

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

А.С. Момот,
М.С. Мамута

ІНТЕЛЕКТУАЛЬНІ ТА ІНФОРМАЦІЙНІ СИСТЕМИ ПРАКТИКУМ

Навчальний посібник

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня магістра
за освітньою програмою «Комп'ютерно-інтегровані
системи та технології в приладобудуванні»
спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології»

Електронне мережне навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2022

Рецензенти: *Войцехович Валерій Степанович*, канд. фіз.-мат. наук, старший науковий співробітник відділу когерентної і квантової оптики Інституту фізики НАН України
Безугла Наталя Василівна, канд. техн. наук, доцент, доцент кафедри виробництва приладів КПІ ім. Ігоря Сікорського

Відповідальний редактор: *Боровицький Володимир Миколайович*, док. техн. наук, доцент, професор кафедри комп'ютерно-інтегрованих оптичних та навігаційних систем КПІ ім. Ігоря Сікорського

*Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол №5 від 26.05.2022 р.)
за поданням Вченої ради приладобудівного факультету
(протокол №4/22 від 26.04.2022 р.)*

Практикум містить теоретичний матеріал, приклади та комплекс завдань для комп'ютерного практикуму та самостійної роботи студентів з дисципліни «Інтелектуальні та інформаційні системи». Надано інформацію про аналіз та синтез нейронних мереж, проаналізовано методи оптимізації навчання нейронних мереж та надано рекомендації з їх використання. Висвітлено особливості побудови і використання повнозв'язних, згорткових та рекурентних мереж, а також прийоми навчання глибоких нейронних мереж, зокрема метод перенесення навчання та метод тонких налаштувань. Практикум може бути корисний студентам та фахівцям з інтелектуального аналізу даних.

Реєстр. № НП 21/22-402. Обсяг 6,8 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Перемоги, 37, м. Київ, 03056
<https://kpi.ua>

Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

© А.С. Момот, М.С. Мамута
© КПІ ім. Ігоря Сікорського, 2022

ЗМІСТ

ВСТУП	5
ПРАКТИКУМ 1. БІБЛІОТЕКА PANDAS	6
1.1. Теоретичні відомості	6
1.2. Операції з даними в Pandas	16
1.3. Оглядовий аналіз даних з Pandas.....	32
1.4. Аналіз набору даних "Титанік"	59
1.5 Завдання для самостійного виконання	73
ПРАКТИКУМ 2. НЕЙРОННІ МЕРЕЖІ В KERAS.....	80
2.1 Загальні відомості	80
2.1 Завдання для самостійного виконання	89
ПРАКТИКУМ 3. ОПТИМІЗАЦІЯ НЕЙРОННИХ МЕРЕЖ	94
3.1. Загальні відомості	94
3.2 Завдання для самостійного виконання	134
ПРАКТИКУМ 4. НЕЙРОННІ МЕРЕЖІ В ЗАДАЧАХ ПРОГНОЗУВАННЯ (РЕГРЕСІЇ)	135
4.1. Загальні відомості	135
4.2. Завдання для самостійного виконання	150
ПРАКТИКУМ 5. ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ.....	154
5.1. Загальні відомості	154
5.2. Завдання для самостійного виконання	166
ПРАКТИКУМ 6. АУГМЕНТАЦІЯ ДАНИХ.....	169

6.1. Загальні відомості	169
6.2. Завдання для самостійного виконання	180
ПРАКТИКУМ 7. ПРИЙОМИ ГЛИБИННОГО НАВЧАННЯ.....	183
7.1. Підготовка власного набору даних	183
7.2. Перенесення навчання	187
7.3. Тонкі налаштування мережі.....	194
7.4. Завдання для самостійного виконання	198
ПРАКТИКУМ 8. РЕКУРЕНТНІ НЕЙРОННІ МЕРЕЖІ.....	199
8.1. Загальні відомості	199
8.2. Завдання для самостійного виконання	216
Список рекомендованої літератури	221

ВСТУП

Навчальний посібник призначено для забезпечення інформаційними та методичними матеріалами дисципліни "Інтелектуальні та інформаційні системи".

Дисципліна "Інтелектуальні та інформаційні системи" викладається на першому курсі магістратури та має обсяг в 36 годин лекцій та 36 годин комп'ютерного практикуму. Увага приділена базовим можливостям глибинного навчання, зокрема побудові та оптимізації нейронних мереж в пакеті Anaconda з використанням мови програмування Python.

В посібнику надано загальні відомості про аналіз та синтез нейронних мереж, висвітлено особливості побудови та навчання повнозв'язних, згорткових та рекурентних нейронних мереж, проаналізовано основні методи оптимізації навчання нейронних мереж, розглянуто прийоми глибинного навчання, а саме: перенесення навчання та тонкі налаштування нейронних мереж. Посібник орієнтований як на ознайомлення з базовими можливостями глибинного навчання, так і на вивчення перспективних методів та алгоритмів глибинного навчання та їх практичної реалізації. З метою кращого засвоєння матеріалу практикум містить приклади програмної реалізації нейронних мереж різної архітектури з детальним аналізом їх побудови та оптимізації та завдання для самостійного виконання.

Метою навчального посібника є допомога студентам в самостійному вивченні відповідних розділів навчальної дисципліни та в набутті навичок застосування штучних нейронних мереж під час проектування інтелектуальних приладів та систем. Посібник може використовуватись для самостійної роботи та дистанційного навчання.

ПРАКТИКУМ 1. БІБЛІОТЕКА PANDAS

1.1. Теоретичні відомості

Pandas – це бібліотека, яка надає зручні інструменти для зберігання даних і роботи з ними. Використовується в області аналізу даних та машинного навчання.

Pandas чудово підходить для роботи з одновимірними і двовимірними таблицями даних, надає можливість працювати з файлами CSV, таблицями Excel, SQL та багатьма іншими форматами.

- Повна документація за бібліотекою: <https://pandas.pydata.org/pandas-docs/stable/>

Для початку роботи, бібліотеку необхідно імпортувати:

```
import pandas as pd
```

Зазвичай у парі з Pandas використовується бібліотека NumPy.

```
import numpy as np
```

Структура даних Series

Series – це маркована одновимірна структура даних, яку можна уявити як таблицю з одним рядком. З Series можна працювати як зі звичайним масивом (звертатися за номером індексу), а також як з асоційованим масивом, тобто використовувати ключ для доступу до елементів даних.

