Вступ до ML на прикладі дослідження діабету в індіанському племені Пума

Постановка задачі:

1 частина

1) Провести датаналіз даних (пошук аномалій, видалення викидів, візуалізація і т.п.);
2) Розібрати код;
3) Здійснити тренування на дефолтному коді;
4) Спробувати відкинути деякі фічі і подивитися, чи буде змінюватися точність;
5) Змінити кількість шарів/активаційні функції і подивитися, як це буде впливати на вихідну точність;
6) Отримати максимально можливу точність;
7) Додати нормалізацію даних.

2 частина:

1) Натренувати на простій архітектурі класифікацію фізичної активності.

Критерії валідації:

1 частина:

1) Точність моделі нейронної мережі 95% і більше.

2 частина:

1) Точність моделі нейронної мережі 80% і більше;
2) Точність валідації моделі нейронної мережі 70% і більше.

Частина I

1) Виконання дефолтного коду з туторіалу

Спочатку було виконано код, наведений у туторіалі. В результаті було отримано модель нейронної мережі, точність якої після 150 епох становить 75.13%.

```
# evaluate the keras model
_, accuracy = model.evaluate(X, y)
print('Accuracy: %.2f' % (accuracy*100))
```

```
24/24 [==============================] - 0s 2ms/step - loss: 0.5291 - accuracy: 0.7513
Accuracy: 75.13
```

Рис. 1. Точність DNN-моделі

Для відображення зміни точності та втрат під час тренування моделі, було створено функцію plot_graph, яка відображає значення точності та втрат для кожної епохи.

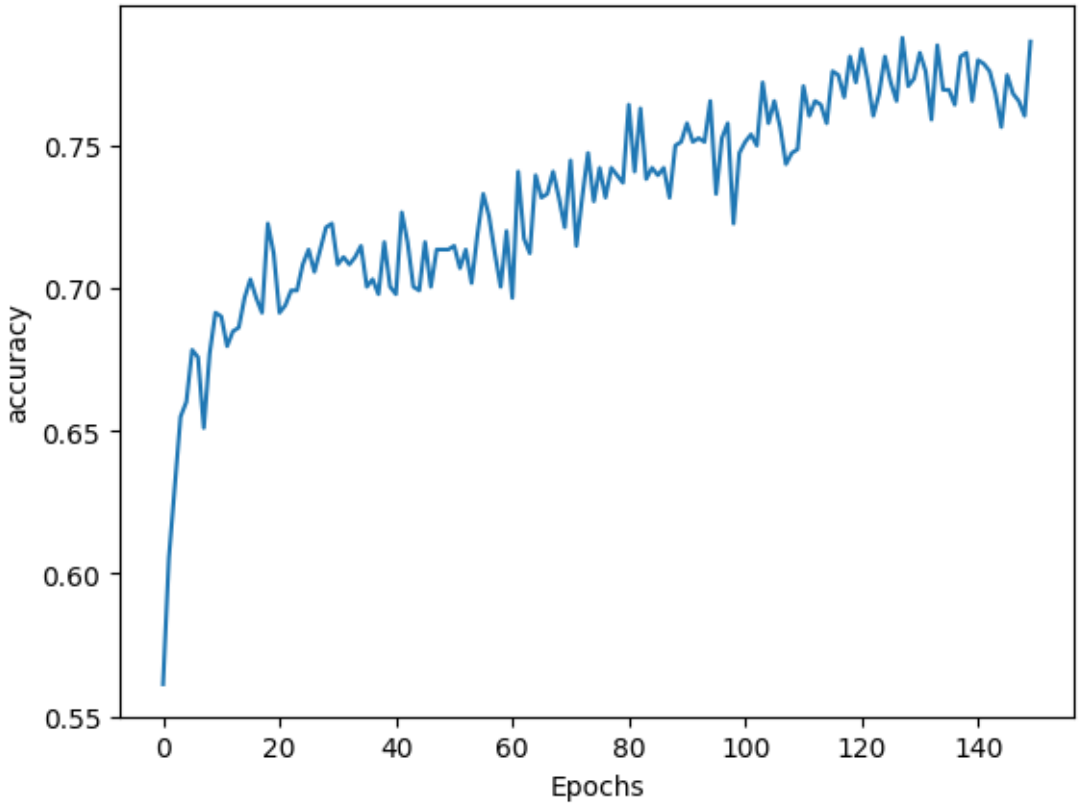

Рис. 2. Зміна точності протягом епох під час навчання моделі

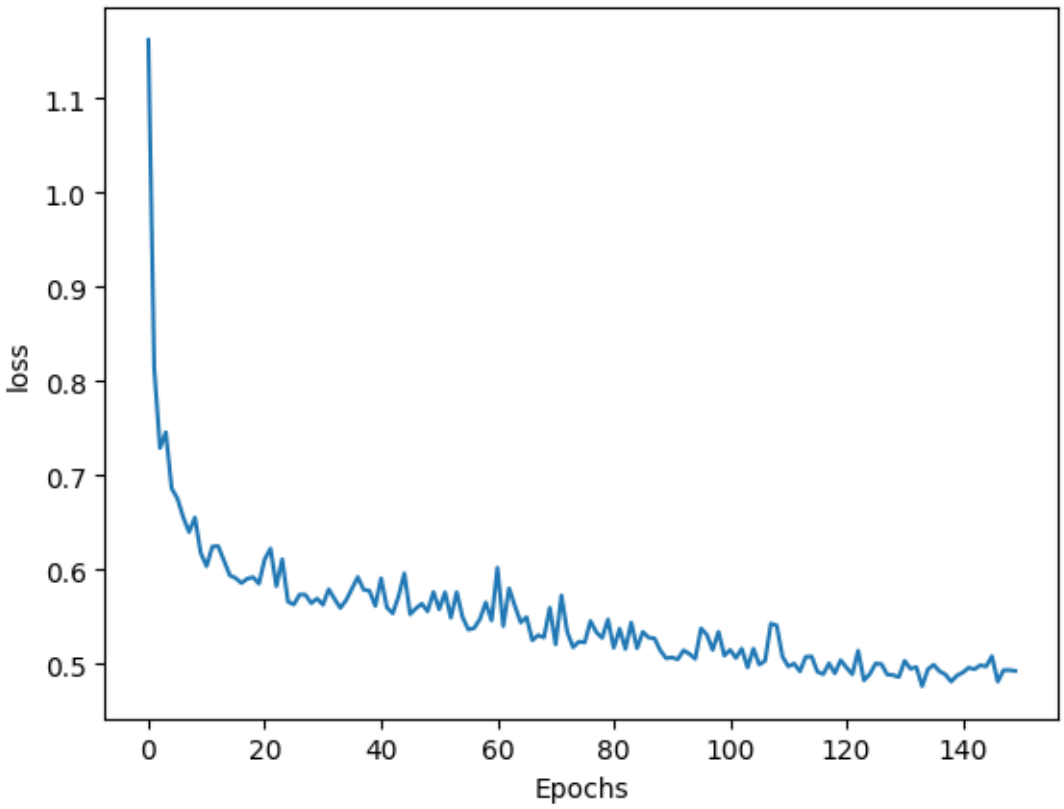

Рис. 3. Зміна втрат протягом епох під час навчання моделі

Повторюючи останні кроки туторіалу, розглянемо також перші п’ять передбачень/прогнозів моделі на тому ж тренувальному датасеті.

```
# make class predictions with the model
predictions = (model.predict(X) > 0.5).astype(int)

# summarise the first 5 cases
for i in range(5):
    print(f"{X[i].tolist()} => {predictions[i][0]} (true value = {y[i]})")
```

```
24/24 [==============================] - 0s 1ms/step
[6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0] => 1 (true value = 1.0)
[1.0, 85.0, 66.0, 29.0, 0.0, 26.6, 0.351, 31.0] => 0 (true value = 0.0)
[8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0] => 1 (true value = 1.0)
[1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.167, 21.0] => 0 (true value = 0.0)
[0.0, 137.0, 40.0, 35.0, 168.0, 43.1, 2.288, 33.0] => 1 (true value = 1.0)
```

Рис. 4. Прогнози натренованої моделі

Як бачимо із рис. 4, прогнози моделі для перших п’яти векторів ознак (feature vectors) виявилися правильними.

Оскільки нейронні мережі є стохастичними алгоритмами (той самий алгоритм на тих самих даних може навчати іншу модель з різними навичками кожного разу, коли виконується код), повторимо цей приклад ще 4 рази і знайдемо середнє значення точності моделі.

```
24/24 [==============================] - 0s 2ms/step - loss: 0.5291 - accuracy: 0.7513
Accuracy: 75.13

24/24 [==============================] - 0s 1ms/step - loss: 0.4642 - accuracy: 0.7865
Accuracy: 78.65

24/24 [==============================] - 0s 2ms/step - loss: 0.4810 - accuracy: 0.7747
Accuracy: 77.47

24/24 [==============================] - 0s 991us/step - loss: 0.5054 - accuracy: 0.7565
Accuracy: 75.65

24/24 [==============================] - 0s 2ms/step - loss: 0.4728 - accuracy: 0.7721
Accuracy: 77.21
```

Рис. 5. Результати тренування моделі (5 експериментів)

Таблиця 1.

Точність моделей нейронної мережі

№ експерименту	1	2	3	4	5
Точність моделі (%)	75.13	78.65	77.47	75.65	77.21

Із табл.1 видно, що всі показники точності становлять близько 77%, а середній показник становить 76.822%.

2) Аналіз датасету

Коротко про дослідження

Двогодинний пероральний тест на толерантність до глюкози (ПТТГ) – аналіз на цукор з навантаженням, який виконується в період 25–26 тижнів вагітності. Дослідження проводиться з метою виключення гестаційного діабету (діабету вагітних).

Спочатку беруть кров із вени і в лабораторії (не експрес-методом) визначають глікемію (вміст глюкози) у плазмі крові. Після цього вагітній дають випити 300 мл води, в якій розведено 75 г глюкози. Через 2 години роблять ще одне забирання крові. Протягом цього часу вагітна не повинна їсти, можна пити звичайну, не газовану воду.

Детальніше можна дізнатися на сайті: https://isida.ua/uk/diagnostika/disease/peroralnyij-test-na-tolerantnost-k-glyukoze-pttg/

"2-годинний сироватковий інсулін" (2-Hour serum insulin (mu U/ml) - стовпець №5) у наборі даних відноситься до вимірювання рівня інсуліну в сироватці крові, отриманого через 2 години після того, як пацієнт спожив навантаження глюкози в рамках перорального тесту на толерантність до глюкози (OGTT).

Вимірювання «2-годинного сироваткового інсуліну» в цьому наборі даних вказує на концентрацію інсуліну в сироватці крові через 2 години після вживання стандартизованої кількості глюкози під час перорального тесту на толерантність до глюкози.

Функція походження діабету (Diabetes pedigree function - DPF) розраховує ймовірність діабету залежно від віку суб'єкта та його/її сімейної історії діабету.

Етап 1: Data Cleaning and Preprocessing

Досліджуваний датасет було перетворено у датафрейм (pandas) із назвами стовпців, які відповідають вказаним в описі датафрейму.

```
columns = ['pregnancies', 'glucose_concentration', 'blood_pressure', 'triceps_skin_thickness', 'serum_insulin', 'bmi',
           'diabetes_pedigree_function', 'age', 'has_diabetes']
df = pd.read_csv('data/pima-indians-diabetes.csv', names=columns)
df.head()
```

	pregnancies	glucose_concentration	blood_pressure	triceps_skin_thickness	serum_insulin	bmi	diabetes_pedigree_function	age	has_diabetes
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Рис. 6. Вміст датафрейму df

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   pregnancies               768 non-null    int64
 1   glucose_concentration     768 non-null    int64
 2   blood_pressure            768 non-null    int64
 3   triceps_skin_thickness    768 non-null    int64
 4   serum_insulin             768 non-null    int64
 5   bmi                       768 non-null    float64
 6   diabetes_pedigree_function  768 non-null    float64
 7   age                       768 non-null    int64
 8   has_diabetes              768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

Рис. 7. Інформація про датафрейм df

Як бачимо із рис. 7, датафрейм не містить NaN значень.

Однак у пункті 8 опису датасету вказано, що наявні пропуски значень (відсутні значення) векторів ознак.

```
8. Missing Attribute Values: Yes
```

Рис. 8. В описі датасету вказано, що є відсутні значення

Тому було вирішено переглянути унікальні значення кожного із стовпців датафрейму із метою виявлення пропусків.

```
df['pregnancies'].unique()
```

```
array([ 6,  1,  8,  0,  5,  3, 10,  2,  4,  7,  9, 11, 13, 15, 17, 12, 14],
      dtype=int64)
```

Рис. 9. Унікальні значення стовпця 'pregnancies'

Розглянувши рис. 9, на перший погляд підозрілими є показники 0 та n > 10. Показник 0 можна пояснити тим, що на момент дослідження це була перша вагітність жінки (тобто до цього, вона не була вагітною). А от великі показники (n>10) є доволі підозрілими. Так, наприклад, значення 17 можна вважати викидом, оскільки воно значно вище за інші значення. Причиною цього можуть бути помилки введення даних, аномалії збору даних або фактичні рідкісні випадки. Для остаточного висновку потрібно провести додатковий аналіз. Однак, точно можна сказати про відсутність втрачених даних у цьому стовпці.

Далі розглянемо стовпець 'glucose_concentration'.

```
df['glucose_concentration'].unique()
```

```
array([148,  85, 183,  89, 137, 116,  78, 115, 197, 125, 110, 168, 139,
       189, 166, 100, 118, 107, 103, 126,  99, 196, 119, 143, 147,  97,
       145, 117, 109, 158,  88,  92, 122, 138, 102,  90, 111, 180, 133,
       106, 171, 159, 146,  71, 105, 101, 176, 150,  73, 187,  84,  44,
       141, 114,  95, 129,  79,   0,  62, 131, 112, 113,  74,  83, 136,
        80, 123,  81, 134, 142, 144,  93, 163, 151,  96, 155,  76, 160,
       124, 162, 132, 120, 173, 170, 128, 108, 154,  57, 156, 153, 188,
       152, 104,  87,  75, 179, 130, 194, 181, 135, 184, 140, 177, 164,
        91, 165,  86, 193, 191, 161, 167,  77, 182, 157, 178,  61,  98,
       127,  82,  72, 172,  94, 175, 195,  68, 186, 198, 121,  67, 174,
       199,  56, 169, 149,  65, 190], dtype=int64)
```

Рис. 10. Унікальні значення стовпця 'glucose_concentration'

Значення концентрації глюкози, що дорівнює 0, у стовпці 'glucose_concentration' (або будь-яке вимірювання рівня глюкози в крові) зазвичай вважається незвичним і потенційно вказує на проблему з даними. У контексті вимірювання рівня глюкози в крові значення 0 не є фізіологічно правдоподібним і, ймовірно, є заповнювачем або індикатором відсутніх даних. Перевіримо, скільки таких рядків присутні у датасеті.

```
len(df[df['glucose_concentration']==0])
```

```
5
```

Рис. 11. Кількість рядків із значенням 0 стовпця 'glucose_concentration'

Вважатимемо рядки із значенням 0 стовпця 'glucose_concentration' індикатором відсутніх даних і вони підлягатимуть видаленню.

