

## Цель работы

Обучить нейронную сеть для выполнения задачи регрессии и классификации.

## Вариант 1

В первом этапе работы был выбран набор данных сдачи экзаменов студентами. В этом датасете содержится информация о поле, расе, уровне образования родителей, был ли перед экзаменом приём пищи, пройден ли подготовительный курс и какие оценки получены по математике, письму и чтению.

[Ссылка на набор данных.](#)

Во втором этапе работы требовалось обучить модель на наборе данных сердечных болезней. Датасет содержит информацию о возрасте, поле, типе боли, артериальном давлении, холестерине, уровне сахара в крови, ЭКГ, частоте сердечных сокращений, стенокардии, депрессии ST, наклоне ST и есть ли болезнь.

## Ход работы

### Часть 1

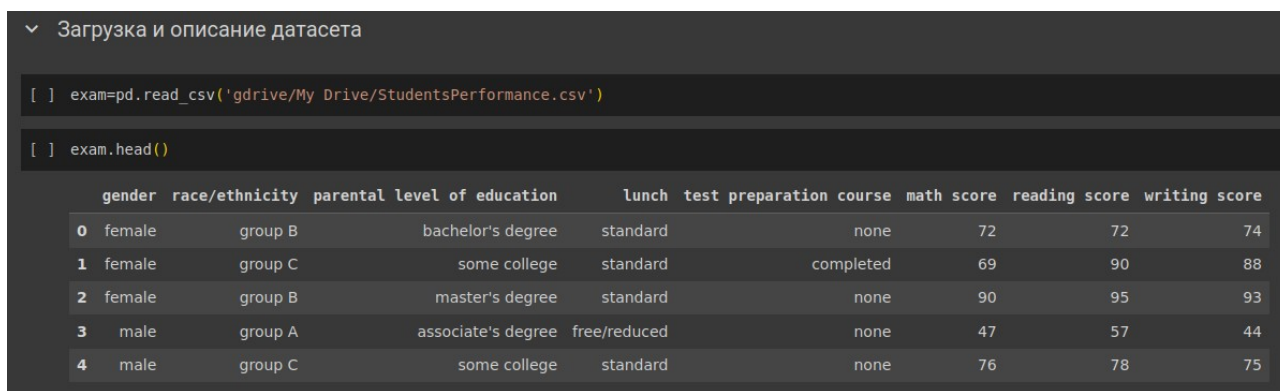
Был произведен импорт всех необходимых библиотек. Результат представлен на рисунке 1.

```
from google.colab import drive
drive.mount('/content/gdrive')
import pandas as pd
from sklearn.metrics import roc_curve, auc
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from sklearn.model_selection import train_test_split

Mounted at /content/gdrive
```

Рисунок 1 - Импортируемые библиотеки

Был загружен и прочитан исходный файл. По каждому столбцу было указано краткое описание. На рисунке 2 представлен результат выполненных действий.



```

[ ] exam=pd.read_csv('gdrive/My Drive/StudentsPerformance.csv')

[ ] exam.head()

```

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	female	group B	bachelor's degree	standard	none	72	72	74
1	female	group C	some college	standard	completed	69	90	88
2	female	group B	master's degree	standard	none	90	95	93
3	male	group A	associate's degree	free/reduced	none	47	57	44
4	male	group C	some college	standard	none	76	78	75

Рисунок 2 - Загрузка и описание датасета

Данный датасет содержит информацию о результатах сдачи экзаменов студентами.

Описание столбцов:

1. gender - пол студента
2. race/ethnicity - расу/этнос разбитые по группам
3. parental level of education - уровень образования родителей
4. lunch - какой завтрак был в день экзамена
5. test preparation course - был ли пройден подготовительный курс
6. math score - оценка за экзамен по математике
7. reading score - оценка за экзамен по чтению
8. writing score - оценка за экзамен по письму

Была выведена информация о датасете. Результат представлен на рисунке 3.

```

v  Информация о датасете

Была отображена информация о столбцах.

[ ] exam.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
#   Column                                     Non-Null Count  Dtype
---  -
0   gender                                     1000 non-null   object
1   race/ethnicity                             1000 non-null   object
2   parental level of education               1000 non-null   object
3   lunch                                      1000 non-null   object
4   test preparation course                   1000 non-null   object
5   math score                                1000 non-null   int64
6   reading score                             1000 non-null   int64
7   writing score                              1000 non-null   int64
dtypes: int64(3), object(5)
memory usage: 62.6+ KB

```

Рисунок 3 - Информация о датасете

В данном датасете 1000 записей и 8 столбцов. Можно заметить, что в столбцах нет пропущенных значений. Типы данных для каждого столбца указаны верно.

Было выведено подробное описание числовых значений набора данных. Результат представлен на рисунке 4.

Значения числовых столбцов датасета

```
[ ] exam.describe()
```

	math score	reading score	writing score
count	1000.00000	1000.000000	1000.000000
mean	66.08900	69.169000	68.054000
std	15.16308	14.600192	15.195657
min	0.00000	17.000000	10.000000
25%	57.00000	59.000000	57.750000
50%	66.00000	70.000000	69.000000
75%	77.00000	79.000000	79.000000
max	100.00000	100.000000	100.000000

Рисунок 4 - Значения числовых столбцов датасета

На основе таблицы данных можно сказать, что средняя оценка по математике – 66. Диапазон оценок от 0 до 100. Также можно выяснить максимальные, средние и минимальные значения для всех остальных экзаменов.

Был произведён поиск явных дубликатов. Результат представлен на рисунке 5.

```
[ ] exam.duplicated().sum()

0
```

Рисунок 5 - Поиск явных дубликатов

Явных дубликатов не обнаружено.

Была произведена замена строковых значений на числовые, корректировка типов данных и поиск неявных дубликатов. Результат представлен на рисунке 6.

```
▼ Замена строковых значений на числовые, корректировка типов данных и поиск неявных дубликатов.

[ ] print(exam['gender'].unique())
exam['gender'] = exam['gender'].replace(to_replace=['female', 'male'], value=[0, 1])
exam['gender'] = exam['gender'].astype(int)
print(exam['gender'].unique())

['female' 'male']
[0 1]

[ ] print(exam['race/ethnicity'].unique())
exam['race/ethnicity'] = exam['race/ethnicity'].replace(to_replace=['group A', 'group B', 'group C', 'group D', 'group E'], value=[0, 1, 2, 3, 4])
exam['race/ethnicity'] = exam['race/ethnicity'].astype(int)
print(exam['race/ethnicity'].unique())

['group B' 'group C' 'group A' 'group D' 'group E']
[1 2 0 3 4]

[ ] print(exam['parental level of education'].unique())
exam['parental level of education'] = exam['parental level of education'].replace(to_replace=["bachelor's degree", 'some college', "master's degree", "associate's degree", 'high school', 'some high school'], value=[0, 1, 2, 3, 4, 5])
exam['parental level of education'] = exam['parental level of education'].astype(int)
print(exam['parental level of education'].unique())

["bachelor's degree" 'some college' "master's degree" "associate's degree"
'high school' 'some high school']
[0 1 2 3 4 5]

[ ] print(exam['lunch'].unique())
exam['lunch'] = exam['lunch'].replace(to_replace=['standard', 'free/reduced'], value=[0, 1])
exam['lunch'] = exam['lunch'].astype(int)
print(exam['lunch'].unique())

['standard' 'free/reduced']
[0 1]

[ ] print(exam['test preparation course'].unique())
exam['test preparation course'] = exam['test preparation course'].replace(to_replace=['none', 'completed'], value=[0, 1])
exam['test preparation course'] = exam['test preparation course'].astype(int)
print(exam['test preparation course'].unique())

['none' 'completed']
[0 1]
```

Рисунок 6 - Замена строковых значений на числовые, корректировка типов данных и поиск неявных дубликатов.

Заменённые на числовые строковые значения отображаются корректно. Неявных дубликатов не обнаружено.

Были удалены лишние столбцы. Результат представлен на рисунке 7.

```
▼ Удаление ненужных для предсказания столбцов

Так как в ходе работы будут предсказываться оценки по экзамену по математике, оценки по другим экзаменам можно убрать из датасета.

[ ] exam = exam.drop('reading score', axis=1)
exam = exam.drop('writing score', axis=1)
```

Рисунок 7 - Удаление ненужных для предсказания столбцов

Для проверки корректности изменённого состояния датасета был произведён его вывод. Результат представлен на рисунке 8.

```
▼ Вывод предобработанного датасета

[ ] exam.head()

   gender  race/ethnicity  parental level of education  lunch  test preparation course  math score
0       0              1                0      0              0           72
1       0              2                1      0              1           69
2       0              1                2      0              0           90
3       1              0                3      1              0           47
4       1              2                1      0              0           76

[ ] exam.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 6 columns):
#   Column                                Non-Null Count  Dtype
---  ---                                -
0   gender                                1000 non-null   int64
1   race/ethnicity                        1000 non-null   int64
2   parental level of education           1000 non-null   int64
3   lunch                                 1000 non-null   int64
4   test preparation course               1000 non-null   int64
5   math score                            1000 non-null   int64
dtypes: int64(6)
memory usage: 47.0 KB
```

Рисунок 8 - Вывод предобработанного датасета

Датафрейм и таблица выше показывают, что теперь все данные датасета являются числовыми, а также, что удалены столбцы с оценками по письму и чтению.

