала убедимся, что пяти эпох достаточно для обучения модели.

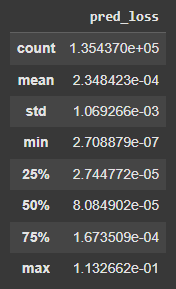
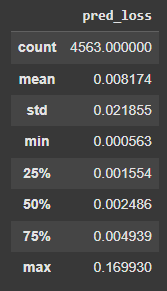
|  |
| --- |
| plt.plot(history['train\_loss'])  plt.title('model loss')  plt.ylabel('loss')  plt.xlabel('epoch')  plt.axis([0,5,0.0001,0.004])  plt.show() |

1. Ошибки модели в процессе обучения

Мы видим, что модель очень быстро обучилась. Ошибка почти нулевая, нам этого достаточно.

Так как автоэнкодер обучался на нулевом классе, это значит, что средняя ошибка на классе «1» должна быть больше чем на классе «0».

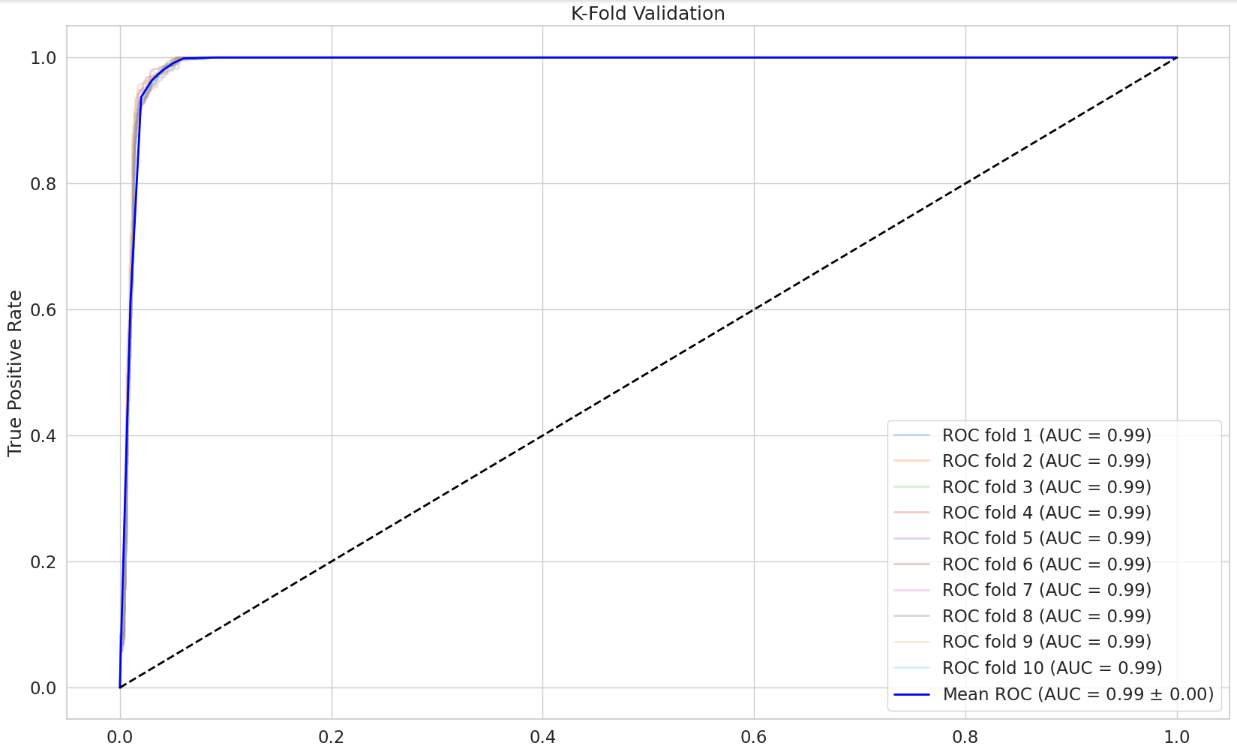
|  |
| --- |
| pred\_losses = {'pred\_loss' : []} # Ошибки  model.eval() # переводит модель в режим оценки  with torch.no\_grad():      for data in test\_loader:          inputs = data          outputs = model(inputs)          loss = criterion(outputs, inputs).data.item() # Ошибка реконструкция          pred\_losses['pred\_loss'].append(loss)  reconstructionErrorDF = pd.DataFrame(pred\_losses)  reconstructionErrorDF[48] = y\_test  # В результатах будет DataFrame, где каждый пример тестовых данных будет иметь ошибку реконструкции и соответствующую метку класса.  reconstructionErrorDF['pred\_loss'][reconstructionErrorDF[48] == 0].describe()  reconstructionErrorDF['pred\_loss'][reconstructionErrorDF[48] == 1].describe() |

1. Ошибки на нулевом классе
2. Ошибки на первом классе

Можем заметить, что средняя ошибка в классе 0 на порядок меньше, чем в классе 1.

Далее построим ROC-кривые.