Проте безпосередньо перед тим, як вносити зміни в оригінальний датасет, розглянемо спочатку усі його стовпці.

```
blood_pressure_arr = df['blood_pressure'].unique()
blood_pressure_arr.sort()
blood_pressure_arr
```

```
array([  0,  24,  30,  38,  40,  44,  46,  48,  50,  52,  54,  55,  56,
        58,  60,  61,  62,  64,  65,  66,  68,  70,  72,  74,  75,  76,
        78,  80,  82,  84,  85,  86,  88,  90,  92,  94,  95,  96,  98,
       100, 102, 104, 106, 108, 110, 114, 122], dtype=int64)
```

Рис. 12. Унікальні значення стовпця 'blood_pressure'

Як і для стовпця 'glucose_concentration', для стовпця 'blood_pressure' наявність значення 0 є аномальною. Дуже малоймовірно, що діастолічний артеріальний тиск людини дорівнюватиме рівно 0, оскільки це означатиме відсутність тиску в артеріях під час фази розслаблення серцебиття. Це значення може бути використане для представлення відсутніх або невідомих даних.

```
len(df[df['blood_pressure']==0])
```

```
35
```

Рис. 13. Кількість рядків із значенням 0 стовпця 'blood_pressure'

```
triceps_skin_thickness = df['triceps_skin_thickness'].unique()
triceps_skin_thickness.sort()
triceps_skin_thickness
```

```
array([ 0,  7,  8, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,
       24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
       41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 54, 56, 60, 63, 99],
      dtype=int64)
```

Рис. 14. Унікальні значення стовпця 'triceps_skin_thickness'

Із рис. 14 видно, що для стовпця 'triceps_skin_thickness', який відповідає за товщину шкіряної складки тріцепса у мм, присутнє значення 0. Але значення 0 для товщини шкірної складки не має фізіологічного сенсу та ймовірно використовується як заповнювач для відсутніх даних.

```
len(df[df['triceps_skin_thickness']==0])
```

```
227
```

Рис. 15. Кількість рядків із значенням 0 стовпця 'triceps_skin_thickness'

Як бачимо із рис. 15, загальна кількість рядків із значенням 0 стовпця 'triceps_skin_thickness', яке не має фізіологічного сенсу, становить 227, а це близько 30% даних датасету. Тому видалення цих рядків призведе до втрати значної кількості даних, що потенційно може призвести до менш репрезентативної моделі. Для вирішення цієї проблеми використано два підходи:

1) імпутація відсутніх значень за допомогою сучасних методів;
2) вилучення цілого стовпця.

Наступним розглянемо стовпець, відповідальний за вимірювання рівня інсуліну в сироватці крові – 'serum_insulin'.

```
serum_insulin = df['serum_insulin'].unique()
serum_insulin.sort()
serum_insulin
```

```
array([  0,  14,  15,  16,  18,  22,  23,  25,  29,  32,  36,  37,  38,
        40,  41,  42,  43,  44,  45,  46,  48,  49,  50,  51,  52,  53,
        54,  55,  56,  57,  58,  59,  60,  61,  63,  64,  65,  66,  67,
        68,  70,  71,  72,  73,  74,  75,  76,  77,  78,  79,  81,  82,
        83,  84,  85,  86,  87,  88,  89,  90,  91,  92,  94,  95,  96,
        99, 100, 105, 106, 108, 110, 112, 114, 115, 116, 119, 120, 122,
       125, 126, 127, 128, 129, 130, 132, 135, 140, 142, 144, 145, 146,
       148, 150, 152, 155, 156, 158, 159, 160, 165, 166, 167, 168, 170,
       171, 175, 176, 178, 180, 182, 183, 184, 185, 188, 190, 191, 192,
       193, 194, 196, 200, 204, 205, 207, 210, 215, 220, 225, 228, 230,
       231, 235, 237, 240, 245, 249, 250, 255, 258, 265, 270, 271, 272,
       274, 275, 277, 278, 280, 284, 285, 291, 293, 300, 304, 310, 318,
       321, 325, 326, 328, 330, 335, 342, 360, 370, 375, 387, 392, 402,
       415, 440, 465, 474, 478, 480, 485, 495, 510, 540, 543, 545, 579,
       600, 680, 744, 846], dtype=int64)
```

Рис. 16. Унікальні значення стовпця 'serum_insulin'

Як бачимо із рис. 16, і для цього стовпця наявне значення 0. Однак важливо зазначити, що на відміну від попередніх стовпців, наявність 0 не обов’язково вказує на відсутність значень у цьому випадку. Значення 0 справді може бути дійсним вимірюванням інсуліну в сироватці крові, особливо якщо аналіз, що використовується для вимірювання, має нижню межу виявлення, яка включає 0. Це поширене явище в багатьох лабораторних аналізах, і значення 0 може означати рівні інсуліну, які справді близькі або нижчі від нижньої межі виявлення.

```
len(df[df['serum_insulin']==0])
```

```
374
```

Рис. 17. Кількість рядків із значенням 0 стовпця 'serum_insulin'

Враховуючи кількість рядків із значенням 0 для стовпця 'serum_insulin', а також той факт, що значення 0 може означати близькі до нижньої межі рівні інсуліну, вважатимемо такі рядки дійсними.

```
bmi = df['bmi'].unique()
bmi.sort()
bmi
```

```
array([ 0. , 18.2, 18.4, 19.1, 19.3, 19.4, 19.5, 19.6, 19.9, 20. , 20.1,
       20.4, 20.8, 21. , 21.1, 21.2, 21.7, 21.8, 21.9, 22.1, 22.2, 22.3,
       22.4, 22.5, 22.6, 22.7, 22.9, 23. , 23.1, 23.2, 23.3, 23.4, 23.5,
       23.6, 23.7, 23.8, 23.9, 24. , 24.1, 24.2, 24.3, 24.4, 24.5, 24.6,
       24.7, 24.8, 24.9, 25. , 25.1, 25.2, 25.3, 25.4, 25.5, 25.6, 25.8,
       25.9, 26. , 26.1, 26.2, 26.3, 26.4, 26.5, 26.6, 26.7, 26.8, 26.9,
       27. , 27.1, 27.2, 27.3, 27.4, 27.5, 27.6, 27.7, 27.8, 27.9, 28. ,
       28.1, 28.2, 28.3, 28.4, 28.5, 28.6, 28.7, 28.8, 28.9, 29. , 29.2,
       29.3, 29.5, 29.6, 29.7, 29.8, 29.9, 30. , 30.1, 30.2, 30.3, 30.4,
       30.5, 30.7, 30.8, 30.9, 31. , 31.1, 31.2, 31.3, 31.6, 31.9, 32. ,
       32.1, 32.2, 32.3, 32.4, 32.5, 32.6, 32.7, 32.8, 32.9, 33.1, 33.2,
       33.3, 33.5, 33.6, 33.7, 33.8, 33.9, 34. , 34.1, 34.2, 34.3, 34.4,
       34.5, 34.6, 34.7, 34.8, 34.9, 35. , 35.1, 35.2, 35.3, 35.4, 35.5,
       35.6, 35.7, 35.8, 35.9, 36. , 36.1, 36.2, 36.3, 36.4, 36.5, 36.6,
       36.7, 36.8, 36.9, 37. , 37.1, 37.2, 37.3, 37.4, 37.5, 37.6, 37.7,
       37.8, 37.9, 38. , 38.1, 38.2, 38.3, 38.4, 38.5, 38.6, 38.7, 38.8,
       38.9, 39. , 39.1, 39.2, 39.3, 39.4, 39.5, 39.6, 39.7, 39.8, 39.9,
       40. , 40.1, 40.2, 40.5, 40.6, 40.7, 40.8, 40.9, 41. , 41.2, 41.3,
       41.5, 41.8, 42. , 42.1, 42.2, 42.3, 42.4, 42.6, 42.7, 42.8, 42.9,
       43.1, 43.2, 43.3, 43.4, 43.5, 43.6, 44. , 44.1, 44.2, 44.5, 44.6,
       45. , 45.2, 45.3, 45.4, 45.5, 45.6, 45.7, 45.8, 46.1, 46.2, 46.3,
       46.5, 46.7, 46.8, 47.9, 48.3, 48.8, 49.3, 49.6, 49.7, 50. , 52.3,
       52.9, 53.2, 55. , 57.3, 59.4, 67.1])
```

Рис. 18. Унікальні значення стовпця 'bmi'

Розглянувши рис. 18, можна знову побачити значення 0. Важливо розуміти, що ІМТ, що дорівнює 0, або має надзвичайно низькі значення, як-от 0, можуть свідчити про відсутність або недійсність даних, оскільки ІМТ є розрахованим значенням, яке вимагає ненульових вимірювань ваги та зросту. Тому такі рядки вважатимемо рядками із втраченими даними (missing values).

```
len(df[df['bmi']==0])
```

```
11
```

Рис. 19. Кількість рядків із значенням 0 стовпця 'bmi'

Кількість рядків із нульовим значенням ІМТ становить близько 1.5% від загального обсягу датасету, тому їх можна буде видалити.

```
diabetes_pedigree_func_arr = df['diabetes_pedigree_function'].unique()
diabetes_pedigree_func_arr.sort()
diabetes_pedigree_func_arr
```

```
array([0.078, 0.084, 0.085, 0.088, 0.089, 0.092, 0.096, 0.1  , 0.101,
       0.102, 0.107, 0.108, 0.115, 0.118, 0.121, 0.122, 0.123, 0.126,
       0.127, 0.128, 0.129, 0.13 , 0.133, 0.134, 0.135, 0.136, 0.137,
       0.138, 0.14 , 0.141, 0.142, 0.143, 0.144, 0.145, 0.147, 0.148,
       0.149, 0.15 , 0.151, 0.153, 0.154, 0.155, 0.156, 0.157, 0.158,

       1.114, 1.127, 1.136, 1.138, 1.144, 1.154, 1.159, 1.162, 1.174,
       1.182, 1.189, 1.191, 1.213, 1.222, 1.224, 1.251, 1.258, 1.268,
       1.282, 1.292, 1.318, 1.321, 1.353, 1.39 , 1.391, 1.394, 1.4  ,
       1.441, 1.461, 1.476, 1.6  , 1.698, 1.699, 1.731, 1.781, 1.893,
       2.137, 2.288, 2.329, 2.42 ])
```

Рис. 20. Унікальні значення стовпця 'diabetes_pedigree_function' (перші та останні 5 рядків)

Дивлячись на список унікальних значень для стовпця 'diabetes_pedigree_function', здається, що немає значення 0 або інших значень, які могли б безпосередньо представляти відсутні значення.

Залишилося дослідити вміст останнього стовпця вектора ознак – стовпця 'age'.

```
age_arr = df['age'].unique()
age_arr.sort()
age_arr
```

```
array([21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37,
       38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54,
       55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 72,
       81], dtype=int64)
```

Рис. 21. Унікальні значення стовпця 'age'

Із рис. 21 видно, що найменший вік жінки становить 21 рік, а найбільший – 81. Найменший вік співпадає із вказаним в описі до датасету (рис. 22).

```
4. Relevant Information:
   Several constraints were placed on the selection of these instances from
   a larger database.  In particular, all patients here are females at
   least 21 years old of Pima Indian heritage.  ADAP is an adaptive learning
   routine that generates and executes digital analogs of perceptron-like
   devices.  It is a unique algorithm; see the paper for details.
```

Рис. 22. Мінімальний вік, вказаний в описі датасету

Розглянемо також унікальні значення для стовпця 'has_diabetes', який виступатиме в ролі лейбелів (labels) для моделі нейронної мережі.

```
df['has_diabetes'].unique()
```

```
array([1, 0], dtype=int64)
```

Рис. 23. Унікальні значення стовпця 'has_diabetes'

Вміст стовпця 'has_diabetes' співпадає із очікуваним.

Отже, дослідивши усі 9 стовпців датафрейму, отримано такі результати:

1) Стовпці 'pregnancies', 'serum_insulin', 'diabetes_pedigree_function' та 'has_diabetes' не мають відсутніх значень (missing values).
2) Стовпці 'glucose_concentration', 'blood_pressure' і 'bmi' мають декілька відсутніх значень (5, 35 і 11 відповідно), замінених значенням 0. Тому рядки із такими значеннями можна спробувати видалити.
3) Стовпець 'triceps_skin_thickness' містить 227 відсутніх/неправильних значень (= 0). Видалення рядків із цим значенням призведе до втрати значної кількості даних, тому для вирішення проблеми планується використати підхід імпутації даних або вилучення цілого стовпця під час тренування моделі нейронної мережі (тобто цей стовпець просто не входитиме до вектора ознак).

Перед видаленням рядків, в яких стовпці 'glucose_concentration', 'blood_pressure' і 'bmi' мають значення 0, розглянемо скільки рядків буде втрачено.

```
len(df[(df['glucose_concentration'] == 0) | (df['blood_pressure'] == 0) | (df['bmi'] == 0)])
```

```
44
```

Рис. 24. Кількість рядків, які буде видалено

Із рис. 24 видно, що внаслідок Data Cleaning буде видалено 44 рядки, що становить 7.3% від загальної кількості рядків у датасеті.

Проведемо процес Data Cleaning для вказаних стовпців.

```
df = df[(df['glucose_concentration'] != 0) & (df['blood_pressure'] != 0) & (df['bmi'] != 0)]
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 724 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   pregnancies               724 non-null    int64  
 1   glucose_concentration     724 non-null    int64  
 2   blood_pressure            724 non-null    int64  
 3   triceps_skin_thickness    724 non-null    int64  
 4   serum_insulin             724 non-null    int64  
 5   bmi                       724 non-null    float64
 6   diabetes_pedigree_function  724 non-null    float64
 7   age                       724 non-null    int64  
 8   has_diabetes              724 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 56.6 KB
```

Рис. 25. Інформація про датафрейм df після проведення Data Cleaning

Як було вказано вище, для стовпця 'triceps_skin_thickness' використаємо підхід імпутації даних, зокрема, K-Nearest Neighbors (KNN) Imputation. Вибір саме цього методу імпутації даних можна пояснити тим, що цей метод оцінює відсутні значення, використовуючи значення k-найближчих сусідів у наборі даних. Він враховує подібність між зразками та може бути хорошим вибором, якщо існує кореляція між стовпцем 'triceps_skin_thickness' та іншими характеристиками.