Створити структуру Series можна на базі різних типів даних:

- словники;
- списки;
- масиви з numpy: ndarray;
- скалярні величини.

Конструктор класу Series виглядає наступним чином:

- *data* – масив, словник або скалярне значення, на базі якого буде побудована Series;
- *index* – список міток, який буде використовуватися для доступу до елементів Series. Довжина списку повинна дорівнювати довжині *data*;
- *dtype* – об'єкт `numpy.dtype`, який визначає тип даних;
- *copy* – створює копію масиву даних, якщо параметр дорівнює `True`, в іншому випадку нічого не робить.

У більшості випадків під час створення Series використовують тільки перші два параметра. Розглянемо різні варіанти, як це можна зробити.

Створення Series зі списку

Найпростіший спосіб створити Series – це передати в якості єдиного параметра в конструктор класу список.

```
s1 = pd.Series([1, 2, 3, 4, 5])
print(s1)

>>
0    1
1    2
2    3
3    4
4    5
```

Для доступу до елементів Series, в даному випадку, можна використовувати тільки позитивні цілі числа - лівий стовпець чисел, який починається з нуля. Це і є індекси елементів структури, які показані в правому стовпці.

```
s1[1]

>>
2
```

Тепер передамо в якості другого параметру список рядків (в нашому випадку – окремі символи). Це дозволить нам звертатися до елементів структури Series не

тільки за числовим індексом, а й за міткою, що зробить роботу з таким об'єктом схожою на роботу зі словником.

```
s2 = pd.Series([1, 2, 3, 4, 5], ['a', 'b', 'c', 'd', 'e'])
print(s2)

>>

a    1
b    2
c    3
d    4
e    5
```

Зверніть увагу на лівий стовпець, в ньому містяться мітки, які ми передали в якості параметра *index* під час створення структури. Правий стовпець – це, як і раніше, елементи структури.

```
s2['b']

>>

2
```

Створення Series з масиву ndarray

Створимо масив з п'яти чисел, аналогічний списку з попереднього розділу.

```
ndarr = np.array([1, 2, 3, 4, 5])
type(ndarr)

>>

numpy.ndarray
```

Тепер створимо Series з символьними знаками.

```
s3 = pd.Series(ndarr, ['a', 'b', 'c', 'd', 'e'])
print(s3)

>>

a    1
b    2
c    3
d    4
e    5
```


Створення Series зі словника

Ще один спосіб створити структуру Series - це використовувати словник для одночасного задання міток і значень.

```
d = {'a':1, 'b':2, 'c':3}
s4 = pd.Series(d)
print(s4)

>>

a    1
b    2
c    3
```

Ключі зі словника *d* стануть мітками структури *s4*, а значення словника - значеннями в структурі.

Створення Series з використанням константи

Розглянемо ще один спосіб створення структури. Цього разу значення в комірках структури будуть однаковими.

```
a = 7
s5 = pd.Series(a, ['a', 'b', 'c'])
print(s5)

>>

a    7
b    7
c    7
```

У створеній структурі Series є три елементи з однаковим значенням.

Робота з елементами Series

До елементів Series можна звертатися за чисельним індексом. У разі такого підходу робота зі структурою не відрізняється від роботи зі списками.

```
s6 = pd.Series([1, 2, 3, 4, 5], ['a', 'b', 'c', 'd', 'e'])
s6[2]

>>

3
```

Можна використовувати мітку, тоді робота з Series буде схожою на роботу зі словником.

```
s6['d']  
  
>>  
  
4
```

Можна отримувати зрізи.

```
s6[:2]  
  
>>  
  
a    1  
b    2
```

В поле для індексу можна помістити умовний вираз. Це дозволить вивести лише ті елементи, які задовольняють вказаній умові.

```
s6[s6 <= 3]  
  
>>  
  
a    1  
b    2  
c    3
```

З структурами Series можна працювати так само, як з векторами: додавати, множити вектор на число тощо.

```
s7 = pd.Series([10, 20, 30, 40, 50], ['a', 'b', 'c', 'd', 'e'])  
s6 + s7  
  
>>  
  
a    11  
b    22  
c    33  
d    44  
e    55
```

```
s6 * 3  
  
>>  
  
a      3  
b      6  
c      9  
d     12  
e     15
```

Структура даних DataFrame

Якщо Series являє собою одновимірну структуру, яку можна уявити як таблицю з одним рядком, то **DataFrame** – це вже двовимірна структура, повноцінна таблиця з великою кількістю рядків і стовпців.

Конструктор класу DataFrame виглядає так:

- *data* - масив ndarray, словник або інший DataFrame;
- *index* - список міток для записів (назви рядків таблиці);
- *columns* - список міток для полів (назви стовпців таблиці);
- *dtype* - об'єкт *numpy.dtype*, який визначає тип даних;
- *copy* - створює копію масиву даних, якщо параметр дорівнює True, в іншому випадку нічого не робить.

Структуру DataFrame можна створити на базі:

- словника, в якості елементів якого повинні виступати: одновимірні ndarray, списки, інші словники, структури Series;
- структури Series;
- двовимірних ndarray;
- структурованих ndarray;
- інших DataFrame.

Створення DataFrame зі словника

В даному випадку буде використовуватися одновимірний словник, елементами якого будуть списки, структури Series тощо.

Для початку розглянемо Series.

```
d = {"price": pd.Series([1, 2, 3], index=['v1', 'v2', 'v3']),  
     "count": pd.Series([10, 12, 7], index=['v1', 'v2', 'v3'])}  
df1 = pd.DataFrame(d)  
print(df1)
```

```
>>
```

	price	count
v1	1	10
v2	2	12
v3	3	7

Тепер побудуємо аналогічний словник, але на елементах ndarray.

```
d2 = {"price": np.array([1, 2, 3]),  
     "count": np.array([10, 12, 7])}  
df2 = pd.DataFrame(d2, index=['v1', 'v2', 'v3'])  
print(df2)
```

```
>>
```

	price	count
v1	1	10
v2	2	12
v3	3	7

Як видно - результат аналогічний попередньому. Замість ndarray також можна використовувати звичайний список.

```
d3 = {"price": [1, 2, 3],  
     "count": [10, 12, 7]}  
df3 = pd.DataFrame(d3, index=['v1', 'v2', 'v3'])  
print(df3)
```

```
>>
```

	price	count
v1	1	10
v2	2	12
v3	3	7

Створення DataFrame зі списку словників

До цього моменту ми створювали DataFrame зі словника, елементами якого були структури Series, списки і масиви. Тепер ми створимо DataFrame зі списку, елементами якого є словники.

```
d4 = [{"price": 3, "count": 8}, {"price": 4, "count": 11}]
df4 = pd.DataFrame(d4, index=['v1', 'v2'])
print(df4)
```

```
>>
```

	price	count
v1	3	8
v2	4	11

Створення DataFrame з двовимірного масиву

Створити DataFrame можна також і з двовимірного масиву. В наступному прикладі це буде ndarray з бібліотеки NumPy.

```
nda1 = np.array([[1, 2, 3], [10, 20, 30]])
df4 = pd.DataFrame(nda1)
print(df4)
```

```
>>
```