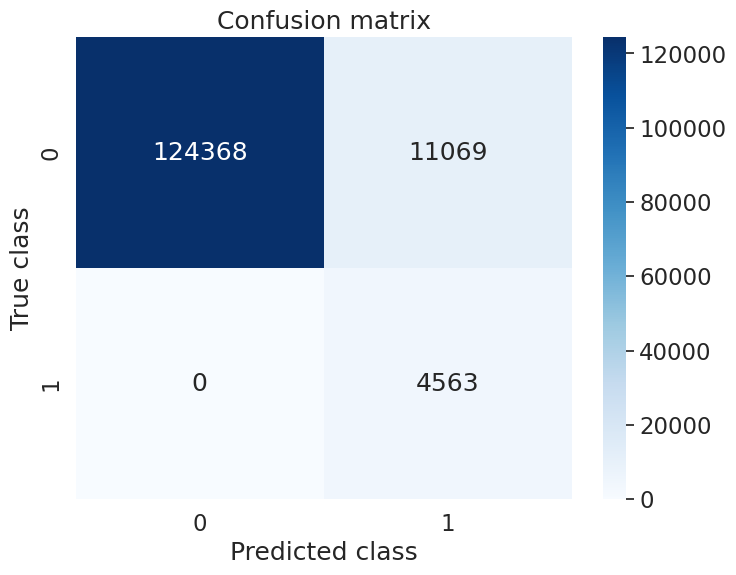
|  |
| --- |
| # Построение ROC-кривых  from sklearn.model\_selection import StratifiedKFold  cv = StratifiedKFold(n\_splits=10, shuffle=True, random\_state=42)  tprs, aucs = [], []  mean\_fpr = np.linspace(0, 1, 100)  fig = plt.figure(figsize=(20, 12), dpi=100)  i = 1  for train\_idx, test\_idx in cv.split(X\_test, y\_test):      X\_test\_fold = X\_test[test\_idx]      y\_test\_fold = y\_test[test\_idx]      # Рассчитываем ошибки реконструкции для тестовой выборки      model.eval()      reconstruction\_errors = []      with torch.no\_grad():          for data in X\_test\_fold:              data = torch.tensor(data).unsqueeze(0).float()              outputs = model(data)              loss = criterion(outputs, data).item()              reconstruction\_errors.append(loss)      # Строим ROC-кривую      fpr, tpr, thresholds = roc\_curve(y\_test\_fold, reconstruction\_errors, pos\_label=1)      roc\_auc = auc(fpr, tpr)      tprs.append(np.interp(mean\_fpr, fpr, tpr))      tprs[-1][0] = 0.0      aucs.append(roc\_auc)      plt.plot(fpr, tpr, lw=2, alpha=0.3, label=f'ROC fold {i} (AUC = {roc\_auc:.2f})')      i += 1  # Средние значения и доверительный интервал  mean\_tpr = np.mean(tprs, axis=0)  mean\_tpr[-1] = 1.0  mean\_auc = auc(mean\_fpr, mean\_tpr)  std\_auc = np.std(aucs)  plt.plot(mean\_fpr, mean\_tpr, color='blue',           label=r'Mean ROC (AUC = %0.2f $±$ %0.2f)' % (mean\_auc, std\_auc),           lw=2, alpha=1)  plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='black')  plt.xlabel('False Positive Rate')  plt.ylabel('True Positive Rate')  plt.title('K-Fold Validation')  plt.legend(loc="lower right")  plt.show() |

1. ROC-кривые

Значение 0,99 означает, что модель отлично предсказывает.

Выведем Confusion matrix.

|  |
| --- |
| y\_pred = [1 if e > threshold else 0 for e in reconstructionErrorDF.pred\_loss.values]  conf\_matrix = confusion\_matrix(reconstructionErrorDF[48], y\_pred)  plt.figure(figsize=(8, 6))  sns.heatmap(conf\_matrix, xticklabels=[0, 1], yticklabels=[0, 1], annot=True, fmt="d",              cmap=plt.cm.get\_cmap('Blues'));  plt.title("Confusion matrix")  plt.ylabel('True class')  plt.xlabel('Predicted class')  plt.show()  print(f'Точность предсказания вредоносного трафика - {(conf\_matrix[1, 1]) / float(conf\_matrix[1, 0] + conf\_matrix[1, 1])}')  print(f'Точность предсказания нормального трафика - {(conf\_matrix[0, 0]) / float(conf\_matrix[0, 1] + conf\_matrix[0, 0])}') |

1. Confusion matrix 
2. Абсолютные значения AUC

Автоэнкодер с высокой точностью определяет вредоносный трафик.

# Выводы

В ходе выполнения лабораторной работы удалось достичь следующих результатов:

1. Реализована модель автоэнкодера, состоящая из трехслойного кодировщика и декодировщика.
2. Модель была обучена на данных нормального интернет-трафика и показала способность эффективно выявлять аномалии.
3. Средняя ошибка реконструкции для нормального трафика значительно ниже, чем для вредоносного, что подтверждает качество обучения.
4. Построенные ROC-кривые демонстрируют высокую точность модели с AUC, равной 0.99.
5. Конфигурация автоэнкодера обеспечивает высокую точность в определении вредоносного трафика, подтвержденную результатами Confusion Matrix.

Эти результаты показывают, что модель автоэнкодера может быть успешно использована для задач обнаружения аномалий в интернет-трафике.

# Список литературы

1. Официальная документация PyTorch: <https://pytorch.org>
2. Исходные данные: unsw-nb15, доступный по адресу <https://www.kaggle.com/datasets/mrwellsdavid/unsw-nb15>
3. Chollet F. “Deep Learning with Python”. Manning Publications, 2018.