Была произведена стандартизация данных и разделение данных. Результат представлен на рисунке 9.

```
▼ Выбор целевой переменной и стандартизация данных

[ ] y = exam['math score']
    x = exam.drop('math score', axis=1)

Целевая переменная в данном случае - оценка за экзамен по математике. Она и будет прогнозироваться.

Так как данные в датасете имеют разный диапазон, их нужно стандартизировать.

[ ] scaler = StandardScaler() # создание
    #StandardScaler используется для изменения размера распределения значений так, чтобы среднее значение наблюдаемых значений было равно 0, а стандартное отклонение - 1.
    scaler.fit(x) # обучение
    x = scaler.transform(x) # преобразование
```

Рисунок 9 - Выбор целевой переменной и стандартизация данных

Целевая переменная в данном случае - оценка за экзамен по математике. Она и будет прогнозироваться.

Данные были разбиты на тренировочные и тестовые. Результат представлен на рисунке 10.

## 2. Разделить набор на обучающие и валидационные данные.

Данные требуется случайно разделить на обучающую и тестовую выборки. 20% данных будут являться тестовыми, а остальные 80% - обучающими.

```
[ ] x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=33)
```

Рисунок 10 - Разделение набора на обучающие и валидационные данные

Была создана модель, состоящая из трёх слоёв: входного (500 нейронов), скрытого (1000 нейронов) и выходного (1 нейрон). Результат представлен на рисунке 11.

## 3. Создать последовательную модель с помощью Sequential, добавить слои с помощью Dense. Поэкспериментировать с количеством скрытых слоёв. На выходе - 1 нейрон, без функции активации.

```
[ ] model = tf.keras.Sequential([
    tf.keras.layers.Dense(500, activation='relu', input_shape=(x_train.shape[1],)),
    tf.keras.layers.Dense(1000, activation='relu'),
    tf.keras.layers.Dense(1)
])
```

Рисунок 11 - Создание модели

В созданной модели присутствует 3 слоя: 1 входной, 1 скрытый и 1 выходной.

В нашем примере мы использовали слой **Dense** – полносвязный слой (каждый нейрон данной слоя связан с каждым нейроном предыдущего/следующего слоя).

- Первый параметр слоя – **units**, данный параметр задаёт кол-во нейронов слое, в данном случае у первого слоя 500 нейронов, у второго 1000, у третьего 1.
- Второй параметр **activation** – параметр, задающий функцию активации нейронов, у первого и второго слоя функция активации – Relu.
- Крайний параметр **input\_shape** – параметр, отвечающий за размер входных данных, в нашем случае на вход каждого нейрона первого слоя подаётся массив размера (800, 5). Наличие данного параметра указывает на то, что слой является входным, следовательно из-за наличия всего трёх слоёв, третий слой – выходной.

Была выведена общая информация о модели. Результат представлен на



рисунке 12.

```
✓ 4. Вывести model.summary() и прокомментировать результат.

[ ] print(model.summary())

Model: "sequential_25"

Layer (type)                 Output Shape                 Param #
=====
dense_70 (Dense)             (None, 500)                 3000
dense_71 (Dense)             (None, 1000)               501000
dense_72 (Dense)             (None, 1)                  1001
=====
Total params: 505001 (1.93 MB)
Trainable params: 505001 (1.93 MB)
Non-trainable params: 0 (0.00 Byte)

None
```

Рисунок 12 - Описание модели и слоёв

- Model - название модели
- Layer - название и тип слоя
- Output Shape - размер данных на выход из слоя
- Param - количество параметров

Общее количество параметров = 505001.

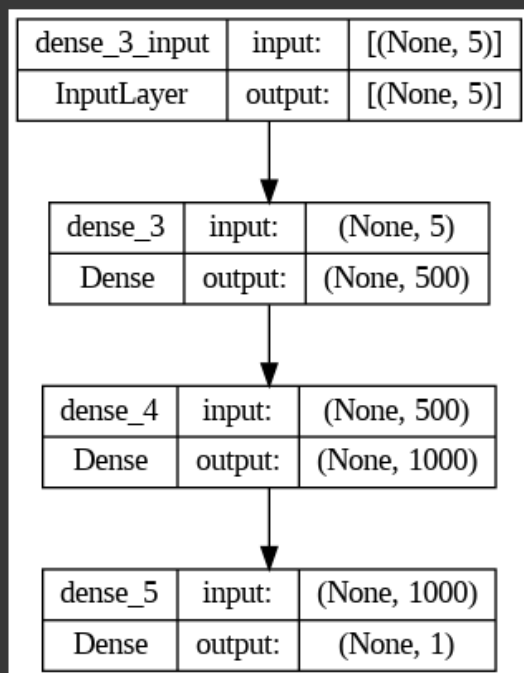
Количество тренируемых параметров = 505001.

Для данной модели было создано изображение её архитектуры. Результат представлен на рисунке 13.



## Визуализация архитектуры нейронной сети

```
[72] plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
```



```
[61] visualkeras.layered_view(model)
```



Рисунок 13 - Архитектура НС

Были заданы необходимые параметры модели. Результат представлен на рисунке 14.

5. Задать параметры в compile (в качестве loss можно использовать mse, в качестве метрики - mae. Можете также поэкспериментировать с моделью, метриками и выбрать наиболее подходящую.

Были заданы обучающие параметры

```
[47] model.compile(loss="mse", optimizer="adam", metrics=["mae"])
```

Рисунок 14 - Параметры модели

- Параметр **loss** – ошибка между результатом, выдаваемым моделью и реальным значением, по сути - это цель, которую модель пытается свести к минимуму, например, categorical\_crossentropy или mse.

- **optimizer** – алгоритм по которому будет проводится обновление синапсов нейронов,
- **metrics** – метрики, которые будут выводиться при обучении модели, нужны для оценки качества обучения

Модель была обучена. Результат представлен на рисунке 15.

```

✓ 6. Обучить модель с помощью fit.

Было произведено обучение модели на тестовой выборке.

[48] history = model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))

Epoch 1/5
25/25 [=====] - 1s 11ms/step - loss: 3154.8743 - mae: 53.3381 - val_loss: 1344.1492 - val_mae: 34.0479
Epoch 2/5
25/25 [=====] - 0s 7ms/step - loss: 435.4438 - mae: 16.5097 - val_loss: 225.4020 - val_mae: 12.0873
Epoch 3/5
25/25 [=====] - 0s 6ms/step - loss: 224.2424 - mae: 12.1062 - val_loss: 205.7027 - val_mae: 11.6318
Epoch 4/5
25/25 [=====] - 0s 6ms/step - loss: 201.7462 - mae: 11.4155 - val_loss: 187.5069 - val_mae: 11.0226
Epoch 5/5
25/25 [=====] - 0s 7ms/step - loss: 196.0560 - mae: 11.2957 - val_loss: 195.8643 - val_mae: 11.3040

• x_train – Numpy массив, содержащий в себе обучающую выборку
• y_train – Numpy массив, содержащий разметку данных для x_train (верные ответы), (если несколько выходов - список массивов)
• epoch – кол-во эпох для обучения модели. В данном случае, количество эпох равно пяти.

Были выведены метрики модели.

[49] model.evaluate(x_test, y_test)

7/7 [=====] - 0s 3ms/step - loss: 195.8643 - mae: 11.3040
[195.8642578125, 11.304046630859375]

```

Рисунок 15 - Обучение модели

Loss рассчитывается по формуле mse. Среднеквадратичная ошибка (Mean Squared Error) – Среднее арифметическое (Mean) квадратов разностей между предсказанными и реальными значениями Модели (Model) Машинного обучения.

Метрика mae(Средняя абсолютная ошибка) рассчитывается как среднее абсолютных разностей между целевыми значением и значением, предсказанным моделью на данном обучающем примере в процессе обучения.

Было выполнено предсказание на тестовых данных. Результат представлен на рисунке 16.

## ✓ 7. Выполнить предсказание на валидационных данных.

Была произведена проверка работоспособности модели, используя метод `predict()`.

```
[50] y_pred = model.predict(x_test)
```

```
7/7 [=====] - 0s 2ms/step
```

Рисунок 16 - Предсказание на тестовых данных

Был создан датафрейм с истинными и предсказанными значениями. Результат представлен на рисунке 17.