	0	1	2
0	1	2	3
1	10	20	30

Робота з елементами DataFrame

Основні способи роботи з DataFrame:

- `df['col']` - вибір стовпця; повертає Series
- `df.loc['label']` - вибір рядка за міткою; повертає Series
- `df.iloc[loc]` - вибір рядка за індексом; повертає Series
- `df[0:4]` - зріз за рядками; повертає DataFrame
- `df[bool_vec]` - вибір рядків, які відповідають умові; повертає DataFrame

Створимо DataFrame для розгляду цих операцій.

```
d = {"price": np.array([1, 2, 3]),
      "count": np.array([10, 20, 30]),
      "weight": np.array([2.3, 5.7, 8.9])}
df = pd.DataFrame(d, index=['a', 'b', 'c'])
print(df)
```

```
>>
```

	price	count	weight
a	1	10	2.3
b	2	20	5.7
c	3	30	8.9

Вибір стовпця:

```
df['count']
```

```
>>
```

a	10
b	20
c	30

Вибір рядка за міткою:

```
df.loc['a']
```

```
>>
```

price	1.0
count	10.0
weight	2.3

Вибір рядка за індексом:

```
df.iloc[1]
```

```
>>
```

price	2.0
count	20.0
weight	5.7

Зріз за рядками:

```
df[0:2]

>>
   price  count  weight
a      1     10     2.3
b      2     20     5.7
```

Зріз за індексами стовпців:

```
df.iloc[:, 0:2]

>>
   price  count
a      1     10
b      2     20
c      3     30
```

Зріз за іменами стовпців:

```
df.loc[:, 'price':'count']

>>
   price  count
a      1     10
b      2     20
c      3     30
```

Вибір рядків, які відповідають умові:

```
df[df['count'] >= 20]

>>
   price  count  weight
b      2     20     5.7
c      3     30     8.9
```

1.2. Операції з даними в Pandas

Доступ до даних DataFrame з використанням міток

Розглянемо різні варіанти використання міток, які можуть бути як іменами стовпців таблиці, так і іменами рядків. Для початку створимо новий DataFrame.

```
d = {"price": [1, 2, 3], "count": [10, 20, 30], "percent": [24, 51, 71]}
df = pd.DataFrame(d, index=['a', 'b', 'c'])
df
```

```
>>
```

	price	count	percent
a	1	10	24
b	2	20	51
c	3	30	71

Звернення до конкретного стовпця - отримання всіх елементів стовпця *'count'*:

```
df['count']
```

```
>>
```

a	10
b	20
c	30

Звернення з використанням масиву стовпців - отримання елементів стовпців *'count'* і *'price'*:

```
df[['count', 'price']]
```

```
>>
```

	count	price
a	10	1
b	20	2
c	30	3

Звернення за зрізами міток - отримання елементів з мітками від *'a'* до *'b'*:

```
df['a':'b']
```

```
>>
```


	price	count	percent
a	1	10	24
b	2	20	51

Звернення з використанням функції - отримання всіх елементів, у яких значення в стовпці *'count'* більше 15:

```
df[lambda x: x['count'] > 15]
```

```
>>
```

	price	count	percent
b	2	20	51
c	3	30	71

Звернення через логічний вираз. Під час формування логічного виразу необхідно вказувати імена стовпців, за якими буде здійснюватися вибірка (так як і при роботі з функціями).

Отримати всі елементи, у яких *'price'* більше або дорівнює 2:

```
df[df['price'] >= 2]
```

```
>>
```

	price	count	percent
b	2	20	51
c	3	30	71

Використання атрибутів для доступу до даних

Для доступу до даних можна використовувати атрибути структур, в якості яких виступають мітки.

Створимо нову структуру Series:

```
s = pd.Series([10, 20, 30, 40, 50], ['a', 'b', 'c', 'd', 'e'])
```

Для доступу до елементу через атрибут необхідно вказати його через крапку після імені змінної.

Оскільки структура *s* має мітки *'a'*, *'b'*, *'c'*, *'d'*, *'e'*, то для доступу до елементу з міткою *'a'* ми можемо використовувати синтаксис *s.a*.

```
s.a  
  
>>  
  
10
```

```
s.c  
  
>>  
  
30
```

Такий самий підхід можна застосувати для змінної типу DataFrame.

Отримаємо доступ до стовпця *'price'*:

```
d = {"price": [1, 2, 3], "count": [10, 20, 30], "percent": [24, 51, 71]}  
df = pd.DataFrame(d, index=['a', 'b', 'c'])  
  
df.price  
  
>>  
  
a      1  
b      2  
c      3
```

Отримання випадкової вибірки зі структур Pandas

Бібліотека *pandas* надає можливість отримати випадковий набір даних з уже існуючої структури. Такий функціонал має як Series, так і DataFrame. У даних структур є метод *sample()*, який повертає випадкову підвибірку.

Повернемось до структури Series. Для того, щоб вибрати випадковим чином елемент з Series використовується наступний синтаксис:

```
s.sample()  
  
>>  
  
a      10
```

Можна зробити вибірку з декількох елементів, для цього потрібно передати потрібну кількість через параметр *n*.

```
s.sample(n=3)
```

```
>>
```

```
a    10  
c    30  
e    50
```

Також є можливість вказати частку (відсоток) від загального числа об'єктів в структурі, використовуючи параметр *frac*.

```
s.sample(frac=0.3)
```

```
>>
```

```
b    20  
d    40
```

Цікавою особливістю є те, можна передати вектор ваг, довжина якого повинна дорівнювати кількості елементів в структурі. Сума ваг повинна дорівнювати одиниці. Вага, в даному випадку, це ймовірність появи елемента у вибірці.