Для того, щоб виявити наявність кореляції між 'triceps_skin_thickness' та іншими характеристиками, побудуємо матрицю кореляції для датафрейму df.

Рис. 26. Матриця кореляції датафрейму df

Розглянувши матрицю кореляції датафрейму df (рис. 26), можна помітити наявність кореляції між стовпцем 'triceps_skin_thickness' та стовпцем 'serum_insulin' (0.42), а також між 'triceps_skin_thickness' та 'bmi' (0.40). Хоча згадані кореляції не надто високі, вони все ж припускають певний рівень зв’язку між змінними, тому використання KNN імпутації є доцільним.

Перед виконанням KNN імпутації, розглянемо кількість рядків датафрейму df, які мають значення 0 стовпця 'triceps_skin_thickness', які залишилися після виконання Data Cleaning для стовпців 'glucose_concentration', 'blood_pressure' і 'bmi'.

```
len(df[df['triceps_skin_thickness'] == 0])
```

```
192
```

Рис. 27. Кількість рядків із значенням 0 стовпця 'triceps_skin_thickness'

```python
df_knn = df.copy()
# Subset of relevant columns for KNN imputation
subset_columns = ['triceps_skin_thickness', 'serum_insulin', 'bmi']

# Separate data with missing 'triceps_skin_thickness'
missing_data = df_knn[df_knn['triceps_skin_thickness'] == 0][subset_columns]
known_data = df_knn[df_knn['triceps_skin_thickness'] != 0][subset_columns]

# Reset the index of both DataFrames
known_data.reset_index(drop=True, inplace=True)
missing_indexes = missing_data.index
missing_indexes = list(missing_indexes)
# Fit a NearestNeighbors model
neighbors_number = 5
neighbors_model = NearestNeighbors(n_neighbors=neighbors_number)
neighbors_model.fit(known_data.drop('triceps_skin_thickness', axis=1).values)
changed_indexes = []
# Impute missing values using KNN
for index, row in missing_data.iterrows():
    query_point = row.drop('triceps_skin_thickness')
    distances, indices = neighbors_model.kneighbors([query_point])
    avg_triceps_thickness = np.mean([known_data.loc[idx, 'triceps_skin_thickness'] for idx in indices[0]])
    changed_indexes.append(index)
    df_knn.loc[df_knn.index == index, 'triceps_skin_thickness'] = int(avg_triceps_thickness)

print(f"len(df_knn[df_knn['triceps_skin_thickness'] == 0]) = {len(df_knn[df_knn['triceps_skin_thickness'] == 0])}")
```

```
len(df_knn[df_knn['triceps_skin_thickness'] == 0]) = 0
```

Рис. 28. Результати виконання KNN імпутації

Як видно із рис. 28, внаслідок виконання KNN імпутації кількість рядків зі значенням 0 стовпця 'triceps_skin_thickness' зменшилася від 192 у df датафреймі до 0 у df_knn датафреймі.

Розглянемо, як змінився розподіл міток досліджуваного стовпця після виконання KNN імпутації (рис. 29 і рис. 30).

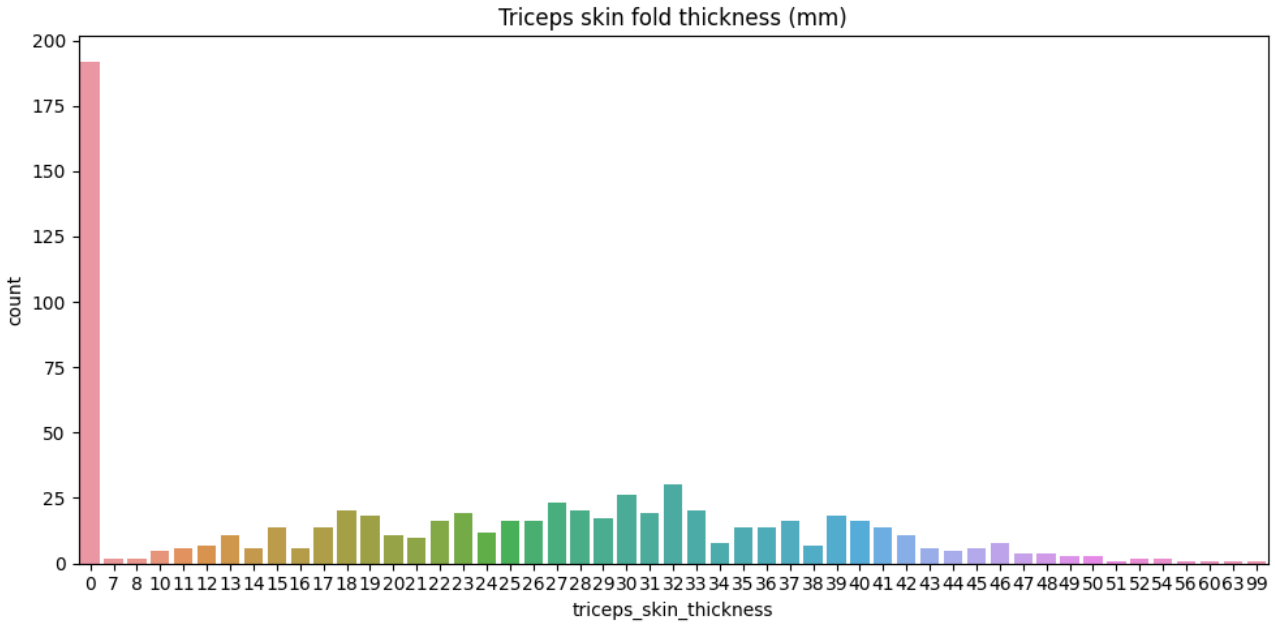

Рис. 29. Розподіл міток стовпця 'triceps_skin_thickness' перед виконанням KNN імпутації

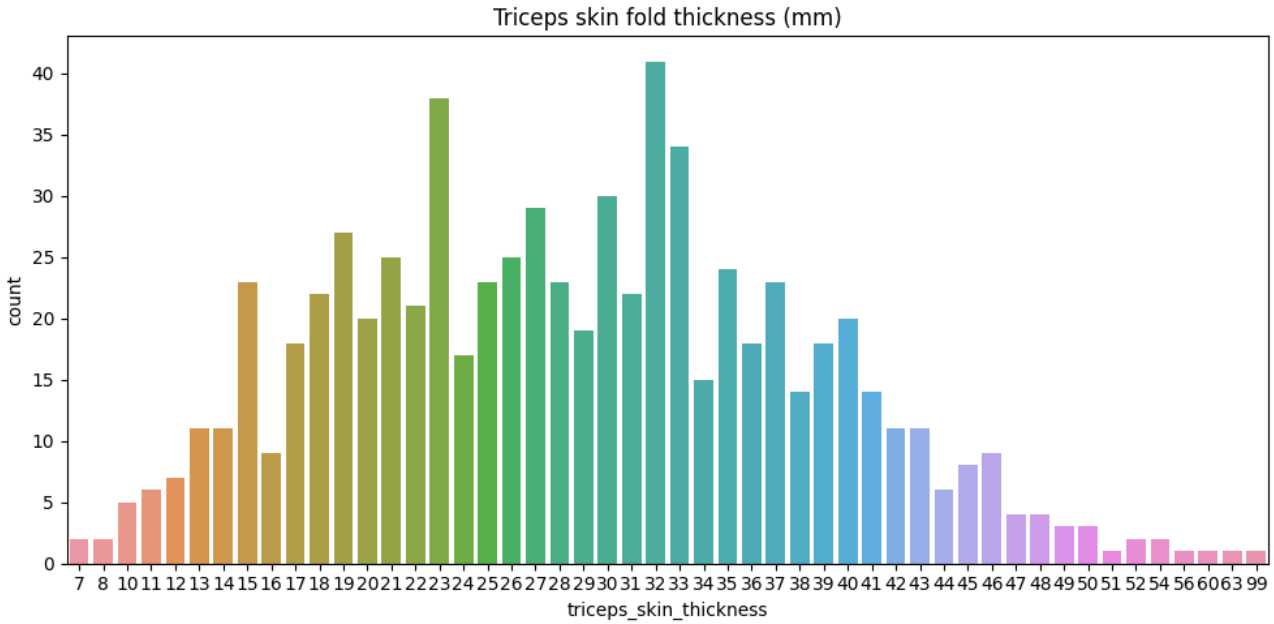

Рис. 30. Розподіл міток стовпця 'triceps_skin_thickness' після виконання KNN імпутації

Із рис. 30 видно, що внаслідок KNN імпутації зникла мітка 0. Також варто зазначити, що внаслідок KNN імпутації загальна картина розподілу міток для стовпця 'triceps_skin_thickness' незначно змінилася, тобто загальна поведінка збереглася (немає строго однієї мітки, яка виділяється серед інших, як би це було, наприклад, внаслідок виконання mean імпутації).

Для наглядніщого відображення зміни розподілу міток стовпця 'triceps_skin_thickness' побудуємо дві відсоткові кругові діаграми (рис. 31).

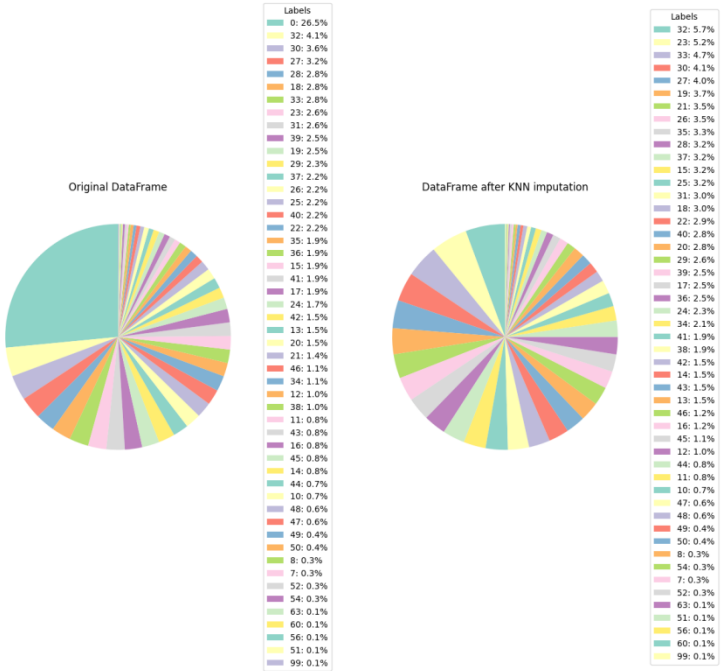

Рис. 31. Розподіл міток стовпця 'triceps_skin_thickness'

Етап 2: Exploratory Data Analysis

Для початку розглянемо розподіл міток для стовпця 'pregnancies'.

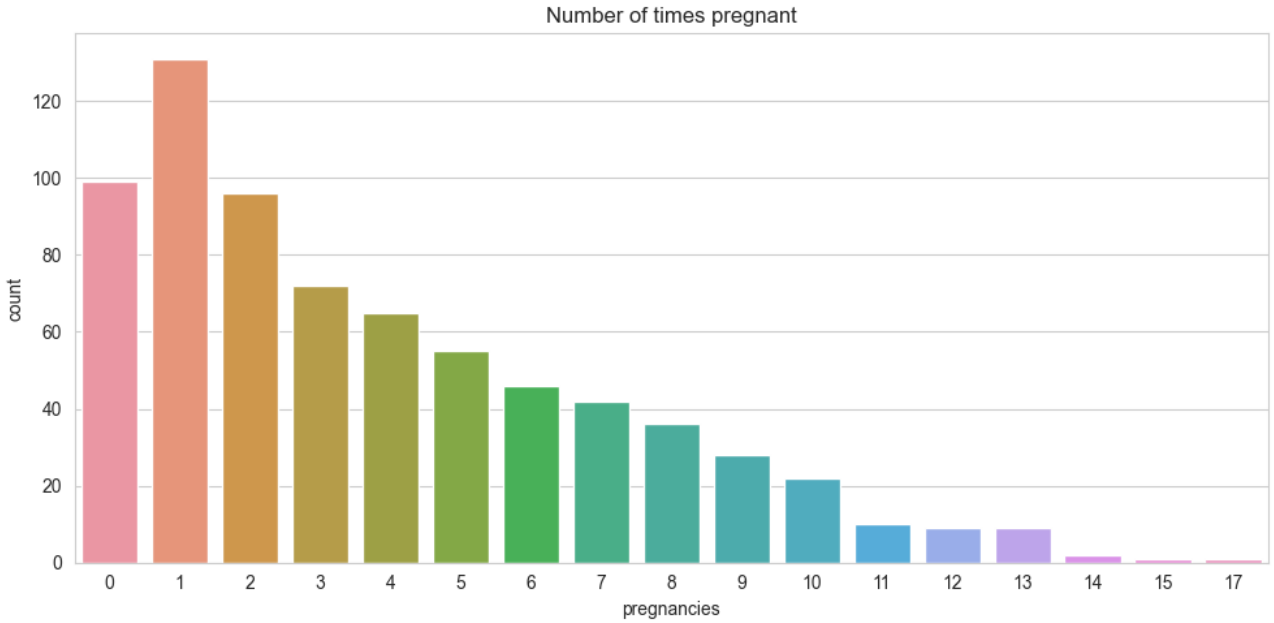

Рис. 32. Розподіл міток стовпця 'pregnancies'

У досліджуваному датафреймі переважають записи про жінок, які були вагітними менше шести разів. Незначну частину становлять мітки із значенням, більшим 10.

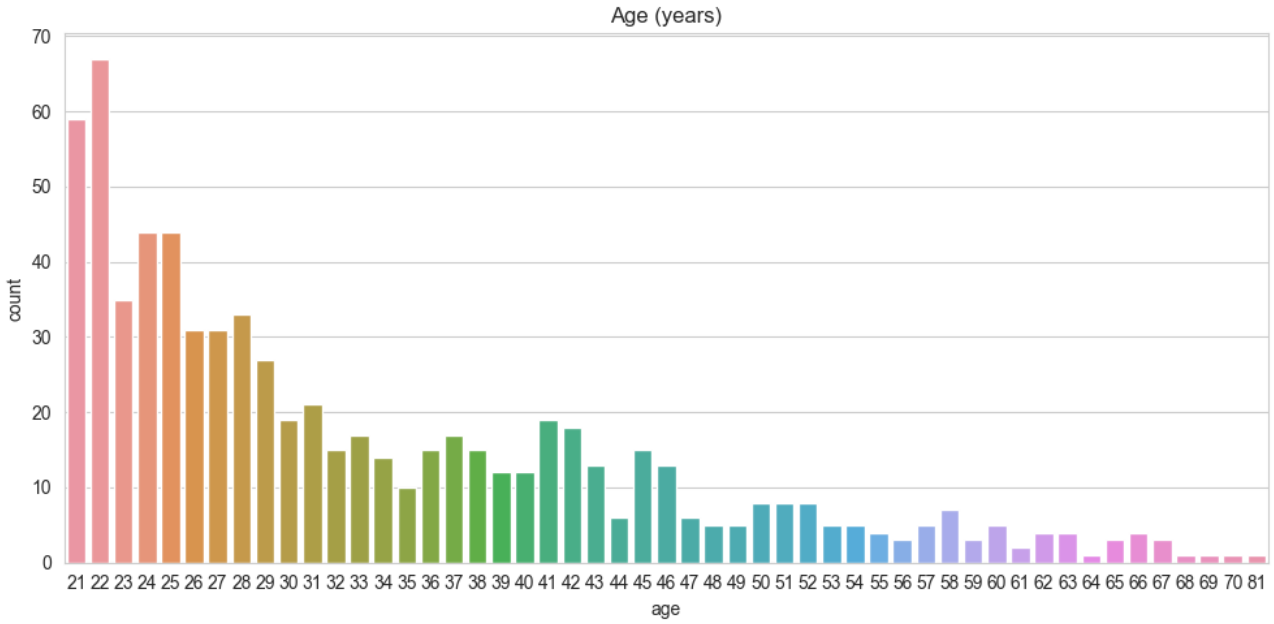

Рис. 33. Розподіл міток стовпця 'age'

Проаналізувавши рис. 32 і рис. 33, можна зробити висновок, що в основному датасет містить багато інформації про молодих жінок (21-30 років) та про жінок, які були вагітними не більше 6 разів.