## ✓ 8. Создать датафрейм с истинными и предсказанными значениями.

Был создан датафрейм реальных и предсказанных данных

```
[51] result_df = pd.DataFrame({'True_Score': y_test, 'Predicted_Score': y_pred.flatten()})  
print(result_df)
```

	True_Score	Predicted_Score
286	97	78.528069
402	49	56.215042
253	80	67.226143
4	76	64.029045
372	74	73.412254
..	...	...
358	59	60.139679
626	69	56.030937
207	81	80.180389
714	60	65.699844
122	88	68.263100

```
[200 rows x 2 columns]
```

Рисунок 17 - Датафрейм с оригинальными значениями и предсказанными

По таблице видно, что значения отклоняются в среднем на 10%.

Был построен график правильных и предсказанных значений. Результат представлен на рисунке 18.

9. Построить график правильных и предсказанных значений (как в ЛРН№3 курса Введение в анализ данных).

```
[671] plt.figure(figsize=(10, 5))
plt.scatter(result_df['True_Score'][:100], result_df['Predicted_Score'][:100])
plt.plot([min(result_df['True_Score']), max(result_df['True_Score'])],
         [min(result_df['True_Score']), max(result_df['True_Score'])],
         color='red', linestyle='--', label='Линия идеального совпадения')
plt.title('Истинные и предсказанные значения')
plt.xlabel('Истинные значения')
plt.ylabel('Предсказанные значения')
```

Text(0, 0.5, 'Предсказанные значения')

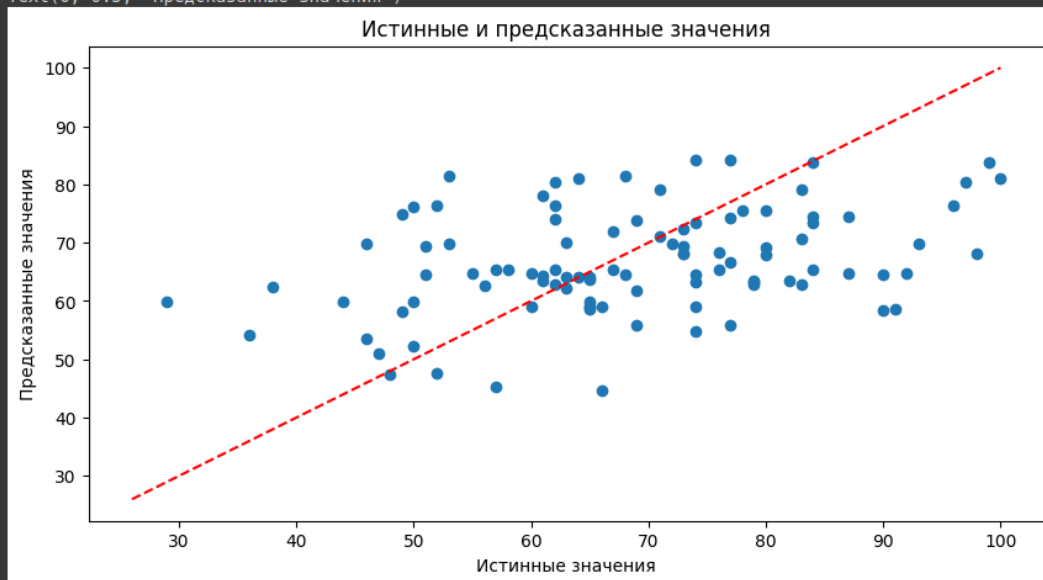


Рисунок 18 - График распределения значений

На основе полученного графика можно сделать вывод что нейронная сеть достаточно точно предсказывает оценку по экзамену.

Был построен график ошибок на валидационном наборе по эпохам обучения. Результат представлен на рисунках 19 – 20.

✓ 10. Построить график ошибок на обучающем и валидационном наборе по эпохам обучения.

```
plt.figure(figsize=(5,5))
plt.plot(history.history['mae'], label='mae')
plt.plot(history.history['val_mae'], label = 'val_mae')
plt.xlabel('Epoch')
plt.ylabel('mae')
plt.legend(loc='lower right')
```

<matplotlib.legend.Legend at 0x7cba84642350>

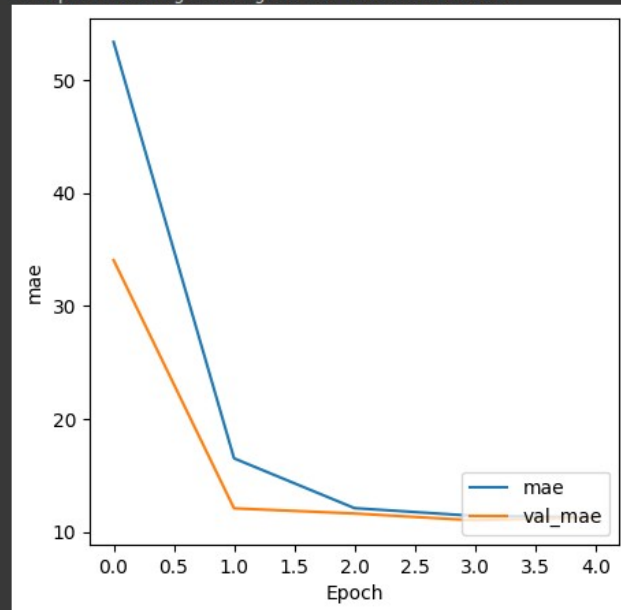


Рисунок 19 - График mae/Epoch

```
[54] plt.figure(figsize=(5,5))
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label = 'val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='lower right')
```

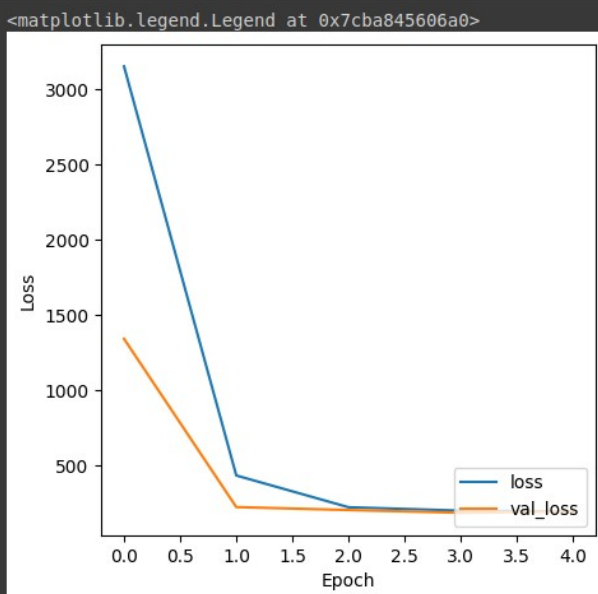


Рисунок 20 - График Loss/Epoch

На полученных графиках можно заметить, что с увеличением количества эпох - метрики Loss и MAE начинают расти медленнее. Оптимальное количество эпох – 3.

## Часть 2

Был произведён импорт датасета сердечных болезней и его предобработка. Результат представлен на рисунках 21 – 27.

Требуется провести предобработку данных. Вначале были выписаны все столбцы датасета и указана информация, что означает каждый из них.

```
[ ] heart = pd.read_csv('gdrive/My Drive/1heart.csv')
heart.head()
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140.0	289	0	Normal	172.0	N	0.0	Up	0
1	49	F	NAP	160.0	180	0	Normal	156.0	N	1.0	Flat	1
2	37	M	ATA	130.0	283	0	ST	98.0	N	0.0	Up	0
3	48	F	ASY	138.0	214	0	Normal	108.0	Y	1.5	Flat	1
4	54	M	NAP	150.0	195	0	Normal	122.0	N	0.0	Up	0

Рисунок 21 - Импорт датасета

Датасет содержит информацию о сердечных болезнях

1. возраст
2. пол
3. тип боли в груди (4 значения [TA: типичная стенокардия, ATA: атипичная стенокардия, NAP: неангинальная боль, ASY: бессимптомная])
4. артериальное давление в покое
5. холестерин сыворотки в мг/дл
6. уровень сахара в крови натощак > 120 мг/дл
7. ЭКГ в покое: результаты электрокардиограммы в покое (норма: нормальная, ST: аномалия ST-T LVN: гипертрофия)
8. MaxHR: максимальная достигнутая частота сердечных сокращений  
\*[Числовое значение от 60 до 202]\*
9. стенокардия, вызванная физической нагрузкой (да, нет)
10. oldpeak = депрессия ST, вызванная физической нагрузкой, по сравнению с состоянием покоя
11. наклон пикового сегмента ST при нагрузке (Вверх: восходящий, Плоский: плоский, Вниз: нисходящий)
12. HeartDisease: выходной класс [1: болезнь сердца, 0: нормальный]



```
[ ] heart.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 925 entries, 0 to 924
Data columns (total 12 columns):
#   Column                Non-Null Count  Dtype  
---  -
0   Age                    925 non-null    int64  
1   Sex                    925 non-null    object  
2   ChestPainType          925 non-null    object  
3   RestingBP              923 non-null    float64 
4   Cholesterol            924 non-null    object  
5   FastingBS              925 non-null    int64  
6   RestingECG            924 non-null    object  
7   MaxHR                  924 non-null    float64 
8   ExerciseAngina         925 non-null    object  
9   Oldpeak                925 non-null    float64 
10  ST_Slope               925 non-null    object  
11  HeartDisease           925 non-null    int64  
dtypes: float64(3), int64(3), object(6)
memory usage: 86.8+ KB

В датасете 925 записей и 12 столбцов.