У нашій тестовій структурі п'ять елементів, сформуємо вектор ваг для неї і зробимо вибірку з трьох елементів.

```
w = [0.1, 0.2, 0.5, 0.1, 0.1]  
s.sample(n = 3, weights=w)
```

```
c    30  
e    50  
b    20
```

Даний функціонал також доступний і для структури DataFrame.

```
d = {"price": [1, 2, 3, 5, 6], "count": [10, 20, 30, 40, 50], "percent":  
[24, 51, 71, 25, 42]}  
df = pd.DataFrame(d)
```

```
df.sample()
```

```
>>
```

	price	count	percent
0	1	10	24

У разі роботи з DataFrame можна вказати вісь.

```
df.sample(axis=1)    # стовпці
```

```
>>
```

	price
0	1
1	2
2	3
3	5
4	6

```
df.sample(n=2, axis=1)
```

```
>>
```

	price	percent
0	1	24
1	2	51
2	3	71
3	5	25
4	6	42

```
df.sample(n=2)    # за замовчуванням axis = 0 (рядки)
```

```
>>
```

	price	count	percent
2	3	30	71
1	2	20	51

Додавання нових елементів до структур

Подібне додаванню нового об'єкта до словників.

```
s = pd.Series([10, 20, 30, 40, 50], ['a', 'b', 'c', 'd', 'e'])
s
```

```
>>
```

a	10
b	20
c	30

```
d    40
e    50
```

```
s['f'] = 60    # додамо новий елемент з міткою 'f'
s
>>
a    10
b    20
c    30
d    40
e    50
f    60
```

Додавання нового елемента в структуру DataFrame аналогічне:

```
d = {"price": [1, 2, 3, 5, 6], "count": [10, 20, 30, 40, 50], "percent":
[24, 51, 71, 25, 42]}
df = pd.DataFrame(d)
df
```

```
>>
   price  count  percent
0      1    10      24
1      2    20      51
2      3    30      71
3      5    40      25
4      6    50      42
```

```
df['value'] = [3, 14, 7, 91, 5]
df
```

```
>>
   price  count  percent  value
0      1    10      24      3
1      2    20      51     14
2      3    30      71      7
3      5    40      25     91
4      6    50      42      5
```

Індексація з використанням логічних виразів

На практиці дуже часто доводиться отримувати певну підвибірку з існуючого набору даних. Наприклад, отримати всі товари, знижка на які більше п'яти відсотків. Або вибрати з бази інформацію про співробітників чоловічої статі старше 30 років. Це дуже схоже на процес фільтрації під час роботи з таблицями або отримання вибірки з бази даних. Схожий функціонал реалізований в *pandas* і ми вже тйого використовували, коли розглядали різні підходи до індексації.

Умовний вираз записується замість індексу в квадратних дужках при зверненні до елементів структури.

Під час роботи з Series можливі наступні варіанти використання:

```
s = pd.Series([10, 20, 30, 40, 50, 10, 10], ['a', 'b', 'c', 'd', 'e', 'f', 'g'])
s[s>30]
```

```
>>
```

```
d    40
e    50
```

```
s[s==10]
```

```
>>
```

```
a    10
f    10
g    10
```

```
s[(s>=30) & (s<50)]
```

```
>>
```

```
c    30
d    40
```

Під час роботи з DataFrame необхідно вказувати стовпець за яким буде проводитися фільтрація (вибірка).

```
d = {"price": [1, 2, 3, 5, 6], "count": [10, 20, 30, 40, 50], "percent": [24, 51, 71, 25, 42], "cat": ["A", "B", "A", "A", "C"]}
df = pd.DataFrame(d)
df
```

```
>>
```

	price	count	percent	cat
0	1	10	24	A
1	2	20	51	B
2	3	30	71	A
3	5	40	25	A
4	6	50	42	C

```
df[df["price"] > 3]
```

```
>>
```

	price	count	percent	cat
3	5	40	25	A
4	6	50	42	C

В якості логічного виразу можна використовувати досить складні конструкції з використанням *map*, *filter*, лямбда-функцій тощо.

```
fn = df["cat"].map(lambda x: x == "A")
df[fn]
```

```
>>
```

	price	count	percent	cat
0	1	10	24	A
2	3	30	71	A
3	5	40	25	A

Використання *isin* для роботи з даними в Pandas

За структурою даних Pandas можна будувати масиви з даними типу *boolean*, за яким можна перевірити наявність або відсутності того чи іншого елемента.

```
s = pd.Series([10, 20, 30, 40, 50, 10, 10], ['a', 'b', 'c', 'd', 'e', 'f', 'g'])
s.isin([10, 20])
```

```
>>
```

```
a    True
b    True
c   False
d   False
e   False
f    True
g    True
```

Робота з DataFrame аналогічна роботі зі структурою Series.

```
df = pd.DataFrame({"price": [1, 2, 3, 5, 6],
                   "count": [10, 20, 30, 40, 50],
                   "percent": [24, 51, 71, 25, 42]})
df.isin([1, 3, 25, 30, 10])
```

```
>>
```

```
   price  count  percent
0   True   True   False
1  False  False   False
2   True   True   False
3  False  False    True
4  False  False   False
```

Виявлення пропущених даних

Дуже часто великі обсяги даних, які готуються для подальшого аналізу, мають пропуски. Для того, щоб можна було використовувати алгоритми машинного навчання, які будують моделі за цими даними, необхідно ці пропуски чимось заповнити.

Створимо структуру DataFrame, яка буде містити пропуски.