Схожа форма графіків для віку та кількості вагітностей наштовхує на думку про існування певної кореляції між цими двома ознаками, що підтверджується матрицею кореляції (рис. 26): кореляція становить 0.56.

Важливо також розглянути розподіл міток стовпця 'has_diabetes', який виступає в якості вихідних значень моделі машинного навчання.

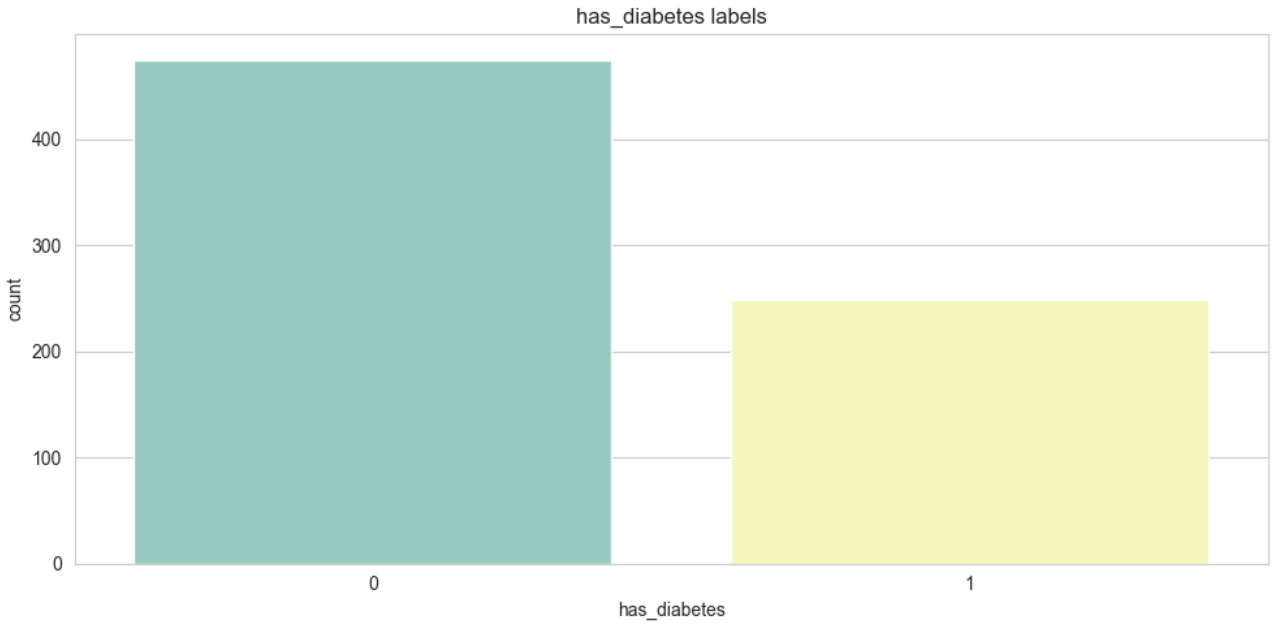

Рис. 34. Розподіл міток стовпця 'has_diabetes'

Із рис. 34 видно, що кількість записів із міткою 0 приблизно вдвічі перевищує кількість записів із міткою 1. Отже, досліджуваний датасет є незбалансованим. Дисбаланс у розподілі класів потенційно може призвести до проблем під час навчання моделі класифікації. Модель може стати упередженою до класу більшості та погано працювати з класом меншості.

Для чистоти експерименту спробуємо спочатку натренувати модель на незбалансованому датасеті.

Етап 3: Train a neural network model

Спочатку для тренування моделі використаємо код із туторіалу (для тренування моделі використано 150 епох), щоб побачити вплив очистки даних та KNN імпутації стовпця 'triceps_skin_thickness'.

Під час першого тренування отримано точність 76% (рис. 35).

```
23/23 [==============================] - 0s 1ms/step - loss: 0.4827 - accuracy: 0.7597
Accuracy: 75.97
```

Рис. 35. Точність моделі нейронної мережі

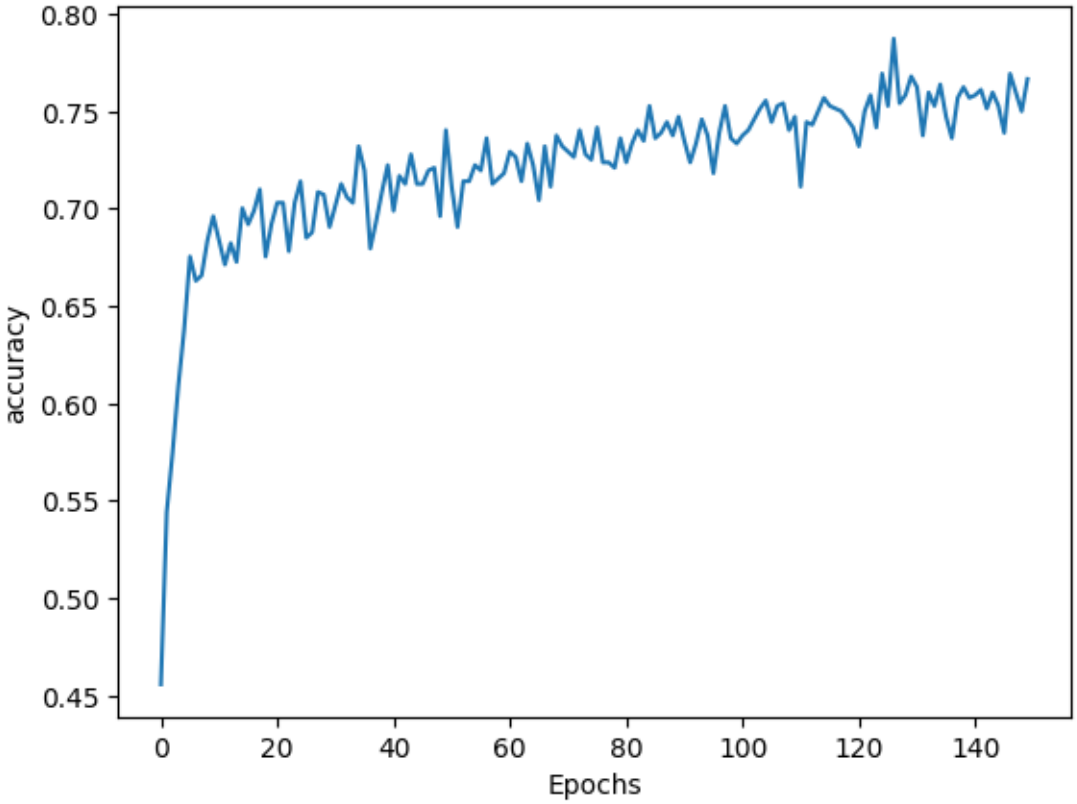

Рис. 36. Зміна точності протягом епох під час навчання моделі

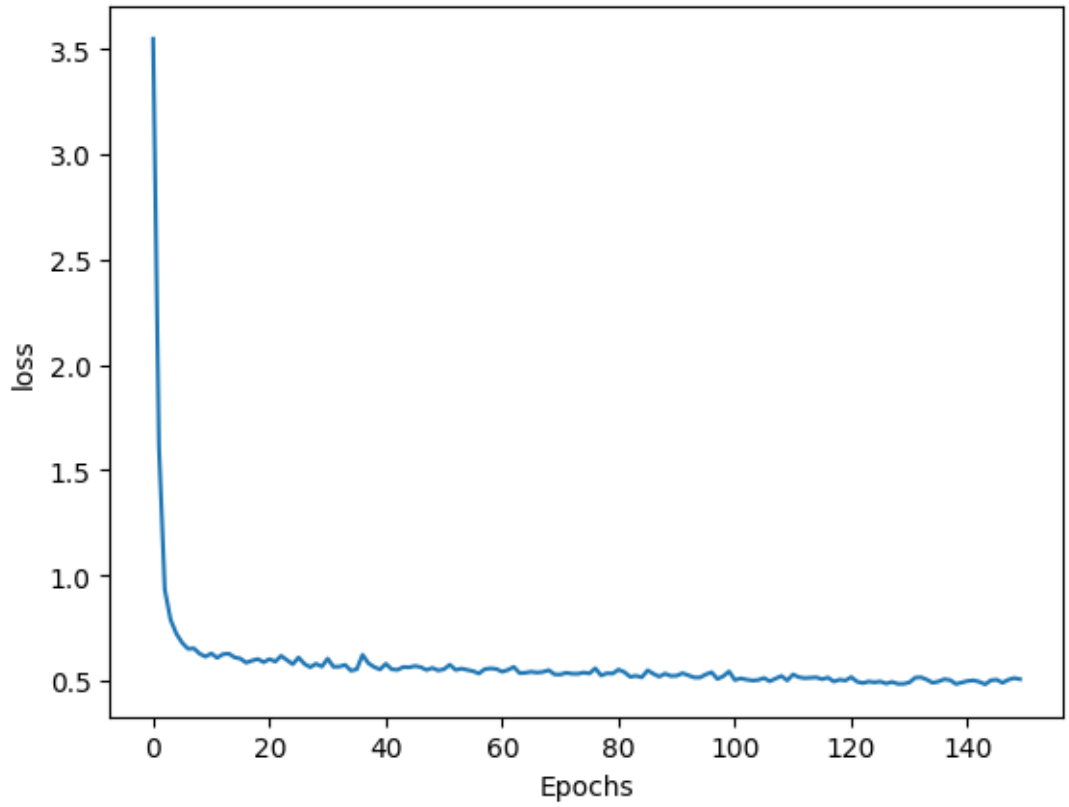

Рис. 37. Зміна втрат протягом епох під час навчання моделі

Порівнюючи зміну втрат протягом епох під час тренування моделі за оригінальним датафреймом df (рис. 3) та датафреймом df_knn з очищеними стовпцями 'glucose_concentration', 'blood_pressure' і 'bmi', та KNN імпутованим

стовпцем 'triceps_skin_thickness' (рис. 37), можна помітити відсутність різких загострень починаючи із 20-ої епохи (тобто більша плавність). Однак така особливість може бути наслідком самого навчання моделі (стохастична природа нейронних мереж), а не зміною вмісту датафрейму.

Як і під час виконання дефолтного коду із туторіалу на оригінальному датасеті, повторимо дослід 5 разів.

```
23/23 [==============================] - 0s 1ms/step - loss: 0.4827 - accuracy: 0.7597
Accuracy: 75.97
23/23 [==============================] - 0s 1ms/step - loss: 0.4729 - accuracy: 0.7859
Accuracy: 78.59
23/23 [==============================] - 0s 1ms/step - loss: 0.4819 - accuracy: 0.7693
Accuracy: 76.93
23/23 [==============================] - 0s 1ms/step - loss: 0.4882 - accuracy: 0.7638
Accuracy: 76.38
23/23 [==============================] - 0s 710us/step - loss: 0.4946 - accuracy: 0.7707
Accuracy: 77.07
```

Рис. 38. Результати тренування моделі (5 експериментів)

Таблиця 2.

Точність моделей нейронної мережі

№ експерименту	1	2	3	4	5
Точність моделі (%)	75.97	78.59	76.93	76.38	77.07

Із табл.2 видно, що всі показники точності становлять близько 77%, а середній показник становить 76.988%.

Порівнюючи отримані результати із результатами із табл.1, можна зробити висновок, що проведений Data Cleaning та Data Imputing не змогли покращити точність моделі.

Етап 4: Improve the neural network model

4.1) Discard feature

Слідуючи плану ДЗ, наступним кроком слід спробувати відкинути деякі фічі і подивитися, чи буде змінюватися точність моделі.

Однак перед цим, дослідимо вплив різних стовпців (ознак, фіч) на тренування моделі шляхом аналізу ваг нейронів.

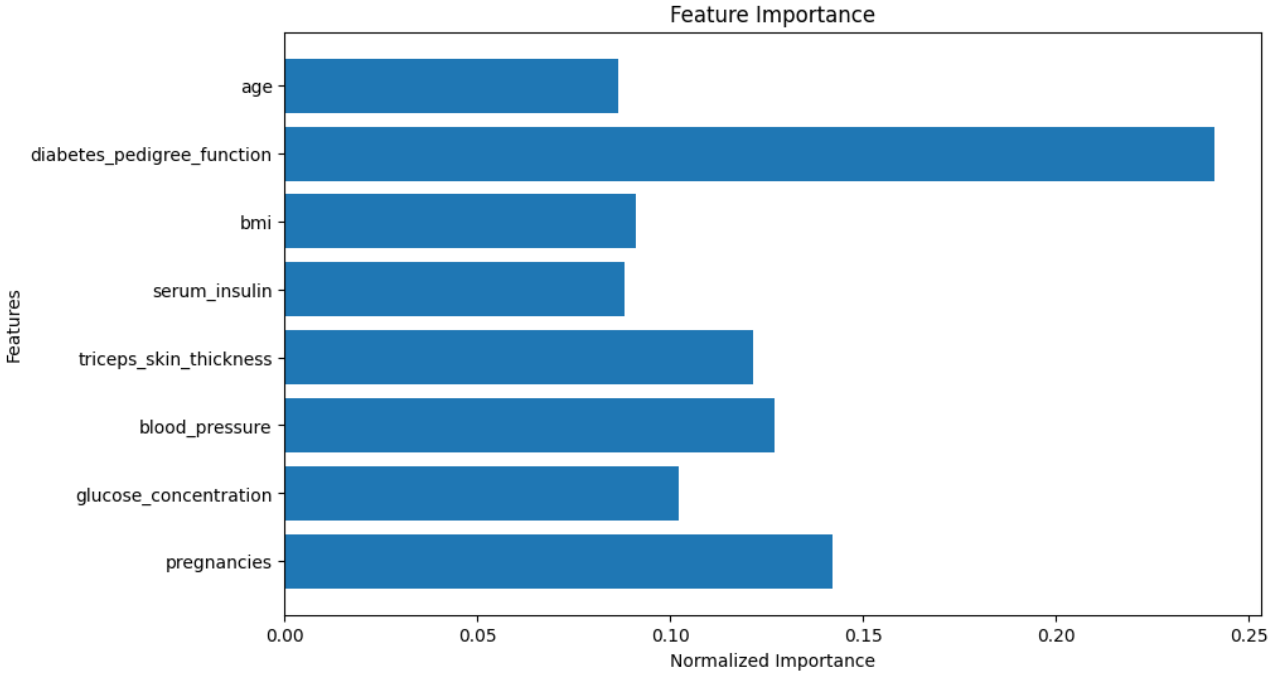

Рис. 39. Вплив різних ознак на тренування моделі (feature importance)

Як бачимо із рис. 39, найбільшу цінність становить стовпець 'diabetes_pedigree_function', а найменшу – стовпці 'bmi' та 'age'.