[ ] heart.describe()

      Age  RestingBP  FastingBS  MaxHR  Oldpeak  HeartDisease
count  925.000000  923.000000  925.000000  924.000000  925.000000  925.000000
mean    53.451892  132.442037    0.231351  136.979437    0.882162    0.550270
std     9.453069   18.464707    0.421925   25.507091    1.064803    0.497736
min     28.000000    0.000000    0.000000   60.000000   -2.600000    0.000000
25%    47.000000  120.000000    0.000000  120.000000    0.000000    0.000000
50%    54.000000  130.000000    0.000000  138.000000    0.500000    1.000000
75%    60.000000  140.000000    0.000000  156.000000    1.500000    1.000000
max     77.000000  200.000000    1.000000  202.000000    6.200000    1.000000
```

Рисунок 22 - Предобработка датасета (1/6)

В датасете 925 записей и 12 столбцов.

На основе таблицы данных можно сказать, что средний возраст пациентов - 53 года, немного больше половины имеют заболевание сердца. Также можно выяснить максимальные, средние и минимальные значения для таких показателей, как возраст, артериальное давление в покое, уровень сахара в крови натощак, максимальная достигнутая частота сердечных сокращений, депрессия ST, вызванная физической нагрузкой, по сравнению с состоянием покоя.

```
[ ] heart.isna().sum() #подсчет количества пропусков
```

Age	0
Sex	0
ChestPainType	0
RestingBP	2
Cholesterol	1
FastingBS	0
RestingECG	1
MaxHR	1
ExerciseAngina	0
Oldpeak	0
ST_Slope	0
HeartDisease	0
dtype: int64	

В датасете в некоторых колонках присутствуют пропуски. Так как все данные, замененные 0 в числовых ячейках повлияют на кластеризацию. Производится удаление всех пустых записей. А затем подсчет количества пропусков для проверки удаления.

```
[ ] heart.dropna(inplace=True)
```

#Параметр inplace определяет, нужно ли вносить изменения в существующий датафрейм или нужно вернуть его копию.  
#По умолчанию значение inplace – False (возвращается копия). У нас inplace=True для внесения изменений на месте

```
[ ] heart.isna().sum()
```

Age	0
Sex	0
ChestPainType	0
RestingBP	0
Cholesterol	0
FastingBS	0
RestingECG	0
MaxHR	0
ExerciseAngina	0
Oldpeak	0
ST_Slope	0
HeartDisease	0
dtype: int64	

Рисунок 23 - Предобработка датасета (2/6)

В датасете в некоторых колонках присутствуют пропуски. Так как все данные, замененные 0 в числовых ячейках повлияют на кластеризацию. Производится удаление всех пустых записей. А затем подсчет количества пропусков для проверки удаления.

Все пропуски данных были успешно удалены.

```
[ ] heart.duplicated().sum() # подсчет количества дубликатов

3

Так как обнаружены дубликаты, они удаляются и осуществляется проверка удаления.

[ ] heart = heart.drop_duplicates() # удаление дубликатов
heart.duplicated().sum()

0

Все явные дубликаты устранены.

[ ] heart.reset_index(drop=True, inplace=True) # обновление индексации; аргумент drop, чтобы не создавать столбец index

Для каждого текстового столбца осуществляется поиск уникальных значений и при наличии дубликатов устранение их.

[ ] print(heart['Sex'].unique()) # Поиск уникальных значений в столбце "Sex"

['M' 'F' 'Ma']

Обнаружены дубликаты. Необходимо заменить 'Ma' на 'M'. Также осуществляется проверка.

[ ] heart['Sex'].replace('Ma', 'M', inplace=True)
print(heart['Sex'].unique()) # Поиск уникальных значений в столбце "Sex"

['M' 'F']
<ipython-input-639-edb97012f2fc>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy
heart['Sex'].replace('Ma', 'M', inplace=True)

Успешное устранение.
```

Рисунок 24 - Предобработка датасета (3/6)

```
[ ] print(heart['ChestPainType'].unique()) # Поиск уникальных значений в столбце "ChestPainType"

['ATA' 'NAP' 'ASY' 'TA']

Дубликаты не обнаружены.

[ ] print(heart['Cholesterol'].unique()) # Поиск уникальных значений в столбце "Cholesterol"

['289' '180' '283' '214' '195' '339' '237' '208' '207' '284' '211' '164'
'204' '234' '273' '196' '248' '267' '223' '184' '201' '288' '209' '260'
'468' '188' '518' '167' '224' '172' '186' '254' '306' '250' '177' '227'
'230' '294' '264' '259' '175' '318' '216' '340' '233' '205' '245' '194'
'270' '213' '365' '342' '253' '277' '202' '297' '225' '246' '412' '265'
'215' '182' '218' '268' '163' '529' '100' '206' '238' '139' '263' '291'
'229' '307' '210' '329' '147' '85' '269' '275' '179' '392' '466' '129'
'241' '255' '276' '282' '338' '160' '156' '272' '240' '393' '161' '228'
'292' '388' '166' '247' '331' '341' '243' '279' '198' '249' '168' '603'
'159' '190' '185' '290' '212' '231' '222' '235' '320' '187' '266' '287'
'404' '312' '251' '328' '285' '280' '192' '193' '308' '219' '257' '132'
'226' '217' '303' '298' '256' '117' '295' '173' '315' '281' '309' '200'
'336' '355' '326' '171' '491' '271' '274' '394' '221' '126' '305' '220'
'242' '347' '344' '358' '169' '181' '0' '236' '203' '153' '316' '311'
'252' '458' '384' '258' '349' '142' '197' '113' '261' '310' '232' '110'
'123' '170' '369' '152' '244' '165' '337' '300' '333' '385' '322' '564'
'239' '293' '407' '149' '199' '417' '178' '319' '354' '330' '302' '313'
'141' '327' '304' '286' '360' '262' '325' '299' '409' '174' '183' '321'
'353' '335' '278' '157' '176' '131' 'a241']

Так как это числа, данные не нуждаются в вычислении дубликатов. Однако есть числа с буквами, которые стоит заменить. Также
сделана проверка для того, чтобы убедиться в корректной замене.
```

Рисунок 25 - Предобработка датасета (4/6)

```
[ ] heart['Cholesterol'].replace('a241', '241', inplace=True)
print(heart['Cholesterol'].unique()) # Поиск уникальных значений в столбце "Cholesterol"

['289' '180' '283' '214' '195' '339' '237' '208' '207' '284' '211' '164'
 '204' '234' '273' '196' '248' '267' '223' '184' '201' '288' '209' '260'
 '468' '188' '518' '167' '224' '172' '186' '254' '306' '250' '177' '227'
 '230' '294' '264' '259' '175' '318' '216' '340' '233' '205' '245' '194'
 '270' '213' '365' '342' '253' '277' '202' '297' '225' '246' '412' '265'
 '215' '182' '218' '268' '163' '529' '100' '206' '238' '139' '263' '291'
 '229' '307' '210' '329' '147' '85' '269' '275' '179' '392' '466' '129'
 '241' '255' '276' '282' '338' '160' '156' '272' '240' '393' '161' '228'
 '292' '388' '166' '247' '331' '341' '243' '279' '198' '249' '168' '603'
 '159' '190' '185' '290' '212' '231' '222' '235' '320' '187' '266' '287'
 '404' '312' '251' '328' '285' '280' '192' '193' '308' '219' '257' '132'
 '226' '217' '303' '298' '256' '117' '295' '173' '315' '281' '309' '200'
 '336' '355' '326' '171' '491' '271' '274' '394' '221' '126' '305' '220'
 '242' '347' '344' '358' '169' '181' '0' '236' '203' '153' '316' '311'
 '252' '458' '384' '258' '349' '142' '197' '113' '261' '310' '232' '110'
 '123' '170' '369' '152' '244' '165' '337' '300' '333' '385' '322' '564'
 '239' '293' '407' '149' '199' '417' '178' '319' '354' '330' '302' '313'
 '141' '327' '304' '286' '360' '262' '325' '299' '409' '174' '183' '321'
 '353' '335' '278' '157' '176' '131']

<ipython-input-642-c4770024f61a>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy
heart['Cholesterol'].replace('a241', '241', inplace=True)

Неявные дубликаты устранены.