Для цього імпортуємо необхідну бібліотеку:

```
from io import StringIO
```

Після цього створимо об'єкт в форматі csv.

```
data = 'price,count,percent\n1,10,\n2,20,51\n3,30,'
df = pd.read_csv(StringIO(data))
```

Отриманий об'єкт *df* - це DataFrame з пропусками.


```
df
>>
   price  count  percent
0      1     10      NaN
1      2     20     51.0
2      3     30      NaN
```

В даному прикладі, у об'єктів з індексами 0 і 2 відсутні дані в полі *percent*. Відсутні дані позначаються як *NaN*. Додамо до існуючої структури ще один запис, у якого буде відсутнє значення в полі *count*.

```
df.loc[3] = {'price':4, 'count':None, 'percent':26.3}
df
>>
   price  count  percent
0      1     10      NaN
1      2     20     51.0
2      3     30      NaN
3      4     None     26.3
```

Для початку використаємо методи з бібліотеки *pandas*, які дозволяють швидко перевірити наявність елементів *NaN* в структурах. Якщо таблиця невелика, то можна використовувати метод *isnull()*.

```
pd.isnull(df)
>>
   price  count  percent
0  False  False    True
1  False  False   False
2  False  False    True
3  False   True   False
```

Таким чином ми отримуємо таблицю того ж розміру, але на місці реальних даних в ній знаходяться логічні змінні. Вони приймають значення *False*, якщо значення поля в об'єкта існує, або *True*, якщо значення в даному полі - це *NaN*. На

додаток до цього можна подивитися детальну інформацію про об'єкт. Для цього можна скористатися методом *info()*.

```
df.info()

>>

<class 'pandas.core.frame.DataFrame'>
Int64Index: 4 entries, 0 to 3
Data columns (total 3 columns):
#   Column      Non-Null Count  Dtype
---  -
0   price       4 non-null      int64
1   count       3 non-null      object
2   percent     2 non-null      float64
dtypes: float64(1), int64(1), object(1)
memory usage: 128.0+ bytes
```

У нашому прикладі видно, що об'єкт *df* має три стовпці (*count*, *percent* і *price*), водночас у стовпці *price* всі об'єкти значимі - НЕ *NaN*, в стовпці *count* – один *NaN* об'єкт, в поле *percent* – два *NaN* об'єкти.

Можна скористатися таким підходом для отримання кількості *NaN* елементів в записах:

```
df.isnull().sum()

>>

price      0
count      1
percent    2
```

Заміна пропущених даних

Пропущені дані об'єктів можна замінити на конкретні числові значення. Для цього можна використовувати метод *fillna()*. Використаємо структуру *df*, створену в попередньому розділі.

```
df

>>
```

	price	count	percent
0	1	10	NaN
1	2	20	51.0
2	3	30	NaN
3	4	None	26.3

```
df.fillna(0)
```

```
>>
```

	price	count	percent
0	1	10	0.0
1	2	20	51.0
2	3	30	0.0
3	4	0	26.3

Цей метод не змінює поточну структуру. Він повертає структуру `DataFrame`, створену на базі існуючої, з заміною *NaN* значень на ті, які передані до методу в якості аргументу. Наприклад, дані можна заповнити середнім значенням по стовпцю.

```
df.fillna(df.mean())
```

```
>>
```

	price	count	percent
0	1	10.0	38.65
1	2	20.0	51.00
2	3	30.0	38.65
3	4	20.0	26.30

В залежності від завдання, використовується той чи інший метод заповнення відсутніх елементів. В якості заміни може бути нульове значення, математичне сподівання, медіана тощо.

Для заміни *NaN* елементів на конкретні значення також можна використовувати інтерполяцію, яка реалізована в методі *interpolate()*. Алгоритм інтерполяції задається через аргументи методу.

Видалення стовпців/об'єктів з пропущеними даними

Досить часто використовують інший підхід до відсутніх даних - це видалення записів (рядків) або колонок (стовпців), в яких зустрічаються пропуски. Для того, щоб видалити всі об'єкти, які містять значення *NaN*, застосовують метод *dropna()* без аргументів.

```
df.dropna()

>>
   price count  percent
1      2     20     51.0
```

Замість записів можна видалити колонки. Для цього потрібно викликати метод *dropna()* з аргументом *axis=1*.

```
df.dropna(axis=1)

>>
   price
0      1
1      2
2      3
3      4
```

Pandas дозволяє задати поріг на кількість не-*NaN* елементів. У наведеному нижче прикладі будуть видалені всі стовпці, в яких кількість не-*NaN* елементів менше трьох.

```
df.dropna(axis = 1, thresh=3)

>>
   price count
0      1     10
1      2     20
2      3     30
3      4    None
```

Об'єднання різних DataFrame

Іноді виникає необхідність об'єднання декількох DataFrame в одну таблицю.

Розглянемо три окремі DataFrame:

```
df1 = pd.DataFrame(  
    {  
        'A': ['A0', 'A1', 'A2', 'A3'],  
        'B': ['B0', 'B1', 'B2', 'B3'],  
        'C': ['C0', 'C1', 'C2', 'C3'],  
        'D': ['D0', 'D1', 'D2', 'D3']  
    },  
    index=[0, 1, 2, 3])  
  
df2 = pd.DataFrame(  
    {  
        'A': ['A4', 'A5', 'A6', 'A7'],  
        'B': ['B4', 'B5', 'B6', 'B7'],  
        'C': ['C4', 'C5', 'C6', 'C7'],  
        'D': ['D4', 'D5', 'D6', 'D7']  
    },  
    index=[4, 5, 6, 7])  
  
df3 = pd.DataFrame(  
    {  
        'A': ['A8', 'A9', 'A10', 'A11'],  
        'B': ['B8', 'B9', 'B10', 'B11'],  
        'C': ['C8', 'C9', 'C10', 'C11'],  
        'D': ['D8', 'D9', 'D10', 'D11']  
    },  
    index=[8, 9, 10, 11])
```

df1

>>

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

```
df2
```

```
>>
```

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

```
df3
```

```
>>
```

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

Тепер об'єднаємо їх в один DataFrame, використовуючи метод `concat()`. Якщо не задавати додаткових атрибутів, об'єднання відбудеться у стовпчик (за стовпцями).

```
pd.concat([df1,df2,df3])
```

```
>>
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

Якщо ж вказати вісь `axis=1`, то об'єднання відбудеться за рядками.

```
pd.concat([df1,df2,df3],axis=1)
```

```
>>
```

	A	B	C	D	A	B	C	D	A	B	C	D
0	A0	B0	C0	D0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	A1	B1	C1	D1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	A2	B2	C2	D2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	A3	B3	C3	D3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	A4	B4	C4	D4	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	A5	B5	C5	D5	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN	A6	B6	C6	D6	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN	A7	B7	C7	D7	NaN	NaN	NaN	NaN
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A8	B8	C8	D8
9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A9	B9	C9	D9
10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A10	B10	C10	D10
11	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A11	B11	C11	D11

Іноді DataFrame можуть мати спільні стовпці. Наприклад, створимо два нових DataFrame з однаковим стовпцем *key*:

```
left = pd.DataFrame({
    'key': ['K0', 'K1', 'K2', 'K3'],
    'A': ['A0', 'A1', 'A2', 'A3'],
    'B': ['B0', 'B1', 'B2', 'B3']
})
```

```
right = pd.DataFrame({
    'key': ['K0', 'K1', 'K2', 'K3'],
    'C': ['C0', 'C1', 'C2', 'C3'],
    'D': ['D0', 'D1', 'D2', 'D3']
})
```

```
left
```

```
>>
```

	key	A	B
0	K0	A0	B0
1	K1	A1	B1
2	K2	A2	B2
3	K3	A3	B3

```
right
```

```
>>
```

	key	C	D
0	K0	C0	D0
1	K1	C1	D1
2	K2	C2	D2
3	K3	C3	D3

Тепер об'єднаємо їх в один DataFrame за стовпцем *key*. Для цього використовується метод *merge()*.