Для початку спробуємо видалити стовпець 'triceps_skin_thickness', 30% вмісту якого є імпутованими значеннями. Його важливість становить приблизно 12.5%. Результати тренування моделі наведено на рис. 40 та у табл.3.

```
23/23 [==============================] - 0s 1ms/step - loss: 0.5080 - accuracy: 0.7610
Accuracy: 76.10
23/23 [==============================] - 0s 2ms/step - loss: 0.4965 - accuracy: 0.7569
Accuracy: 75.69
23/23 [==============================] - 0s 2ms/step - loss: 0.5654 - accuracy: 0.7141
Accuracy: 71.41
23/23 [==============================] - 0s 1ms/step - loss: 0.5174 - accuracy: 0.7459
Accuracy: 74.59
23/23 [==============================] - 0s 2ms/step - loss: 0.4794 - accuracy: 0.7749
Accuracy: 77.49
```

Рис. 40. Результати тренування моделі без стовпця 'triceps_skin_thickness' (5 експериментів)

Таблиця 3.

Точність моделей нейронної мережі

№ експерименту	1	2	3	4	5
Точність моделі (%)	76.10	75.69	71.41	74.59	77.49

Середнє значення точності становить 75.056%. З цього можна зробити висновок, що видалення стовпця 'triceps_skin_thickness' із списку ознак майже не вплинуло на точність моделі.

Далі спробуємо видалити найважливішу ознаку - 'diabetes_pedigree_function'. Результати наведені на рис. 41 та у табл. 4.

```
23/23 [==============================] - 0s 1ms/step - loss: 0.5122 - accuracy: 0.7597
Accuracy: 75.97
23/23 [==============================] - 0s 1ms/step - loss: 0.4980 - accuracy: 0.7680
Accuracy: 76.80
23/23 [==============================] - 0s 1ms/step - loss: 0.4895 - accuracy: 0.7652
Accuracy: 76.52
23/23 [==============================] - 0s 2ms/step - loss: 0.5384 - accuracy: 0.7472
Accuracy: 74.72
23/23 [==============================] - 0s 730us/step - loss: 0.4698 - accuracy: 0.7831
Accuracy: 78.31
```

Рис. 41. Результати тренування моделі без стовпця 'diabetes_pedigree_function' (5 експериментів)

Таблиця 4.

Точність моделей нейронної мережі

№ експерименту	1	2	3	4	5
Точність моделі (%)	75.97	76.80	76.52	74.72	78.31

Середнє значення точності становить 76.464%. З цього можна зробити висновок, що видалення стовпця 'diabetes_pedigree_function' із списку ознак також майже не вплинуло на точність моделі.

Цікавим також є і графік розподілу важливості ознак, які залишилися (рис. 42). Внаслідок видалення найціннішого стовпця, цінність всіх інших ознак під час тренування моделі була майже однаковою (значно вирізняється лише стовпець 'glucose_concentration').

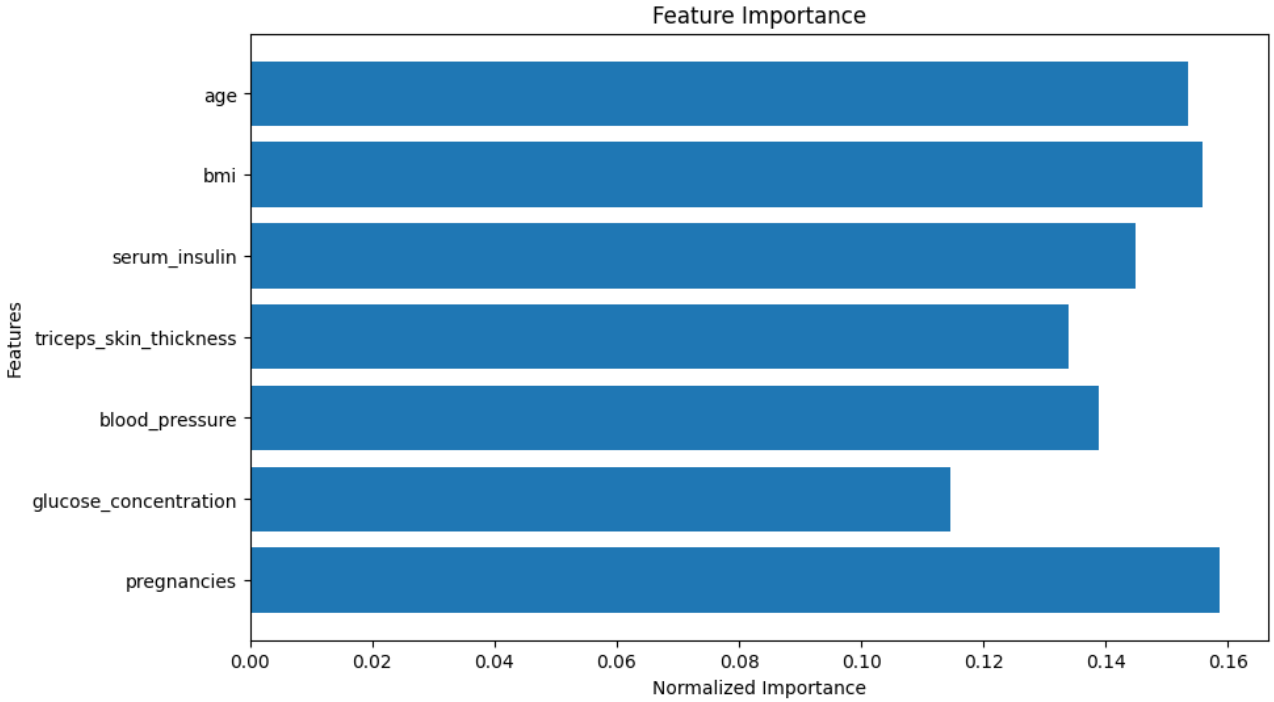

Рис. 42. Вплив різних ознак на тренування моделі (feature importance) без стовпця 'diabetes_pedigree_function'

Наступним кроком вилучимо дві найцінніші ознаки: 'diabetes_pedigree_function' та 'pregnancies'. Результати наведено нижче.

```
23/23 [==============================] - 0s 1ms/step - loss: 0.4885 - accuracy: 0.7638
Accuracy: 76.38
23/23 [==============================] - 0s 1ms/step - loss: 0.4957 - accuracy: 0.7707
Accuracy: 77.07
23/23 [==============================] - 0s 595us/step - loss: 0.5173 - accuracy: 0.7279
Accuracy: 72.79
23/23 [==============================] - 0s 2ms/step - loss: 0.5198 - accuracy: 0.7431
Accuracy: 74.31
23/23 [==============================] - 0s 935us/step - loss: 0.4925 - accuracy: 0.7831
Accuracy: 78.31
```

Рис. 43. Результати тренування моделі без стовпців 'diabetes_pedigree_function' та 'pregnancies' (5 експериментів)

Таблиця 5.

Точність моделей нейронної мережі

№ експерименту	1	2	3	4	5
Точність моделі (%)	76.38	77.07	72.79	74.31	78.31

Середнє значення точності становить 75.772%. З цього можна зробити висновок, що видалення стовпців 'diabetes_pedigree_function' та 'pregnancies' із списку ознак також не вплинуло на точність моделі.

Нарешті, видалимо 4 найвпливовіші ознаки: 'diabetes_pedigree_function', 'pregnancies', 'blood_pressure' та 'triceps_skin_thickness'. Результати експериментів наведено на рис. 44 та у табл. 6.

```
23/23 [==============================] - 0s 2ms/step - loss: 0.5225 - accuracy: 0.7486
Accuracy: 74.86
23/23 [==============================] - 0s 710us/step - loss: 0.4888 - accuracy: 0.7707
Accuracy: 77.07
23/23 [==============================] - 0s 1ms/step - loss: 0.5694 - accuracy: 0.6878
Accuracy: 68.78
23/23 [==============================] - 0s 1ms/step - loss: 0.5526 - accuracy: 0.7293
Accuracy: 72.93
23/23 [==============================] - 0s 1ms/step - loss: 0.4750 - accuracy: 0.7693
Accuracy: 76.93
```

Рис. 43. Результати тренування моделі без стовпців 'diabetes_pedigree_function', 'pregnancies', 'blood_pressure' та 'triceps_skin_thickness' (5 експериментів)

Таблиця 6.

Точність моделей нейронної мережі

№ експерименту	1	2	3	4	5
Точність моделі (%)	74.86	77.07	68.78	72.93	76.93

Середнє значення точності становить 74.114%. З цього можна зробити висновок, що видалення стовпців 'diabetes_pedigree_function' та 'pregnancies' із списку ознак також не вплинуло на точність моделі.

Результати проведених експериментів із видаленням ознак наведені у табл. 7.

Таблиця 7.

Точність моделей нейронної мережі

Видалені стовпці	Середня точність моделі для 5-ти експериментів (%)
'triceps_skin_thickness'	75.056
'diabetes_pedigree_function'	76.464
'diabetes_pedigree_function', 'pregnancies'	75.772
'diabetes_pedigree_function', 'pregnancies', 'blood_pressure', 'triceps_skin_thickness'	74.114

Отже, видалення однієї, двох і навіть чотирьох ознак із вектора ознак не вплинуло на точність моделі. Пояснити отримані результати можна так:

1) Надмірність в ознаках: якщо дві або більше ознак сильно корельовані або зайві, видалення однієї може не мати значного впливу на продуктивність моделі. Наприклад, ознаки pregnancies та age мають значну кореляцію, тому видалення однієї із них не є критичним для моделі.
2) Невеликий обсяг даних: оскільки датасет відносно малий (724 елементи), вплив видалення ознак може бути менш вираженим. З більшою кількістю даних модель матиме більше шансів помітити відсутність важливих ознак.
3) Складні взаємодії: нейронні мережі здатні фіксувати складні взаємодії між ознаками. Навіть якщо ви видалити одну чи декілька ознак, мережа все одно зможе вивчити подібні шаблони за допомогою інших ознак.

4.2) Changing the architecture of the model

Збільшення кількості нейронів

Спочатку спробуємо збільшити кількість нейронів на кожному шарі:

1) На першому прихованому шарі (він також є і вхідним шаром) 16 нейронів;
2) На другому прихованому шарі 32 нейрони.

Тобто реалізовано архітектуру 16/32/1.

```
# define the keras model
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(16, input_shape=(len(X.columns), ), activation='relu'))
model.add(tf.keras.layers.Dense(32, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

Рис. 44. Архітектура моделі 16/32/1

Результати п’яти експериментів наведено нижче.

```
23/23 [==============================] - 0s 1ms/step - loss: 0.4778 - accuracy: 0.7831
Accuracy: 78.31
23/23 [==============================] - 0s 2ms/step - loss: 0.4176 - accuracy: 0.8094
Accuracy: 80.94
23/23 [==============================] - 0s 711us/step - loss: 0.4408 - accuracy: 0.7928
Accuracy: 79.28
23/23 [==============================] - 0s 2ms/step - loss: 0.4353 - accuracy: 0.7887
Accuracy: 78.87
23/23 [==============================] - 0s 1ms/step - loss: 0.4282 - accuracy: 0.7942
Accuracy: 79.42
```

Рис. 45. Результати тренування моделі 16/32/1 (5 експериментів)

Таблиця 8.

Точність моделей нейронної мережі 16/32/1

№ експерименту	1	2	3	4	5
Точність моделі (%)	78.31	80.94	79.28	78.87	79.42

Середнє значення точності становить 79.364%. Спостерігається покращення точності порівняно зі моделлю 12/8/1 із туторіалу, для якої середнє значення точності становило 76.988%.

Модель 32/128/1

Продовжимо додавати нейрони до кожного прихованого шару: наступна модель матиме архітектуру 32/128/1 (рис. 46).

```
# define the keras model
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(32, input_shape=(len(X.columns), ), activation='relu'))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

Рис. 46. Архітектура моделі 32/128/1

```
23/23 [==============================] - 0s 1ms/step - loss: 0.3579 - accuracy: 0.8315
Accuracy: 83.15
23/23 [==============================] - 0s 1ms/step - loss: 0.3142 - accuracy: 0.8674
Accuracy: 86.74
23/23 [==============================] - 0s 1ms/step - loss: 0.3183 - accuracy: 0.8633
Accuracy: 86.33
23/23 [==============================] - 0s 1ms/step - loss: 0.3769 - accuracy: 0.8204
Accuracy: 82.04
23/23 [==============================] - 0s 710us/step - loss: 0.3079 - accuracy: 0.8605
Accuracy: 86.05
```

Рис. 47. Результати тренування моделі 32/128/1 (5 експериментів)

Таблиця 9.

Точність моделей нейронної мережі 32/128/1

№ експерименту	1	2	3	4	5
Точність моделі (%)	83.15	86.74	86.33	82.04	86.05

Середнє значення точності становить 84.862%. Спостерігається покращення точності порівняно зі моделлю 16/32/1.