[ ] print(heart['RestingECG'].unique()) # Поиск уникальных значений в столбце "RestingECG"

['Normal' 'ST' 'LVN']

Дубликаты не обнаружены.

[ ] print(heart['ExerciseAngina'].unique()) # Поиск уникальных значений в столбце "ExerciseAngina"

['N' 'Y']

Дубликаты не обнаружены.
```

Рисунок 26 - Предобработка датасета (5/6)

```
[ ] print(heart['ST_Slope'].unique()) # Поиск уникальных значений в столбце "ST_Slope"

['Up' 'Flat' 'Down' 'UP']

Обнаружен дубликат. Нужно 'UP' заменить на 'Up'. После замены осуществляется проверка.

▶ heart['ST_Slope'].replace('UP', 'Up', inplace=True)
print(heart['ST_Slope'].unique()) # Поиск уникальных значений в столбце "ST_Slope"

['Up' 'Flat' 'Down']

Дубликат устранен.

Ранее было замечено несоответствие данных колонки "Cholesterol" типу текстовому. Соответственно требуется изменить его на float.
После изменения производится проверка.

[ ] heart['Cholesterol'] = heart['Cholesterol'].astype(float)
heart['Cholesterol'].dtypes

dtype('float64')

Тип успешно изменен.
```

Рисунок 27 - Предобработка датасета (6/6)

Была произведена замена строковых значений на числовые. Результат представлен на рисунках 28 – 29.



```

  ✓ Замена строковых значений на числовые.

[ ] heart.head()

   Age Sex ChestPainType RestingBP Cholesterol FastingBS RestingECG MaxHR ExerciseAngina Oldpeak ST_Slope HeartDisease
0   40  M      ATA      140.0      289.0         0     Normal   172.0         N      0.0      Up         0
1   49  F      NAP      160.0      180.0         0     Normal   156.0         N      1.0      Flat        1
2   37  M      ATA      130.0      283.0         0         ST    98.0         N      0.0      Up         0
3   48  F      ASY      138.0      214.0         0     Normal   108.0         Y      1.5      Flat        1
4   54  M      NAP      150.0      195.0         0     Normal   122.0         N      0.0      Up         0

[ ] print(heart['Sex'].unique())
heart['Sex'] = heart['Sex'].replace(to_replace=['M', 'F'], value=[0, 1]) # замена числовых значений на текстовые
print(heart['Sex'].unique())

['M' 'F']
[0 1]

[ ] print(heart['ChestPainType'].unique())
heart['ChestPainType'] = heart['ChestPainType'].replace(to_replace=['ATA', 'NAP', 'ASY', 'TA'], value=[0, 1, 2, 3]) # замена числовых значений на текстовые
print(heart['ChestPainType'].unique())

['ATA' 'NAP' 'ASY' 'TA']
[0 1 2 3]

[ ] print(heart['RestingECG'].unique())
heart['RestingECG'] = heart['RestingECG'].replace(to_replace=['Normal', 'ST', 'LVH'], value=[0, 1, 2]) # замена числовых значений на текстовые
print(heart['RestingECG'].unique())

['Normal' 'ST' 'LVH']
[0 1 2]

```

Рисунок 28 - Замена строковых значений на числовые (1/2)

```

[ ] print(heart['ExerciseAngina'].unique())
heart['ExerciseAngina'] = heart['ExerciseAngina'].replace(to_replace=['N', 'Y'], value=[0, 1]) # замена числовых значений на текстовые
print(heart['ExerciseAngina'].unique())

['N' 'Y']
[0 1]

[ ] print(heart['ST_Slope'].unique())
heart['ST_Slope'] = heart['ST_Slope'].replace(to_replace=['Up', 'Flat', 'Down'], value=[0, 1, 2]) # замена числовых значений на текстовые
print(heart['ST_Slope'].unique())

['Up' 'Flat' 'Down']
[0 1 2]

[ ] heart.head()

   Age Sex ChestPainType RestingBP Cholesterol FastingBS RestingECG MaxHR ExerciseAngina Oldpeak ST_Slope HeartDisease
0   40  0         0         0      140.0      289.0         0         0   172.0         0      0.0         0         0
1   49  1         1         1      160.0      180.0         0         0   156.0         0      1.0         1         1
2   37  0         0         0      130.0      283.0         0         1    98.0         0      0.0         0         0
3   48  1         2         1      138.0      214.0         0         0   108.0         1      1.5         1         1
4   54  0         1         0      150.0      195.0         0         0   122.0         0      0.0         0         0

```

Рисунок 29 - Замена строковых значений на числовые (2/2)

Датафрейм выше показывает, что теперь все данные датасета являются числовыми.

Был произведён выбор целевой переменной и стандартизация данных. Результат представлен на рисунке 30.

```
[ ] y = heart['HeartDisease']
x = heart.drop('HeartDisease', axis=1)

Так как данные в датасете имеют разный диапазон, их нужно стандартизировать.

[ ] scaler = StandardScaler() # создание
#StandardScaler используется для изменения размера распределения значений так, чтобы среднее значение наблюдаемых значений было равно 0, а стандартное отклонение – 1.
scaler.fit(x) # обучение
x = scaler.transform(x) # преобразование
```

Рисунок 30 - Выбор целевой переменной и стандартизация данных

Целевой переменной данного датафрейма является 'HeartDisease', которая показывает, есть ли у человека сердечное заболевание или нет.

Было произведено разделение данных на тренировочную и тестовую выборки. Результат представлен на рисунке 31.

```
▼ Разделение данных на тестовую и тренировочную выборки

Данные требуется случайно разделить на обучающую и тестовую выборки. 20% данных будут являться тестовыми, а остальные 80% - обучающими.

[ ] x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=33)
```

Рисунок 31 - Разделение данных на две выборки

Была создана модель. Она состоит из двух слоёв: входного и выходного. Результат представлен на рисунке 32.

```
▼ Создание модели

[ ] model = tf.keras.Sequential([
    tf.keras.layers.Dense(16, activation='relu', input_shape=(x_train.shape[1],)),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Рисунок 32 - Создание модели

В созданной модели присутствует 2 слоя: 1 входной и 1 выходной.

В данном примере был использован слой **Dense** – полносвязный слой (каждый нейрон данной слоя связан с каждым нейроном предыдущего/следующего слоя).

- Первый параметр слоя – **units**, данный параметр задаёт кол-во нейронов слое, в данном случае у первого слоя 16 нейронов, у второго 1.

- Второй параметр **activation** – параметр, задающий функцию активации нейронов, у первого функция активации relu, у второго – sigmoid.
- Крайний параметр **input\_shape** – параметр, отвечающий за размер входных данных, в нашем случае на вход каждого нейрона первого слоя подаётся массив размера (734, 11). Наличие данного параметра указывает на то, что слой является входным, следовательно из-за наличия всего двух слоёв, второй слой – выходной.

Был произведён вывод информации о модели. Результат представлен на рисунке 33.

```

[105] print(model.summary())

Model: "sequential_2"

```

Layer (type)	Output Shape	Param #
dense_6 (Dense)	(None, 16)	192
dense_7 (Dense)	(None, 1)	17

```

Total params: 209 (836.00 Byte)
Trainable params: 209 (836.00 Byte)
Non-trainable params: 0 (0.00 Byte)
None

```

### Рисунок 33 - Информация о модели

Model - название модели

- Layer - название и тип слоя
- Output Shape - размер данных на выход из слоя
- Param - количество параметров

Общее количество параметров = 209.

Количество тренируемых параметров = 209.



Была произведена визуализация архитектуры нейронной сети. Результат представлен на рисунке 34.