```
pd.merge(left, right, how='inner', on='key')
>>
```

	key	A	B	C	D
0	K0	A0	B0	C0	D0
1	K1	A1	B1	C1	D1
2	K2	A2	B2	C2	D2
3	K3	A3	B3	C3	D3

1.3. Оглядовий аналіз даних з Pandas

Зчитування даних з csv файлу

Розглянемо найпростіший спосіб обробки даних, які зберігаються в форматі *.csv*, а також побудову елементарних графіків.

Для початку роботи імпортуємо необхідні бібліотеки.

```
import pandas as pd
import matplotlib.pyplot as plt

# Красиві графіки
plt.style.use('ggplot')
# Розмір зображень
plt.rcParams['figure.figsize'] = (15, 5)
```

Будемо розглядати дані про велосипедистів Монреалю.

- Завантажити файл з даними: <https://raw.githubusercontent.com/jvns/pandas-cookbook/master/data/bikes.csv>

Цей набір даних описує, скільки людей знаходилося на 9 різних велосипедних доріжках Монреалю кожного дня.

Зчитати файл можна за допомогою методу `read_csv()`. Але в нашому випадку необхідно ще додатково задати атрибути, які зроблять наступне:

- Заміняють роздільник з коми на крапку з комою
- Заміняють кодування на 'latin1' (за замовчуванням ставиться 'utf8')
- Обробляють дати у стовпці 'Date'
- Вкажуть, що спочатку йде день, а потім місяць (формат YYYY-DD-MM)
- Заміняють індекс на значення у стовпці 'Date'

```
fixed_df = pd.read_csv('../bikes.csv', # тут вказуємо шлях до файлу
                        sep=';', encoding='latin1',
                        parse_dates=['Date'], dayfirst=True,
                        index_col='Date')
```

Переглянемо перші три записи з таблиці:

```
fixed_df[:3]
```

```
>>
```

	Berri 1	Brébeuf (données non disponibles)	Côte-Sainte-Catherine	\
Date				
2012-01-01	35	NaN	0	
2012-01-02	83	NaN	1	
2012-01-03	135	NaN	2	

	Maisonneuve 1	Maisonneuve 2	du Parc	Pierre-Dupuy	Rachel1	\
Date						
2012-01-01	38	51	26	10	16	
2012-01-02	68	153	53	6	43	
2012-01-03	104	248	89	3	58	

	St-Urbain (données non disponibles)
Date	
2012-01-01	NaN
2012-01-02	NaN
2012-01-03	NaN

Щоб переглянути стовпці, можна звернутись до них, як до елементів словника. Виведемо перші 10 записів стовпця 'Maisonneuve 1'.

```
fixed_df['Maisonneuve 1'][:10]
```

```
>>
```

```
Date
```

2012-01-01	38
2012-01-02	68
2012-01-03	104
2012-01-04	116
2012-01-05	124
2012-01-06	98
2012-01-07	80
2012-01-08	62
2012-01-09	165
2012-01-10	238

Тепер побудуємо графік, просто використавши метод *plot()* для всіх записів з одного стовпця (рис. 1.1).

```
fixed_df['Maisonneuve 1'].plot()
```

```
>>
```

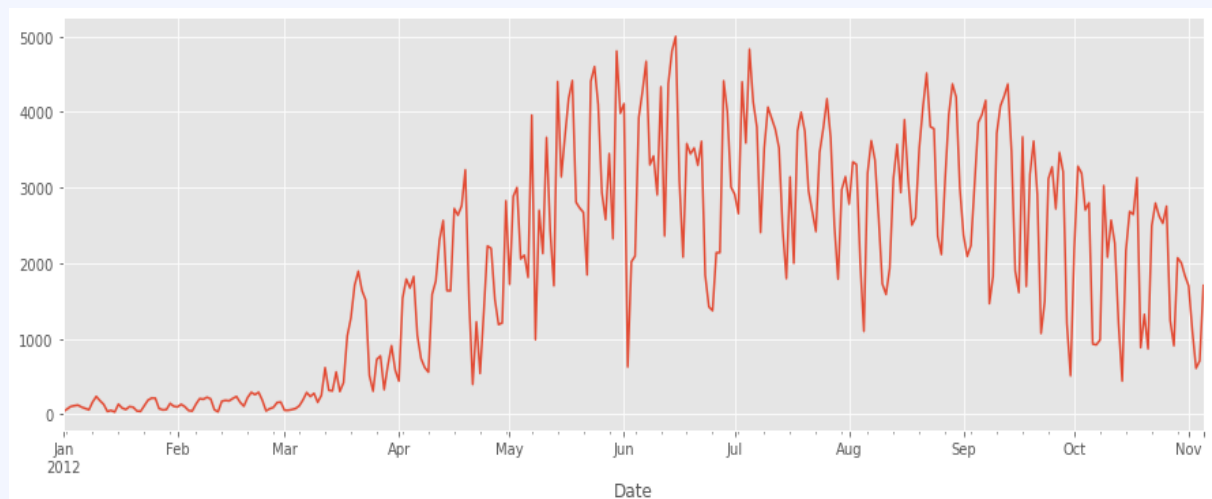


Рис. 1.1. Кількість велосипедистів на велосипедній доріжці Монреаля 'Maisonneuve 1' щодня протягом 2012 року

Найбільш важливим етапом аналізу даних є їх **інтерпретація**. Про що нам говорить такий графік? Очевидно, що не дуже багато людей люблять кататися на велосипеді взимку та на початку весни :)

Також можна побудувати графік одразу для всіх стовпців (рис. 1.2). Для цього бажано трохи збільшити зображення.