Модель 128/512/1

Наступним кроком розглянемо модель 128/512/1 (рис. 48).

```
# define the keras model
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(128, input_shape=(len(X.columns), ), activation='relu'))
model.add(tf.keras.layers.Dense(512, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

Рис. 48. Архітектура моделі 128/512/1

```
23/23 [==============================] - 0s 1ms/step - loss: 0.2162 - accuracy: 0.9116
Accuracy: 91.16
23/23 [==============================] - 0s 1ms/step - loss: 0.1895 - accuracy: 0.9296
Accuracy: 92.96
23/23 [==============================] - 0s 1ms/step - loss: 0.2147 - accuracy: 0.9047
Accuracy: 90.47
23/23 [==============================] - 0s 1ms/step - loss: 0.1591 - accuracy: 0.9434
Accuracy: 94.34
23/23 [==============================] - 0s 1ms/step - loss: 0.1881 - accuracy: 0.9309
Accuracy: 93.09
```

Рис. 49. Результати тренування моделі 128/512/1 (5 експериментів)

Таблиця 10.

Точність моделей нейронної мережі 128/512/1

№ експерименту	1	2	3	4	5
Точність моделі (%)	91.16	92.96	90.47	94.34	93.09

Середнє значення точності становить 92.404%. Спостерігається покращення точності порівняно зі моделлю 32/128/1.

Модель 256/512/1

Продовжимо додавати нейрони до першого прихованого шару: наступна модель матиме архітектуру 256/512/1 (рис. 50).

```
# define the keras model
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(256, input_shape=(len(X.columns), ), activation='relu'))
model.add(tf.keras.layers.Dense(512, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

Рис. 50. Архітектура моделі 256/512/1

```
23/23 [==============================] - 0s 1ms/step - loss: 0.2419 - accuracy: 0.8895
Accuracy: 88.95
23/23 [==============================] - 0s 1ms/step - loss: 0.2239 - accuracy: 0.9075
Accuracy: 90.75
23/23 [==============================] - 0s 1ms/step - loss: 0.2288 - accuracy: 0.9075
Accuracy: 90.75
23/23 [==============================] - 0s 1ms/step - loss: 0.2144 - accuracy: 0.9144
Accuracy: 91.44
23/23 [==============================] - 0s 730us/step - loss: 0.1762 - accuracy: 0.9323
Accuracy: 93.23
```

Рис. 51. Результати тренування моделі 256/512/1 (5 експериментів)

Таблиця 11.

Точність моделей нейронної мережі 256/512/1

№ експерименту	1	2	3	4	5
Точність моделі (%)	88.95	90.75	90.75	91.44	93.23

Середнє значення точності становить 91.024%. Спостерігається погіршення точності порівняно зі моделлю 128/512/1, однак точність все ж краща, ніж в інших попередніх моделях. Отже, подальше збільшення нейронів у прихованих шарах швидше за все не призведе до покращення точності моделі.

Підсумуємо результати експериментів із зміною кількості нейронів у нейронній мережі із двома прихованими шарами.

Таблиця 12.

Точність моделей нейронної мережі із двома прихованими шарами

Архітектура моделі	12/8/1	16/32/1	32/128/1	128/512/1	256/512/1
Середнє значення точності для 5 експериментів, %	76.988	79.364	84.862	92.404	91.024
Максимальна досягнута точність, %	78.59	80.94	86.74	94.34	93.23

Continue training the model 256/512/1

Розглянувши поведінку останньої досліджуваної моделі 256/512/1 (рис. 52 і рис. 53), легко помітити, що точність моделі можна покращити, збільшивши кількість епох. Обгрунтувати таке рішення можна графіком втрат (рис. 53): на 150-ій епосі втрати продовжують зменшуватися, а отже, є сенс продовжити тренування моделі до моменту досягнення мінімуму втрат.

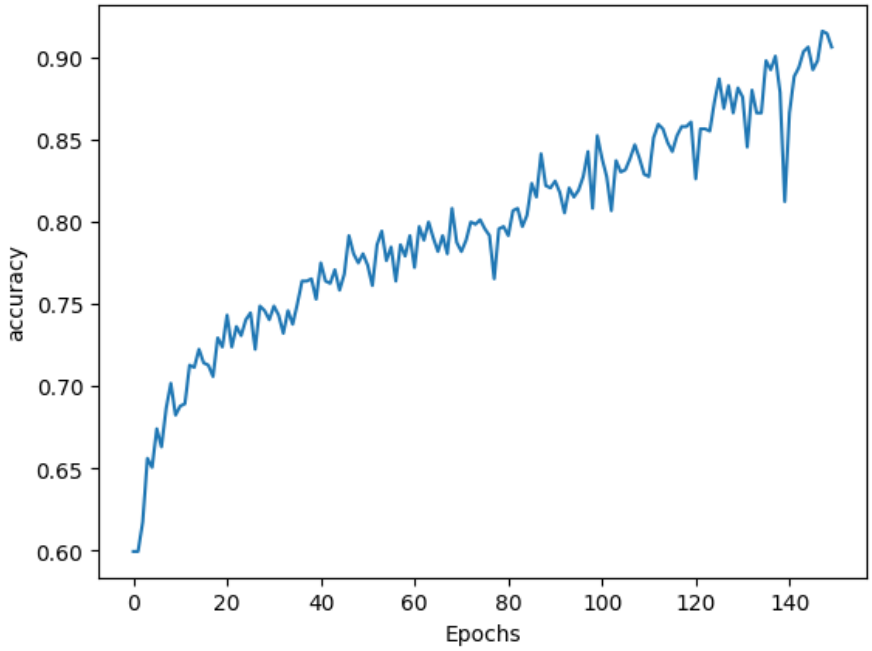

Рис. 52. Зміна точності протягом епох під час навчання моделі 256/512/1

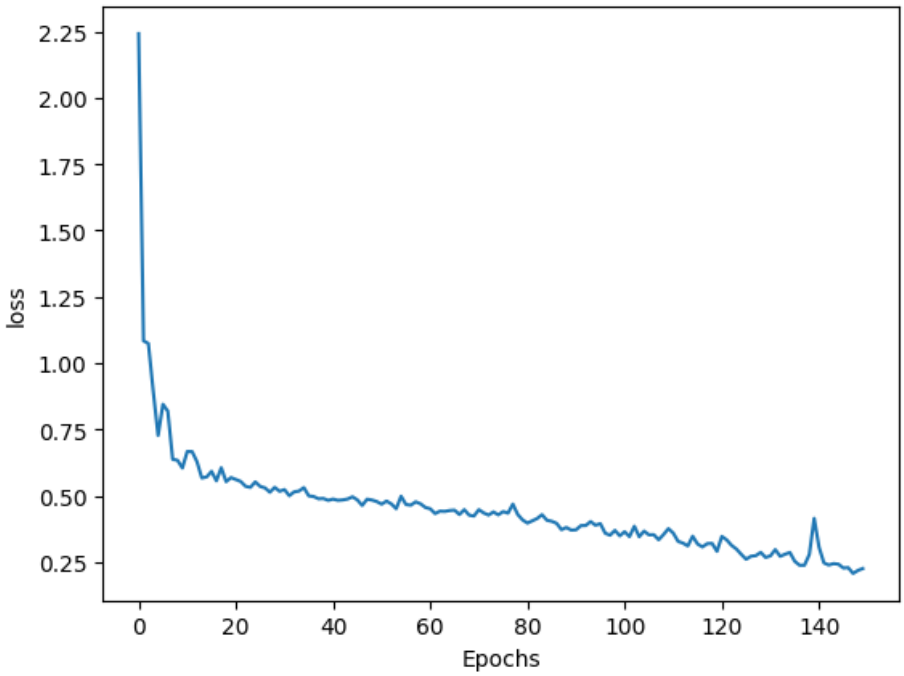

Рис. 53. Зміна втрат протягом епох під час навчання моделі 256/512/1

Тому було продовжено навчання моделі з кроком 50 епох. Уже на 550-ій епосі значення точності досягнуло 100%, а втрати становили 0.0077.

```
23/23 [==============================] - 0s 2ms/step - loss: 0.0077 - accuracy: 1.0000
Accuracy: 100.00
```

Рис. 54. Результат дотреновування моделі 256/512/1 протягом 550 епох

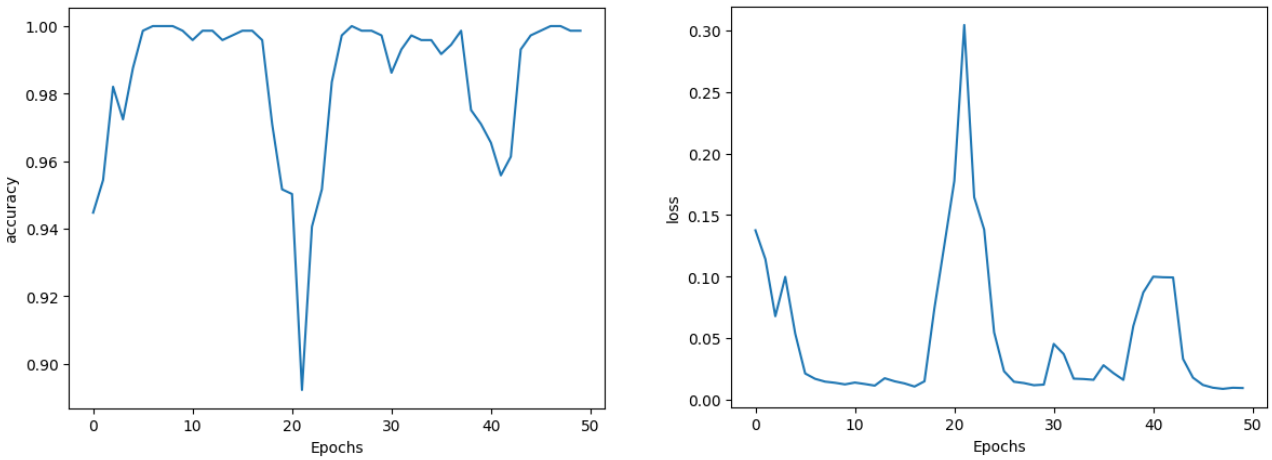

Рис. 55. Зміна точності і втрат протягом останніх 50/550 епох під час навчання моделі 256/512/1

Наступним експериментом було тренування моделі з такою ж архітектурою (256/512/1) протягом 550 епох відразу (без дотреновування з кроком 50).

```python
# define the keras model
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(256, input_shape=(len(X.columns), ), activation='relu'))
model.add(tf.keras.layers.Dense(512, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))

# compile the keras model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# fit the keras model on the dataset
history = model.fit(X, y, epochs = 550, batch_size=10, verbose=0)

# evaluate the keras model
_, accuracy = model.evaluate(X, y)
print('Accuracy: %.2f' % (accuracy*100))
```

```
23/23 [==============================] - 0s 2ms/step - loss: 0.0117 - accuracy: 1.0000
Accuracy: 100.00
```

Рис. 56. Результат тренування моделі 256/512/1 протягом 550 епох

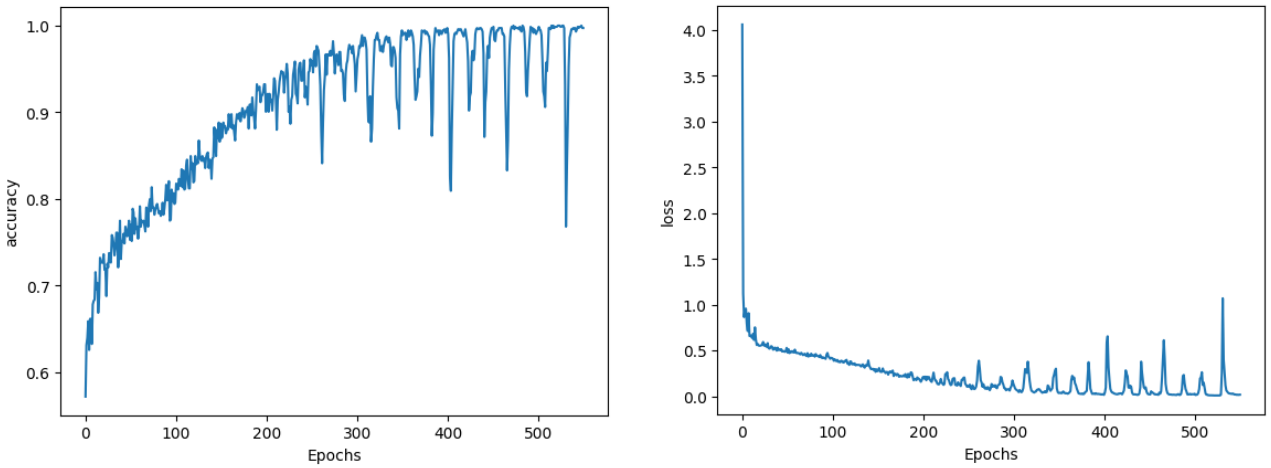

Рис. 57. Зміна точності і втрат протягом 550 епох під час навчання моделі 256/512/1

Із графіків зміни точності і втрат можна помітити, що починаючи із 250-ої епохи точність моделі почала різко коливатися від 95% до 85%, а після 400-ої епохи - від 98-100% до 75-80%. Аналогічна поведінка характерна і для втрат.

Такі коливання як у точності навчання, так і у втратах можуть вказувати на те, що модель перенавчилася (overfitting). Перенавчання (overfitting) відбувається, коли модель занадто добре вивчає навчальні дані, включаючи їх шум і викиди, що може призвести до поганого узагальнення нових, невидимих даних.

Дослідження перенавчання моделі (overfitting)

Коливання точності та втрат вказують на те, що модель могла запам’ятати навчальні дані замість того, щоб вивчати базові закономірності. Це може призвести до зниження продуктивності під час тестування на невідомих даних.