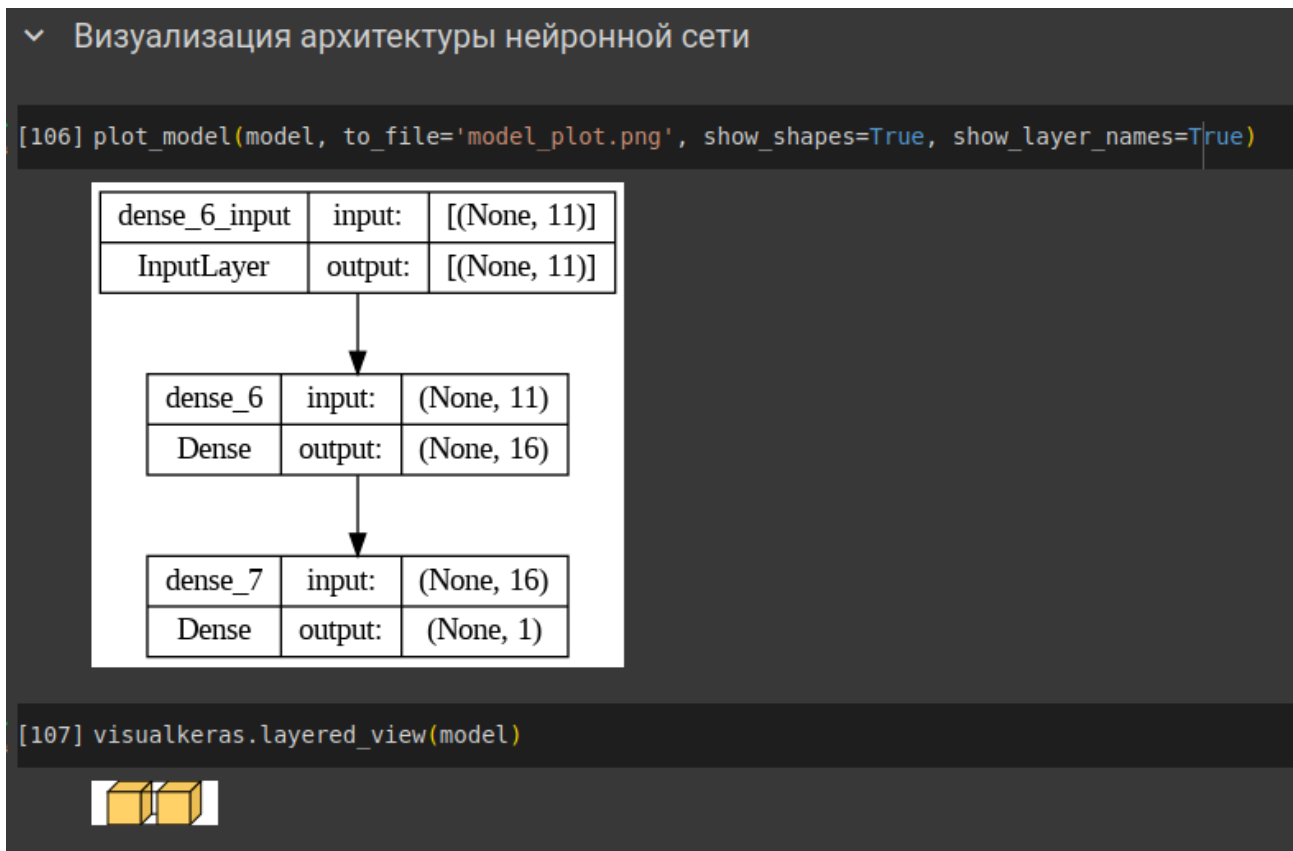


Рисунок 34 - Архитектура НС

Были определены параметры метода `compile()`. Результат представлен на рисунке 35.

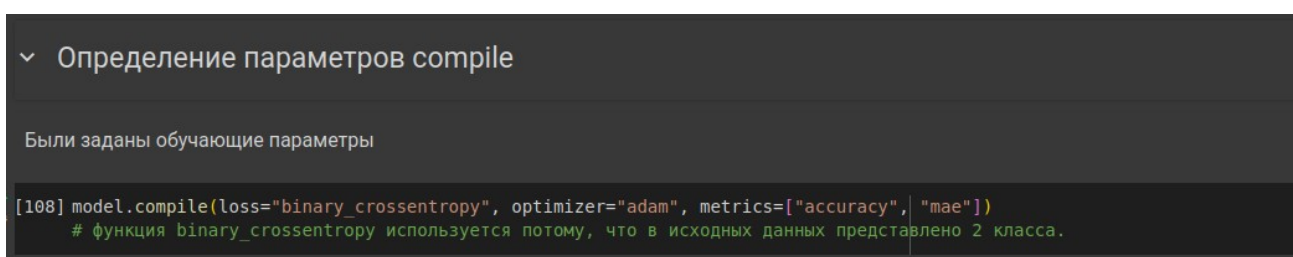


Рисунок 35 - Определение параметров `compile()`

- Параметр `loss` – ошибка между результатом, выдаваемым моделью и реальным значением, по сути - это цель, которую модель пытается свести к минимуму, например, `categorical_crossentropy` или `mse`.
- `optimizer` – алгоритм по которому будет проводится обновление синапсов нейронов,

- metrics – метрики, которые будут выводиться при обучении модели, нужны для оценки качества обучения

Было произведено обучение модели. Результат представлен на рисунке 36.

```

  ▾ Обучение модели, с помощью fit.

  Было произведено обучение модели на тестовой выборке.

  [109] history = model.fit(x_train, y_train, epochs=150, validation_data=(x_test, y_test))

  Epoch 1/150
  23/23 [=====] - 1s 8ms/step - loss: 0.8109 - accuracy: 0.4918 - mae: 0.5203 - val_loss: 0.7772 - val_accuracy: 0.4891 - val_mae: 0.5081
  Epoch 2/150
  23/23 [=====] - 0s 2ms/step - loss: 0.7009 - accuracy: 0.5858 - mae: 0.4772 - val_loss: 0.6874 - val_accuracy: 0.5761 - val_mae: 0.4699
  Epoch 3/150
  23/23 [=====] - 0s 3ms/step - loss: 0.6151 - accuracy: 0.6907 - mae: 0.4357 - val_loss: 0.6147 - val_accuracy: 0.6957 - val_mae: 0.4324
  Epoch 4/150
  23/23 [=====] - 0s 2ms/step - loss: 0.5475 - accuracy: 0.7452 - mae: 0.3973 - val_loss: 0.5597 - val_accuracy: 0.7337 - val_mae: 0.3983
  Epoch 5/150
  23/23 [=====] - 0s 3ms/step - loss: 0.4953 - accuracy: 0.7929 - mae: 0.3630 - val_loss: 0.5176 - val_accuracy: 0.7554 - val_mae: 0.3681
  Epoch 6/150
  23/23 [=====] - 0s 2ms/step - loss: 0.4561 - accuracy: 0.8134 - mae: 0.3338 - val_loss: 0.4870 - val_accuracy: 0.7880 - val_mae: 0.3426
  Epoch 7/150
  23/23 [=====] - 0s 3ms/step - loss: 0.4273 - accuracy: 0.8256 - mae: 0.3100 - val_loss: 0.4658 - val_accuracy: 0.8098 - val_mae: 0.3226
  Epoch 8/150
  23/23 [=====] - 0s 4ms/step - loss: 0.4068 - accuracy: 0.8324 - mae: 0.2908 - val_loss: 0.4500 - val_accuracy: 0.8098 - val_mae: 0.3059
  Epoch 9/150
  23/23 [=====] - 0s 2ms/step - loss: 0.3919 - accuracy: 0.8474 - mae: 0.2752 - val_loss: 0.4386 - val_accuracy: 0.8370 - val_mae: 0.2923
  Epoch 10/150
  23/23 [=====] - 0s 3ms/step - loss: 0.3807 - accuracy: 0.8474 - mae: 0.2639 - val_loss: 0.4315 - val_accuracy: 0.8370 - val_mae: 0.2831
  Epoch 11/150
  23/23 [=====] - 0s 2ms/step - loss: 0.3726 - accuracy: 0.8542 - mae: 0.2545 - val_loss: 0.4259 - val_accuracy: 0.8315 - val_mae: 0.2750
  Epoch 12/150
  23/23 [=====] - 0s 3ms/step - loss: 0.3668 - accuracy: 0.8556 - mae: 0.2465 - val_loss: 0.4214 - val_accuracy: 0.8315 - val_mae: 0.2675
  Epoch 13/150
  23/23 [=====] - 0s 2ms/step - loss: 0.3618 - accuracy: 0.8597 - mae: 0.2399 - val_loss: 0.4177 - val_accuracy: 0.8261 - val_mae: 0.2622
  Epoch 14/150
  23/23 [=====] - 0s 3ms/step - loss: 0.3581 - accuracy: 0.8610 - mae: 0.2350 - val_loss: 0.4149 - val_accuracy: 0.8261 - val_mae: 0.2575
  Epoch 15/150
  23/23 [=====] - 0s 2ms/step - loss: 0.3552 - accuracy: 0.8610 - mae: 0.2310 - val_loss: 0.4126 - val_accuracy: 0.8261 - val_mae: 0.2538
  Epoch 16/150
  23/23 [=====] - 0s 4ms/step - loss: 0.3521 - accuracy: 0.8651 - mae: 0.2272 - val_loss: 0.4096 - val_accuracy: 0.8261 - val_mae: 0.2507
  Epoch 17/150
  23/23 [=====] - 0s 3ms/step - loss: 0.3497 - accuracy: 0.8665 - mae: 0.2246 - val_loss: 0.4085 - val_accuracy: 0.8315 - val_mae: 0.2484
  Epoch 18/150
  23/23 [=====] - 0s 4ms/step - loss: 0.3475 - accuracy: 0.8665 - mae: 0.2217 - val_loss: 0.4068 - val_accuracy: 0.8315 - val_mae: 0.2460
  Epoch 19/150
  23/23 [=====] - 0s 3ms/step - loss: 0.3456 - accuracy: 0.8651 - mae: 0.2194 - val_loss: 0.4056 - val_accuracy: 0.8315 - val_mae: 0.2439
  Epoch 20/150
  23/23 [=====] - 0s 3ms/step - loss: 0.3439 - accuracy: 0.8638 - mae: 0.2178 - val_loss: 0.4044 - val_accuracy: 0.8315 - val_mae: 0.2432
  Epoch 21/150
  23/23 [=====] - 0s 3ms/step - loss: 0.3417 - accuracy: 0.8665 - mae: 0.2163 - val_loss: 0.4023 - val_accuracy: 0.8315 - val_mae: 0.2412
  Epoch 22/150
  23/23 [=====] - 0s 4ms/step - loss: 0.3400 - accuracy: 0.8665 - mae: 0.2150 - val_loss: 0.4012 - val_accuracy: 0.8315 - val_mae: 0.2407
  Epoch 23/150
  23/23 [=====] - 0s 3ms/step - loss: 0.3383 - accuracy: 0.8692 - mae: 0.2133 - val_loss: 0.3990 - val_accuracy: 0.8370 - val_mae: 0.2384
  Epoch 24/150
  23/23 [=====] - 0s 4ms/step - loss: 0.3369 - accuracy: 0.8706 - mae: 0.2120 - val_loss: 0.3981 - val_accuracy: 0.8370 - val_mae: 0.2381