```
fixed_df.plot(figsize=(15, 10))
```

```
>>
```

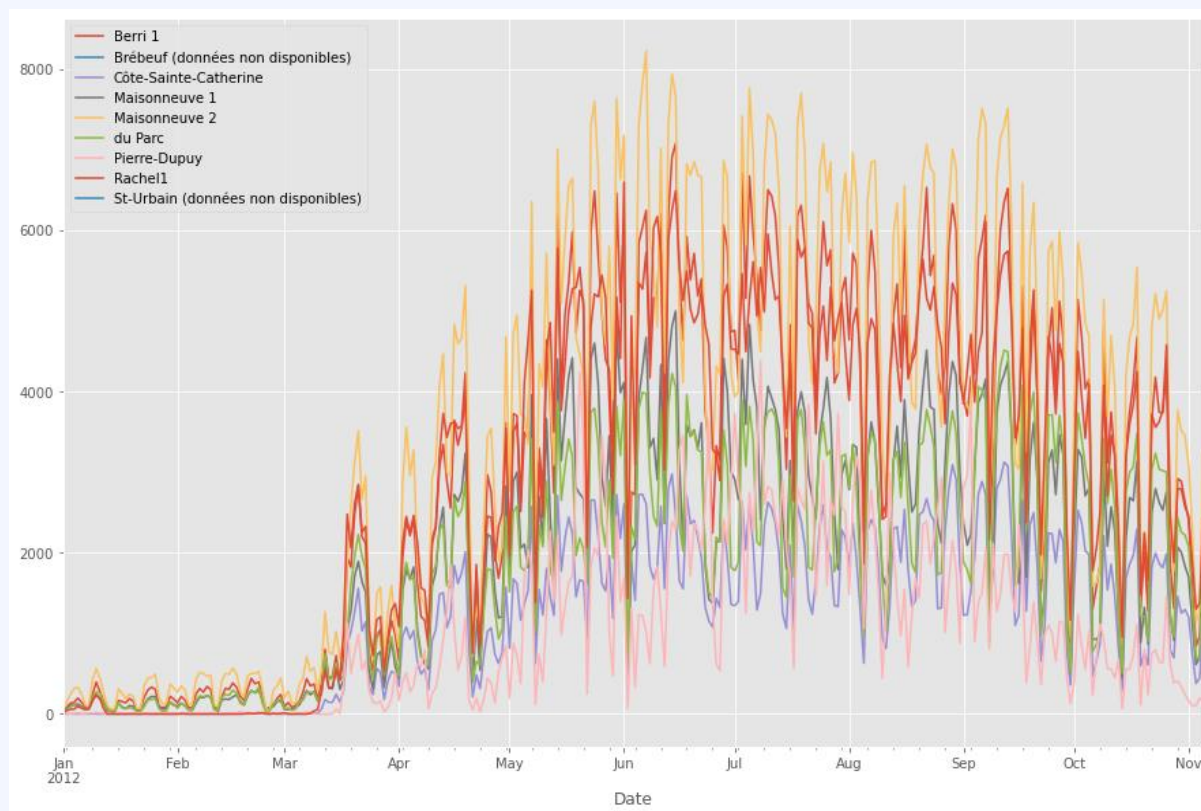


Рис. 1.2. Кількість велосипедистів на всіх 9 велосипедних доріжках Монреаля щодня протягом 2012 року

Що ми тут бачимо? На всіх велосипедних доріжках дані ведуть себе однаково. Якщо це поганий день для велосипедистів, то він поганий всюди.

Наступним кроком з'ясуємо, люди в Монреалі частіше використовують велосипед по буднях чи у вихідні дні?

Як приклад, будемо розглядати доріжку Berri. Створимо для неї окремий DataFrame із використанням методу `copy()`.

```
berri_bikes = fixed_df[['Berri 1']].copy()
berri_bikes[:5]
```

```
>>
```

	Berri 1
Date	
2012-01-01	35
2012-01-02	83
2012-01-03	135
2012-01-04	144
2012-01-05	197

Далі додамо стовпець "день тижня". Зараз в якості індексів рядків використовується дата, це не дуже зручно для нашої мети.

```
berri_bikes.index
```

```
>>
```

```
DatetimeIndex(['2012-01-01', '2012-01-02', '2012-01-03', '2012-01-04',
               '2012-01-05', '2012-01-06', '2012-01-07', '2012-01-08',
               '2012-01-09', '2012-01-10',
               ...,
               '2012-10-27', '2012-10-28', '2012-10-29', '2012-10-30',
               '2012-10-31', '2012-11-01', '2012-11-02', '2012-11-03',
               '2012-11-04', '2012-11-05'],
              dtype='datetime64[ns]', name='Date', length=310, freq=None)
```

Pandas має набір функціоналу для роботи з часовими проміжками. Тому, якщо ми хочемо отримати день місяця для кожного рядка, то можна написати:

```
berri_bikes.index.day
```

```
>>
```

```
Int64Index([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10,
             ...,
             27, 28, 29, 30, 31,  1,  2,  3,  4,  5],
           dtype='int64', name='Date', length=310)
```

Але ми хочемо бачити день тижня, тому доповнюємо:

```
berri_bikes.index.weekday

>>
Int64Index([6, 0, 1, 2, 3, 4, 5, 6, 0, 1,
           ...,
           5, 6, 0, 1, 2, 3, 4, 5, 6, 0],
           dtype='int64', name='Date', length=310)
```

Це дні тижня, 0 - понеділок. Тепер, коли ми знаємо, як отримати день тижня, ми можемо додати його як стовпець в DataFrame.

```
berri_bikes['weekday'] = berri_bikes.index.weekday
berri_bikes[:7]
```

```
>>

      Berri 1  weekday
Date
2012-01-01      35        6
2012-01-02      83        0
2012-01-03     135        1
2012-01-04     144        2
2012-01-05     197        3
2012-01-06     146        4
2012-01-07      98        5
```

Оскільки дні тижня будуть циклічно повторюватись, записи в таблиці необхідно згрупувати. Для цього структура DataFrame має метод *groupby()*, який групує по одному або кількох стовпцях.

У нашому випадку, *berri_bikes.groupby('weekday').sum()* означає "Згрупувати рядки за днями тижня і потім скласти всі значення з однаковим днем тижня".

```
weekday_counts = berri_bikes.groupby('weekday').sum()
weekday_counts

>>

      Berri 1
weekday
0          134298
1          135305
2          152972
```

```
3      160131
4      141771
5      101578
6       99310
```

Тепер перейменуємо 0, 1, 2, 3, 4, 5, 6, щоб розуміти, що вони означають.