Для перевірки твердження про overfitting було здійснено поділ тренувального датасету на дані для тренування та дані для валідації у співвідношенні 80:20 (579 записів – тренування, 145 - валідація).

```
19/19 [==============================] - 0s 2ms/step - loss: 0.0313 - accuracy: 0.9965
5/5 [==============================] - 0s 6ms/step - loss: 1.4714 - accuracy: 0.7034
Accuracy: 99.65; Loss: 0.03
Validation accuracy: 70.34; validation loss: 1.47
```

Рис. 58. Результат тренування моделі 256/512/1 протягом 550 епох

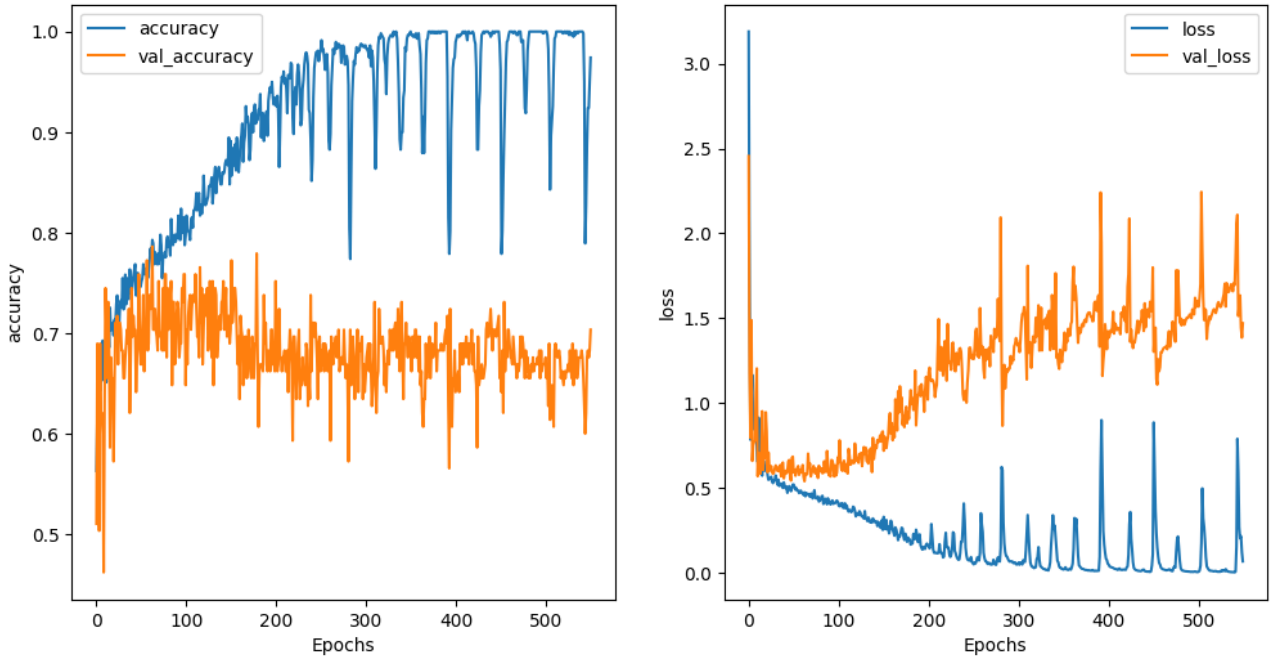

Рис. 59. Зміна тренувальних і валідаційних точності і втрат протягом 550 епох під час навчання моделі 256/512/1

Із рис. 59 добре видно, що приблизно після 100-ої епохи val_loss перестає зменшуватися, а після 150-ої починає збільшуватися. Це безсумнівно свідчить про overfitting моделі (моделі перенавчається).

128/512/1

Overfitting також спостерігається і для моделі 128/512/1 протягом 150 епох.

```
19/19 [==============================] - 0s 1ms/step - loss: 0.1894 - accuracy: 0.9240
5/5 [==============================] - 0s 2ms/step - loss: 0.7787 - accuracy: 0.7172
Accuracy: 92.40; Loss: 0.19
Validation accuracy: 71.72; validation loss: 0.78
```

Рис. 60. Результат тренування моделі 128/512/1 протягом 150 епох

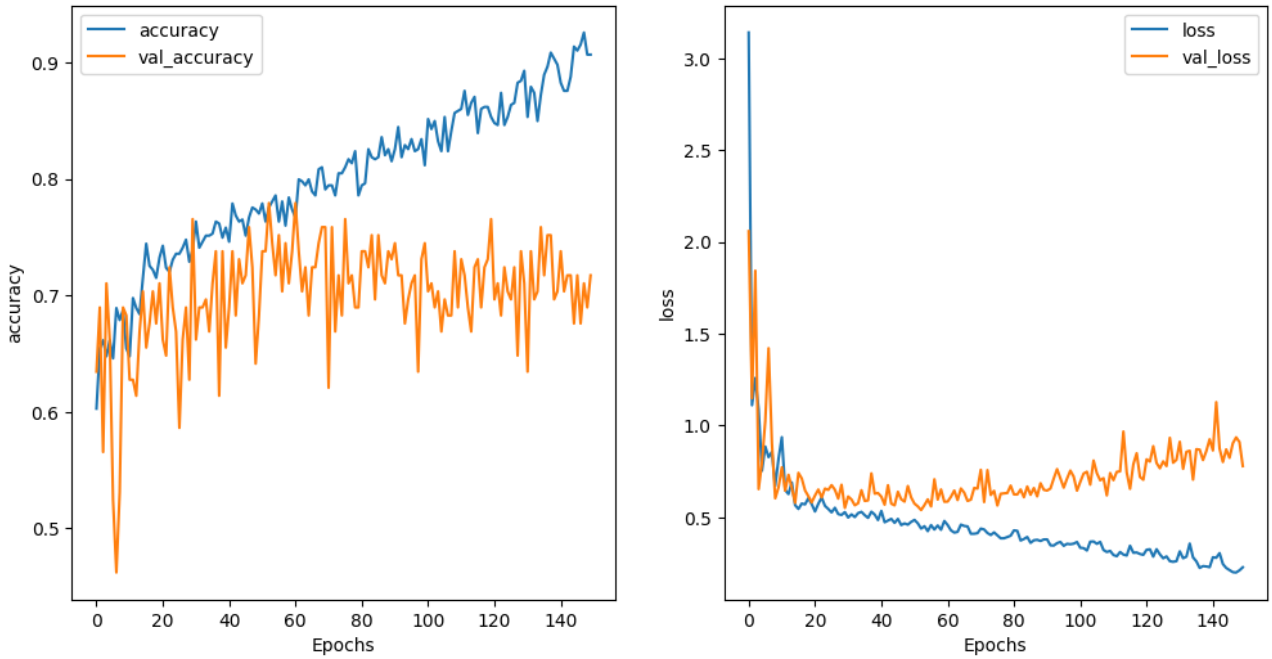

Рис. 61. Зміна тренувальних і валідаційних точності і втрат протягом 150 епох під час навчання моделі 128/512/1

32/128/1

Однак для моделі 32/128/1 overfitting вже майже не зустрічається:

```
19/19 [==============================] - 0s 2ms/step - loss: 0.3060 - accuracy: 0.8653
5/5 [==============================] - 0s 5ms/step - loss: 0.5611 - accuracy: 0.7862
Accuracy: 86.53; Loss: 0.31
Validation accuracy: 78.62; validation loss: 0.56
```

Рис. 62. Результат тренування моделі 32/128/1 протягом 150 епох

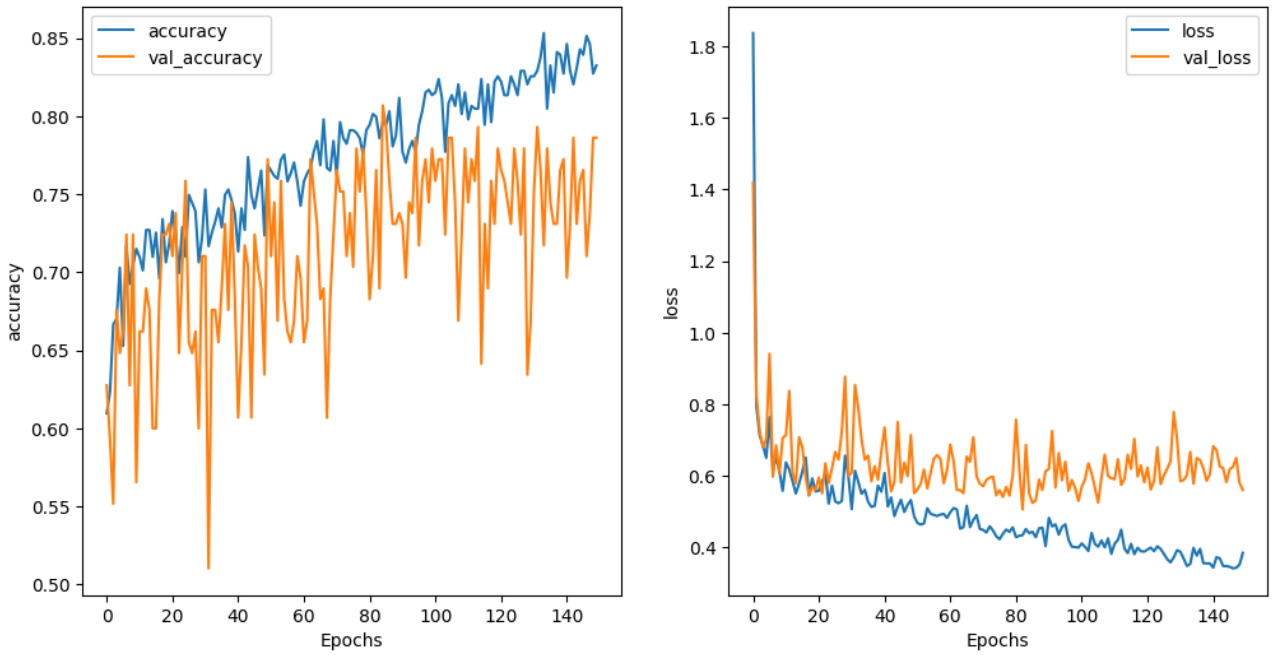

Рис. 63. Зміна тренувальних і валідаційних точності і втрат протягом 150 епох під час навчання моделі 32/128/1

16/32/1

```
19/19 [==============================] - 0s 2ms/step - loss: 0.4861 - accuracy: 0.7634
5/5 [==============================] - 0s 4ms/step - loss: 0.6126 - accuracy: 0.7310
Accuracy: 76.34; Loss: 0.49
Validation accuracy: 73.10; validation loss: 0.61
```

Рис. 64. Результат тренування моделі 16/32/1 протягом 150 епох

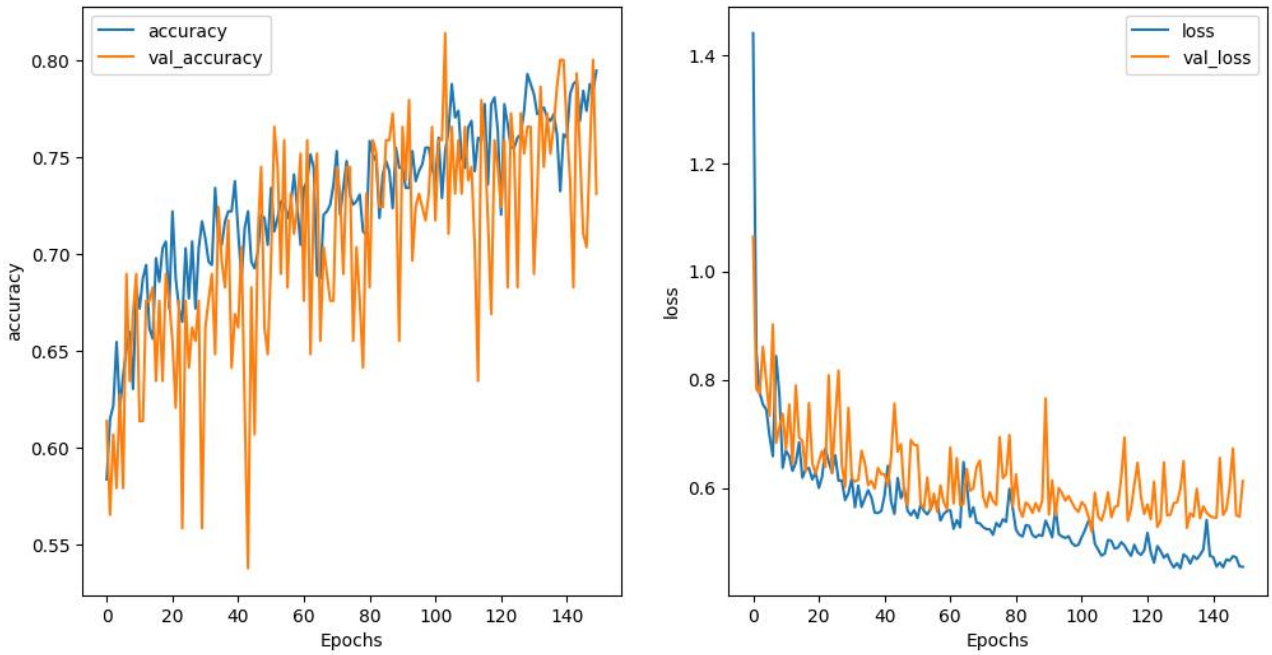

Рис. 65. Зміна тренувальних і валідаційних точності і втрат протягом 150 епох під час навчання моделі 16/32/1

12/8/1

```
19/19 [==============================] - 0s 1ms/step - loss: 0.5051 - accuracy: 0.7496
5/5 [==============================] - 0s 2ms/step - loss: 0.5275 - accuracy: 0.7172
Accuracy: 74.96; Loss: 0.51
Validation accuracy: 71.72; validation loss: 0.53
```

Рис. 66. Результат тренування моделі 12/8/1 протягом 150 епох

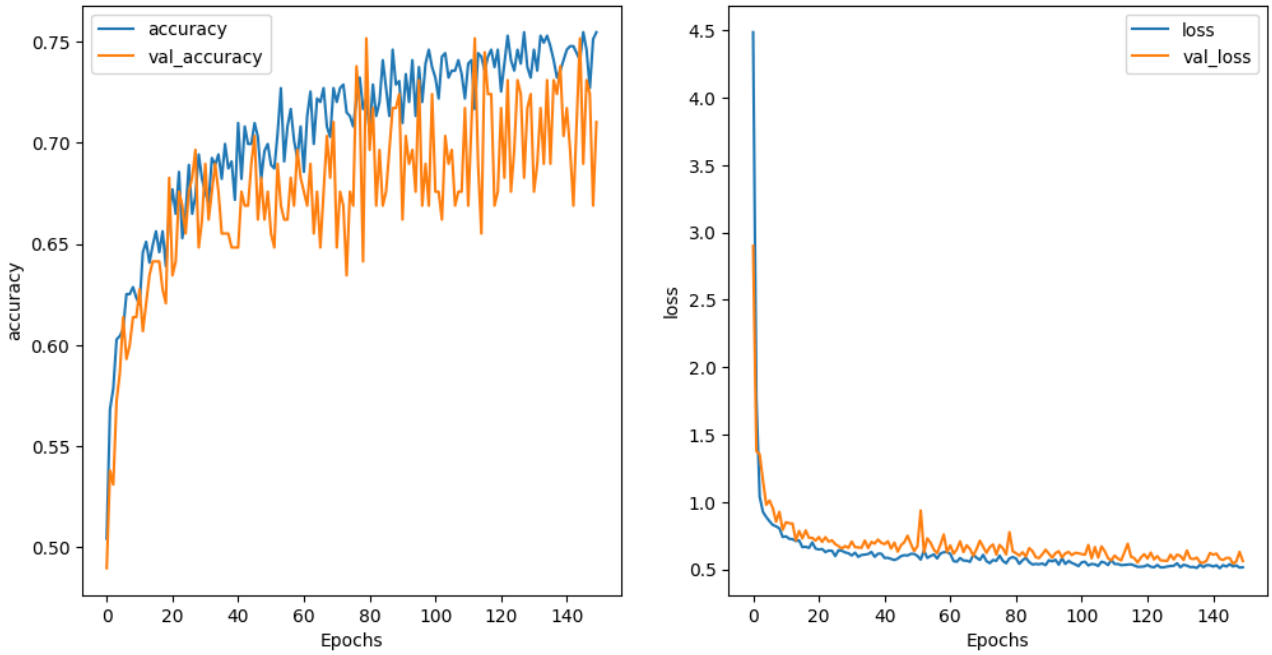

Рис. 67. Зміна тренувальних і валідаційних точності і втрат протягом 150 епох під час навчання моделі 12/8/1

Таблиця 13.