```

Рисунок 36 - Обучение модели

- x\_train – Numpy массив, содержащий в себе обучающую выборку
- y\_train - Numpy массив, содержащий разметку данных для x\_train (верные ответы), (если несколько выходов - список массивов)
- epoch – кол-во эпох для обучения модели. В данном случае, количество эпох равно пяти.

Был произведён вывод точности модели и совершенно предсказание. Результат представлен на рисунке 37.

```

  ▾ Вывод точности модели

Были выведены метрики модели.

[110] model.evaluate(x_test, y_test)

6/6 [=====] - 0s 2ms/step - loss: 0.3799 - accuracy: 0.8207 - mae: 0.2115
[0.37991705536842346, 0.820652186870575, 0.21150848269462585]

```

Рисунок 37 - Точность модели

Loss рассчитывается по формуле mse. Среднеквадратичная ошибка (Mean Squared Error) – Среднее арифметическое (Mean) квадратов разностей между предсказанными и реальными значениями Модели (Model) Машинного обучения.

Метрика mae(Средняя абсолютная ошибка) рассчитывается как среднее абсолютных разностей между целевыми значением и значением, предсказанным моделью на данном обучающем примере в процессе обучения.

Accuracy(Точность) показывает насколько хорошо модель предсказывает значения. 83% является высоким показателем.

Были сравнены истинные и предсказанные значения. Результат представлен на рисунке 38.

## ✓ Сравнение истинных и предсказанных значений

Был создан датафрейм реальных и предсказанных данных

```
result_df = pd.DataFrame({'True_Score': y_test, 'Predicted_Score': y_pred.flatten()})  
print(result_df)
```

	True_Score	Predicted_Score
596	1	0.810090
412	1	0.997263
180	1	0.980178
439	1	0.973143
849	0	0.093989
..	...	...
378	1	0.972689
847	0	0.731551
257	0	0.001320
686	0	0.731351
652	0	0.356720

[184 rows x 2 columns]

Рисунок 38 - Сравнение истинных и предсказанных значений

По таблице видно, что модель достаточно точно угадывает значения.

Был построен график метрики качества по эпохам. Результат представлен на рисунке 39.

### График метрики качества по эпохам

```
[113] plt.figure(figsize=(12, 6))
```

```
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy по эпохам')
plt.xlabel('Эпохи')
plt.ylabel('Accuracy')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7cba6ed8da20>
```

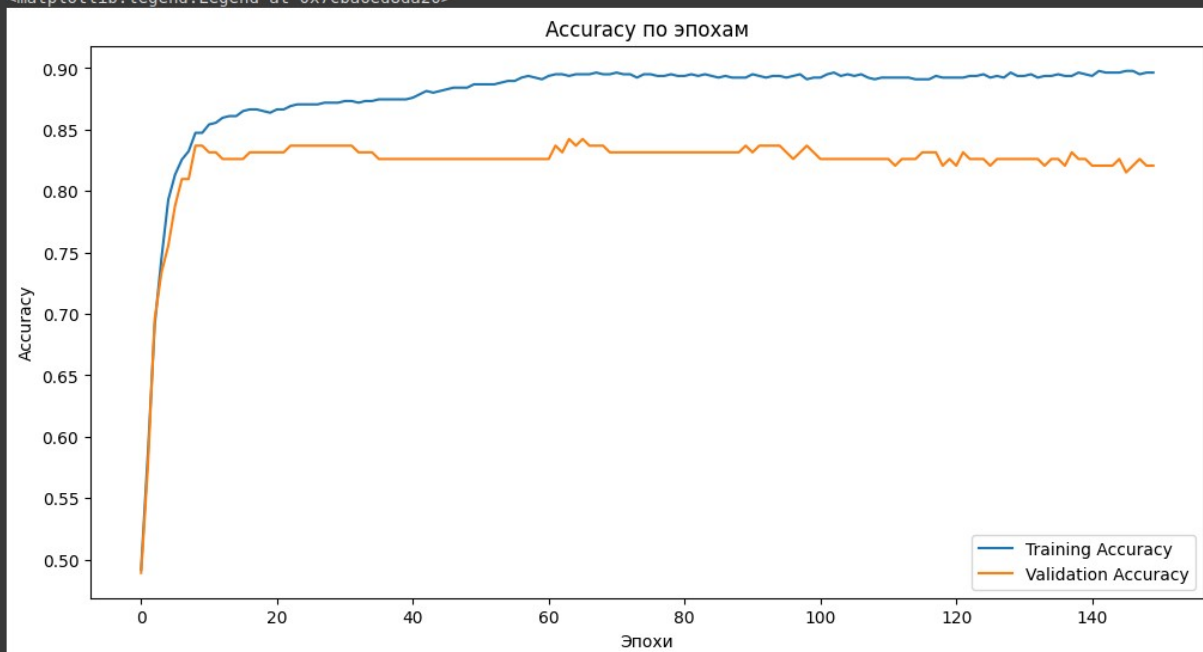


Рисунок 39 - График Accuracy/Epoch

На основе полученного графика можно сделать вывод, что для данной задачи при увеличении количества эпох точность предсказаний также резко увеличивается до 10 эпохи, после чего продолжается медленный рост тренировочной точности, но точность предсказания практически не изменяется.

Были построены графики метрик ошибок по эпохам. Результат представлен на рисунках 40 – 41.

## ✓ Графики метрик ошибок по эпохам

```
[114] plt.figure(figsize=(5,5))  
      plt.plot(history.history['mae'], label='mae')  
      plt.plot(history.history['val_mae'], label = 'val_mae')  
      plt.xlabel('Epoch')  
      plt.ylabel('mae')  
      plt.legend(loc='lower right')
```