Вплив складності архітектури моделі на її перенавчання (overfitting)

Архітектура моделі	12/8/1	16/32/1	32/128/1	128/512/1
Тренувальна точність, %	74.96	76.34	86.53	92.40
Точність валідації, %	71.72	73.10	78.62	71.72
Наявність overfitting	Ні	Ні	Ні	Так

Можна виділити декілька можливих причин перенавчання моделі:

1) Невеликий розмір набору даних: із таким невеликим набором даних (579 навчальних зразків), нейронним мережам може бути важко добре узагальнювати. Наявність обмеженої кількості даних полегшує моделі запам’ятовування навчальних прикладів, що призводить до overfitting.
2) Складність моделі: нейронні мережі з великою кількістю рівнів і параметрів можуть з легкістю ідеально підганяти навчальні дані, особливо якщо набір даних невеликий. Складна модель має більше можливостей для вивчення шуму в навчальних даних.
3) Забагато епох: тренування для великої кількості епох без передчасної зупинки може спричинити перенавчання. Модель починає точніше підбирати навчальні дані, включаючи шум.
4) Набір даних перевірки: якщо набір даних перевірки дуже малий (145 зразків), це може бути поганим відображенням ефективності узагальнення моделі.

У контексті даної задачі можливі такі способи пом’якшити або усунути перенавчання моделі:

1) Зменшення складності моделі: використання простішої архітектури моделі з меншою кількістю прихованих шарів і нейронів може допомогти уникнути перенавчання (що продемонстровано вище).
2) Використання регулярізації: використання випадання (Dropout-шари), регулярізація L1/L2 або обидва способи одночасно.
3) Рання зупинка: можна використовувати ранню зупинку, щоб відстежувати втрати перевірки та припиняти навчання, коли вони починають збільшуватися або коли досягнуто необхідний рівень точності (наприклад, використовуючи callback-и після кожної епохи).

4.3) The impact of data normalization

Дослідимо вплив нормалізації даних на навчання моделі нейронної мережі на прикладі архітектур 12/8/1 (архітектура із туторіалу) та 128/512/1 (архітектура, яка надала найбільшу точність).

12/8/1

```
23/23 [==============================] - 0s 1ms/step - loss: 0.3387 - accuracy: 0.8384
Accuracy: 83.84
23/23 [==============================] - 0s 711us/step - loss: 0.3589 - accuracy: 0.8260
Accuracy: 82.60
23/23 [==============================] - 0s 735us/step - loss: 0.3495 - accuracy: 0.8301
Accuracy: 83.01
23/23 [==============================] - 0s 1ms/step - loss: 0.3454 - accuracy: 0.8315
Accuracy: 83.15
23/23 [==============================] - 0s 2ms/step - loss: 0.3294 - accuracy: 0.8536
Accuracy: 85.36
```

Рис. 68. Результати тренування моделі 12/8/1 (5 експериментів)

Таблиця 14.

Точність моделей 12/8/1 нейронної мережі

№ експерименту	1	2	3	4	5
Точність моделі (%)	83.84	82.60	83.01	83.15	85.36

Середнє значення точності становить 83.592%. Спостерігається покращення точності порівняно зі моделлю 12/8/1 із туторіалу без нормалізації даних, для якої середнє значення точності становило 76.988%.

128/512/1

```
23/23 [==============================] - 0s 1ms/step - loss: 0.0018 - accuracy: 1.0000
Accuracy: 100.00
23/23 [==============================] - 0s 1ms/step - loss: 0.2113 - accuracy: 0.9434
Accuracy: 94.34
23/23 [==============================] - 0s 1ms/step - loss: 6.9268e-04 - accuracy: 1.0000
Accuracy: 100.00
23/23 [==============================] - 0s 2ms/step - loss: 9.4953e-04 - accuracy: 1.0000
Accuracy: 100.00
23/23 [==============================] - 0s 1ms/step - loss: 9.5519e-04 - accuracy: 1.0000
Accuracy: 100.00
```

Рис. 69. Результати тренування моделі 128/512/1 (5 експериментів)

Таблиця 15.

Точність моделей 128/512/1 нейронної мережі

№ експерименту	1	2	3	4	5
Точність моделі (%)	100.00	94.34	100.00	100.00	100.00

Середнє значення точності становить 98.868%. Спостерігається значне покращення точності порівняно зі моделлю 128/512/1 без нормалізації даних, для якої середнє значення точності становило 92.404%.

Таблиця 16.

Точність моделей нейронної мережі із двома прихованими шарами без нормалізації та з нормалізацією даних

Архітектура моделі	Середнє значення точності для 5 експериментів без нормалізації, %	Середнє значення точності для 5 експериментів з нормалізацією, %	Максимальна досягнута точність без нормалізації, %	Максимальна досягнута точність з нормалізацією, %
12/8/1	76.988	83.592	78.59	85.36
128/512/1	92.404	98.868	94.34	100.00

Також варто зазначити вплив нормалізації даних на швидкість досягнення певної точності. Наприклад, для експерименту використано архітектуру 128/512/1 із завданням досягнути точність 100%. Тренування моделі припиняється відразу під час досягнення потрібної точності, що забезпечується використанням класу myCallback, похідного від класу tf.keras.callbacks.Callback (рис. 70).

```
class myCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if(logs.get('accuracy') == 1):
            print('\nAccuracy is bigger than 1 so cancelling training!')
            self.model.stop_training = True
```

Рис. 70. Код класу myCallback

```
Epoch 256/550
40/58 [===================>.....] - ETA: 0s - loss: 0.0296 - accuracy: 1.0000
Accuracy is bigger than 1 so cancelling training!
58/58 [==============================] - 0s 3ms/step - loss: 0.0335 - accuracy: 1.0000 - val_loss: 1.6251 - val_accuracy: 0.6828

Evaluating the model
19/19 [==============================] - 0s 2ms/step - loss: 0.0296 - accuracy: 0.9983
5/5 [==============================] - 0s 2ms/step - loss: 1.6251 - accuracy: 0.6828
Accuracy: 99.83; Loss: 0.03
Validation accuracy: 68.28; validation loss: 1.63
```

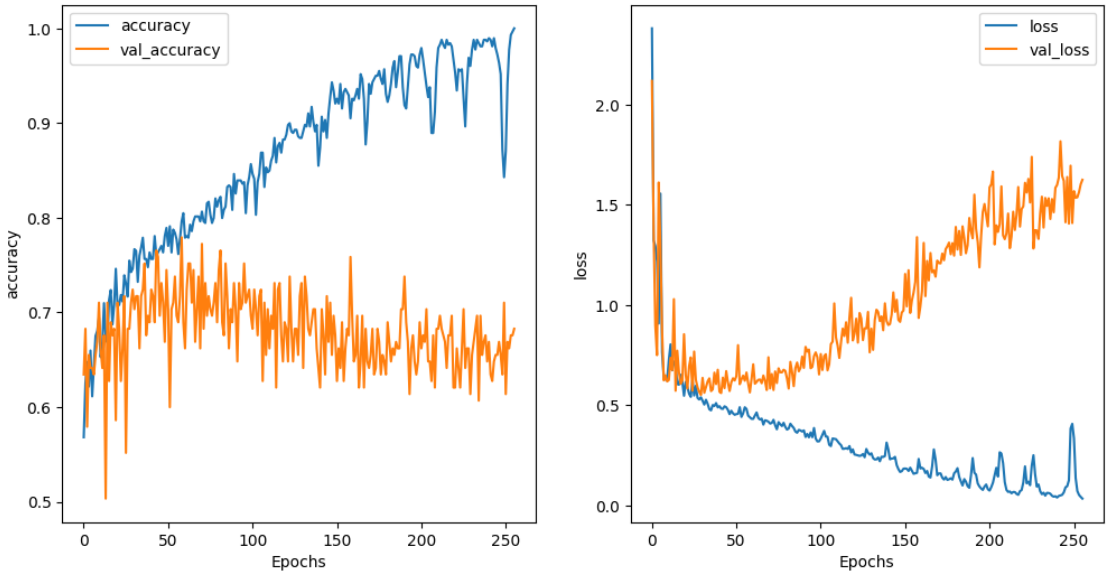

Рис. 71. Результат тренування моделі 128/512/1 без нормалізації даних

```
Epoch 56/550
46/58 [=====================>.......] - ETA: 0s - loss: 0.0213 - accuracy: 1.0000
Accuracy is bigger than 1 so cancelling training!
58/58 [==============================] - 0s 3ms/step - loss: 0.0215 - accuracy: 1.0000 - val_loss: 0.9187 - val_accuracy: 0.7655

Evaluating the model
19/19 [==============================] - 0s 2ms/step - loss: 0.0189 - accuracy: 1.0000
5/5 [==============================] - 0s 4ms/step - loss: 0.9187 - accuracy: 0.7655
Accuracy: 100.00; Loss: 0.02
Validation accuracy: 76.55; validation loss: 0.92
```

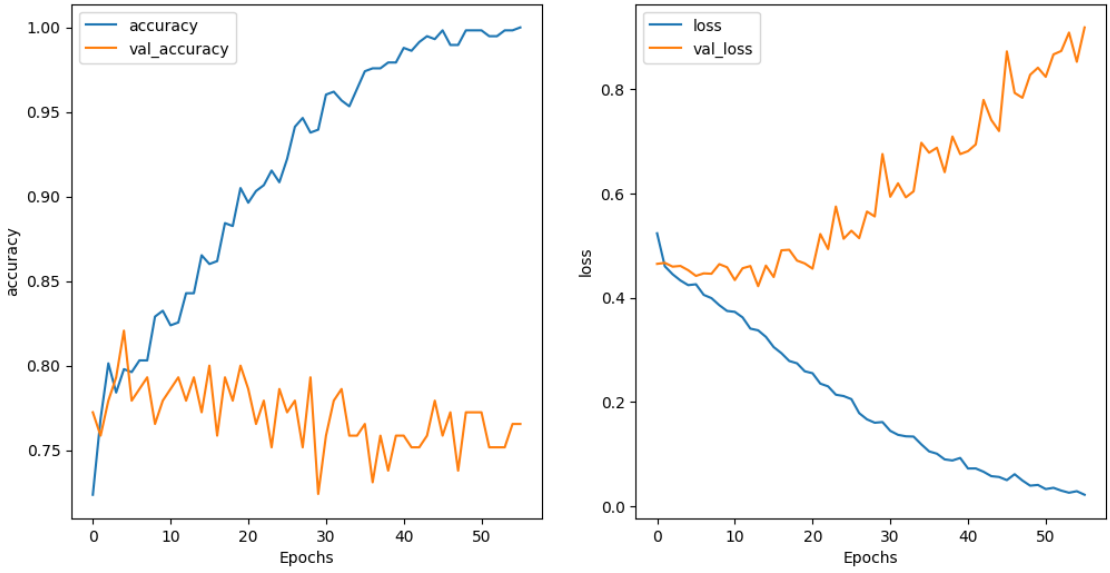

Рис. 72. Результат тренування моделі 128/512/1 з нормалізацією даних

Без нормалізації даних модель 128/512/1 досягнула точності в 100% за 256 епох (рис. 71), а з нормалізацією даних – всього за 56 епох (рис. 72). Іншими словами, нормалізація вектора ознак пришвидшила тренування моделі у 4.5 рази.

Висновки

Досліджуваний датасет Pima Indians Diabetes Database містить 768 записів та 9 стовпців, дослідивши які отримано такі результати:

1) Стовпці 'pregnancies', 'serum_insulin', 'diabetes_pedigree_function' та 'has_diabetes' не мають відсутніх значень (missing values).
2) Стовпці 'glucose_concentration', 'blood_pressure' і 'bmi' мають декілька відсутніх значень (5, 35 і 11 відповідно), замінених значенням 0. Тому рядки із такими значеннями було видалено, що призвело до зменшення розміру датасету до 724 записів.
3) Стовпець 'triceps_skin_thickness' містить 227 відсутніх/неправильних значень (0). Видалення рядків із цим значенням призвело б до втрати значної кількості даних, тому для вирішення проблеми використано KNN імпутацію (k = 5) із використанням стовпців 'serum_insulin' та 'bmi'.

Також варто зазначити, що досліджуваний датасет є незбалансованим, оскільки кількість записів стовпця 'has_diabetes' із міткою 0 приблизно вдвічі перевищує кількість записів із міткою 1.

Результати дослідження відсутності ознак наведено у табл. 17, з якої можна побачити, що видалення однієї, двох і навіть чотирьох ознак із вектора ознак фактично не вплинуло на точність моделі.

Таблиця 17.

Точність моделей нейронної мережі

Видалені стовпці	Середня точність моделі для 5-ти експериментів (%)
-	76.988
'triceps_skin_thickness'	75.056
'diabetes_pedigree_function'	76.464
'diabetes_pedigree_function', 'pregnancies'	75.772
'diabetes_pedigree_function', 'pregnancies', 'blood_pressure', 'triceps_skin_thickness'	74.114

Проведено дослідження впливу збільшення кількості нейронів у двох прихованих шарах моделі нейронної мережі, результати якого наведено у табл. 18.

Таблиця 18.

Точність моделей нейронної мережі із двома прихованими шарами

Архітектура моделі	12/8/1	16/32/1	32/128/1	128/512/1	256/512/1
Середнє значення точності для 5 експериментів, %	76.988	79.364	84.862	92.404	91.024
Максимальна досягнута точність, %	78.59	80.94	86.74	94.34	93.23

Найбільша досягнута точність для 150 епох становить 94.34% (модель 128/512/1).

Із збільшенням складності моделі тренувальна точність зростає, однак для складних моделей (наприклад, 128/512/1) в ході експериментів було виявлено процес перенавчання (overfitting).

Також зроблено висновок про покращення процесу тренування моделей внаслідок використання нормалізації даних (табл. 19).

Таблиця 19.

Точність моделей нейронної мережі із двома прихованими шарами без нормалізації та з нормалізацією даних

Архітектура моделі	Середнє значення точності для 5 експериментів без нормалізації, %	Середнє значення точності для 5 експериментів з нормалізацією, %	Максимальна досягнута точність без нормалізації, %	Максимальна досягнута точність з нормалізацією, %
12/8/1	76.988	83.592	78.59	85.36
128/512/1	92.404	98.868	94.34	100.00

Окрім того, нормалізація даних пришвидшує процес навчання (дозволяє значно швидше досягнути потрібної точності), про що свідчать результати експериментів, наведені у табл. 20.

Таблиця 20.

Час тренування моделей нейронної мережі 128/512/1 з метою досягнення точності 100% без нормалізації та з нормалізацією даних

Кількість епох з нормалізацією	56
Кількість епох без нормалізації	256