<matplotlib.legend.Legend at 0x7cba70fdc7c0>

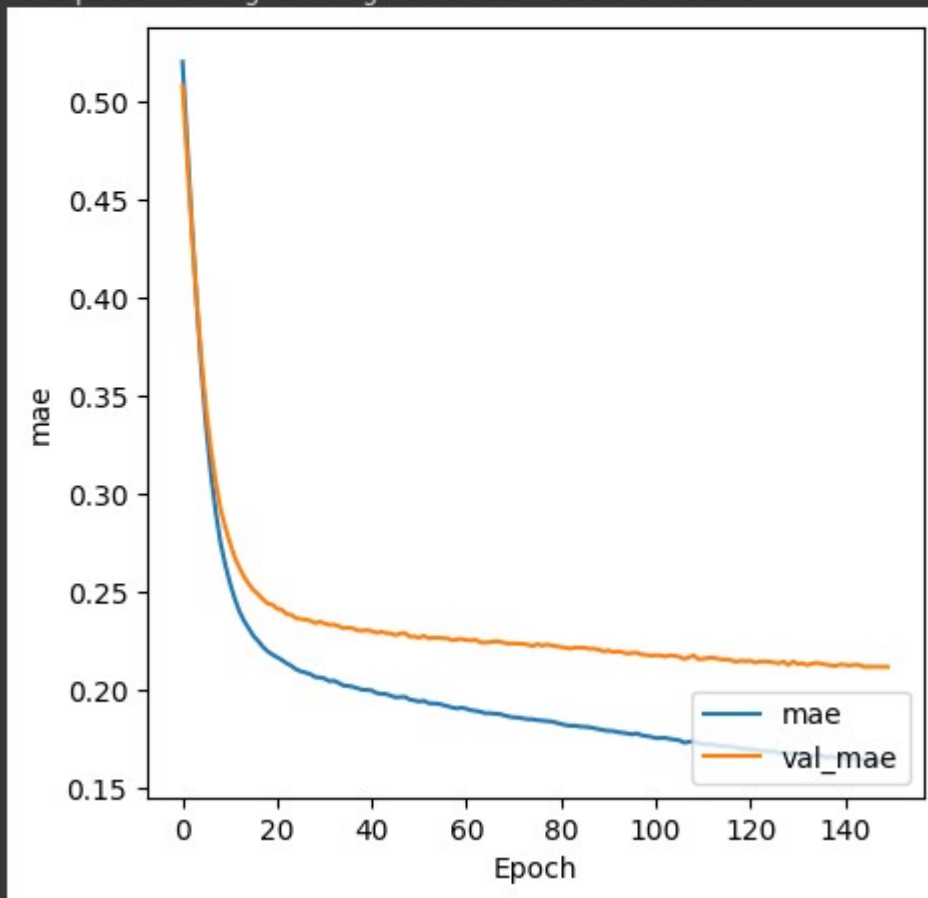


Рисунок 40 - График mae/Epoch

```
[115] plt.figure(figsize=(5,5))
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label = 'val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='lower right')
```

<matplotlib.legend.Legend at 0x7cba7109a9b0>

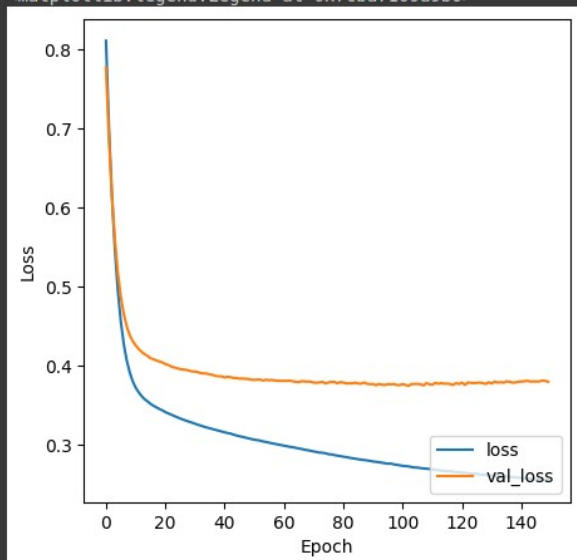


Рисунок 41 - График Loss/Epoch

На полученных графиках можно заметить, что с увеличением количества эпох - метрики Loss и MAE начинают убывать медленнее. Можно заметить по графику Loss/Epoch, что после 60 эпохи происходит переобучение модели. Следовательно оптимальное количество эпох равно 60.

Был построен график ROC-кривой. Результат представлен на рисунке 42.



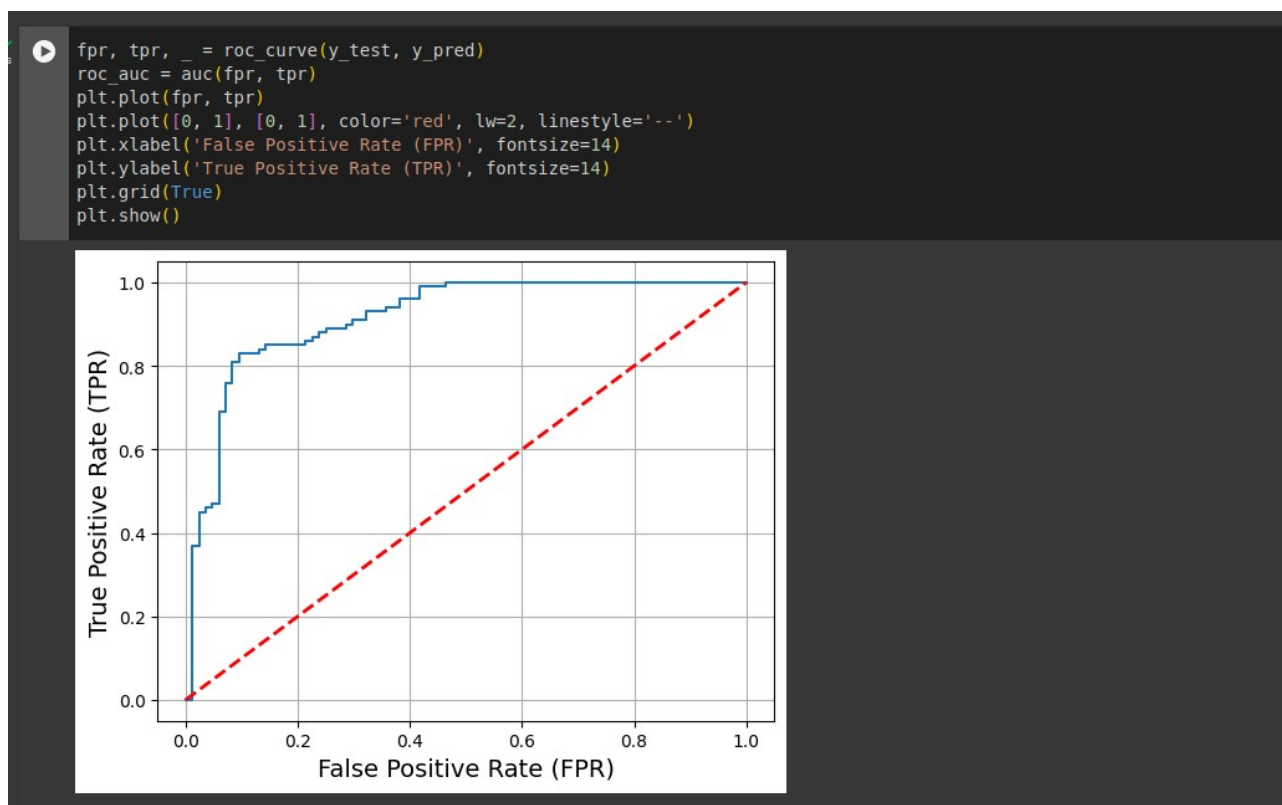


Рисунок 42 - График TPR/FPR

ROC-кривая описывает взаимосвязь между чувствительностью модели (TPR, или true positives rate — доля истинно положительных примеров) и её специфичностью (описываемой в отношении долей ложноположительных результатов:  $1 - \text{FPR}$ ).

TPR, или чувствительность модели, является соотношением корректных классификаций положительного класса, разделённых на все положительные классы, доступные из набора данных.

FPR — доля ложноположительных примеров, false positives rate. Это соотношение между ложными срабатываниями (количество прогнозов, ошибочно отнесённых в положительные), и всеми доступными отрицательными классами.

По графику видно, что доля правильно угаданных значений резко увеличивается, по сравнению с ложно угаданными значениями, что является доказательством корректной работы модели.

**Ссылка на блокнот**

## [Ссылка](#)

### **Вывод**

В ходе лабораторной работы были изучены простые полносвязные нейронные сети, а именно нейронные сети для выполнения задач классификации и регрессии.

Работа по полносвязным нейронным сетям включает в себя следующие этапы:

1. Подготовка данных: сначала необходимо подготовить данные для обучения сети, включая их предобработку, нормализацию и разделение на обучающую и тестовую выборки.
2. Определение архитектуры сети: выбор числа слоев, числа нейронов в каждом слое, функций активации и других параметров, которые определяют структуру нейронной сети.
3. Обучение сети: используя обучающий набор данных, сеть обучается путем минимизации функции потерь с помощью метода обратного распространения ошибки.
4. Оценка производительности: после обучения сети оценивается ее производительность на тестовом наборе данных, чтобы определить точность и другие метрики качества модели.
5. Корректировка модели. На основании качества предсказания, происходит настройка ответственных параметров: количества эпох, нейронов и слоёв.

Эти шаги составляют общий процесс работы по полносвязным нейронным сетям, который может быть адаптирован и расширен в зависимости от конкретной задачи и данных.

С помощью библиотеки TensorFlow была создана нейронная сеть, способная предсказывать оценку за экзамен по математике, основываясь на образовании родителей, расе, виду завтрака и прохождении подготовительного курса.

Также, аналогично была создана нейросеть-классификатор, способная определять болезнь сердца у людей.

На основе проведенных исследований, было установлено:

- Нейронные сети представляют собой удобный способ нахождения зависимостей между данными и выполнения предсказаний на основе уже известной информации.
- Нейронные сети могут быть применены в различных индустриях (медицина, наука, социология, экономика)