```
weekday_counts.index = ['Понеділок', 'Вівторок', 'Середа', 'Четвер',
"П'ятниця", 'Субота', 'Неділя']
weekday_counts
```

```

          Berri 1
Понеділок  134298
Вівторок   135305
Середа     152972
Четвер     160131
П'ятниця   141771
Субота     101578
Неділя     99310
```

Виведемо графік (рис. 1.3):

```
weekday_counts.plot(kind='bar')
```

```
>>
```

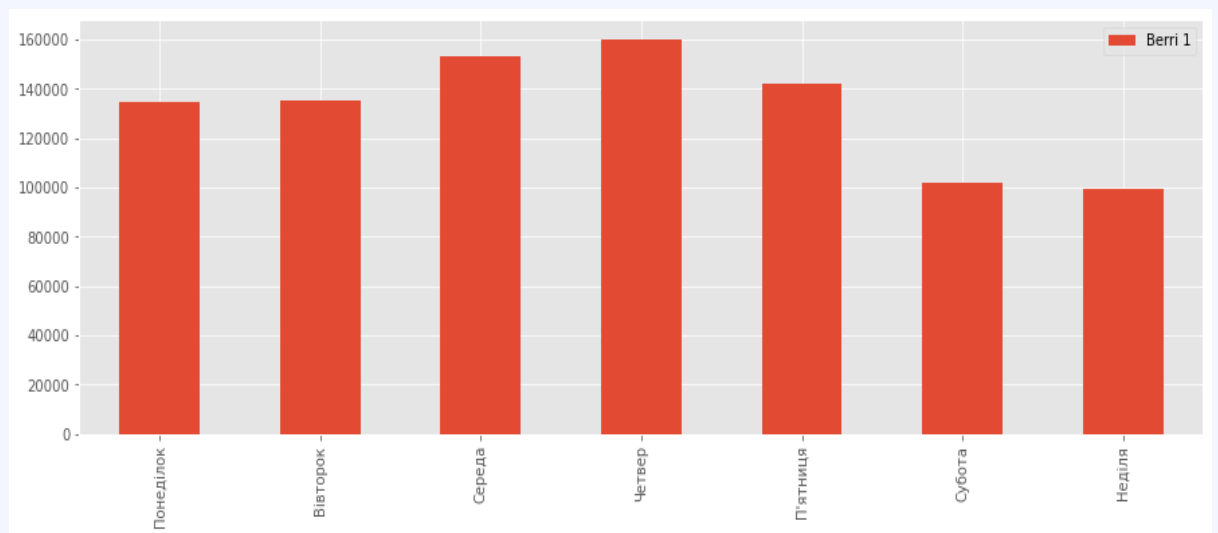


Рис. 1.3. Графік розподілу велосипедистів на велосипедній доріжці Монреаля
‘Berri 1’ в залежності від дня тижня

Як бачимо, в Монреалі частіше їздять на велосипедах по будням.

Аналіз великого обсягу даних

Завантажимо новий набір даних, щоб навчитись працювати з великими обсягами. Це дані про запити (скарги) жителів США до соціальної служби "Сервіс 311" (аналог київської 1551).

- Завантажити набір даних: <https://raw.githubusercontent.com/jvns/pandas-cookbook/master/data/311-service-requests.csv>

```
complaints = pd.read_csv('C:/Drive/IIC/Datasets/311-service-requests.csv')
>>

C:\Py\Anaconda3\envs\PythonGPU\lib\site-packages\IPython\core\interactiv
eshell.py:3072: DtypeWarning: Columns (8) have mixed types.Specify dtype
option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

Як ми бачимо, виникло попередження. Це означає, що pandas зіткнувся з проблемою читання даних. У нашому випадку це майже точно означає, що дані мають стовпці, в яких деякі записи є рядками, а деякі являють собою цілі числа. Поки що будемо це ігнорувати, але зазвичай в таких ситуаціях потрібно більш детально розбиратись.

Переглянемо інформацію про завантажений набір даних:

```
complaints.info()
>>

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 111069 entries, 0 to 111068
Data columns (total 52 columns):
 #   Column                Non-Null Count  Dtype
---  -
 0   Unique Key            111069 non-null int64
 1   Created Date           111069 non-null object
 2   Closed Date            60270 non-null object
 3   Agency                 111069 non-null object
 4   Agency Name            111069 non-null object
 5   Complaint Type         111069 non-null object
 6   Descriptor             110613 non-null object
```

7	Location Type	79022 non-null	object
8	Incident Zip	98807 non-null	object
9	Incident Address	84441 non-null	object
10	Street Name	84432 non-null	object
11	Cross Street 1	84728 non-null	object
12	Cross Street 2	84005 non-null	object
13	Intersection Street 1	19364 non-null	object
14	Intersection Street 2	19366 non-null	object
15	Address Type	102247 non-null	object
16	City	98854 non-null	object
17	Landmark	95 non-null	object
18	Facility Type	19104 non-null	object
19	Status	111069 non-null	object
20	Due Date	39239 non-null	object
21	Resolution Action Updated Date	96507 non-null	object
22	Community Board	111069 non-null	object
23	Borough	111069 non-null	object
24	X Coordinate (State Plane)	98143 non-null	float64
25	Y Coordinate (State Plane)	98143 non-null	float64
26	Park Facility Name	111069 non-null	object
27	Park Borough	111069 non-null	object
28	School Name	111069 non-null	object
29	School Number	111048 non-null	object
30	School Region	110524 non-null	object
31	School Code	110524 non-null	object
32	School Phone Number	111069 non-null	object
33	School Address	111069 non-null	object
34	School City	111069 non-null	object
35	School State	111069 non-null	object
36	School Zip	111069 non-null	object
37	School Not Found	38984 non-null	object
38	School or Citywide Complaint	0 non-null	float64
39	Vehicle Type	99 non-null	object
40	Taxi Company Borough	117 non-null	object
41	Taxi Pick Up Location	1059 non-null	object
42	Bridge Highway Name	185 non-null	object
43	Bridge Highway Direction	185 non-null	object
44	Road Ramp	180 non-null	object
45	Bridge Highway Segment	219 non-null	object
46	Garage Lot Name	49 non-null	object
47	Ferry Direction	24 non-null	object
48	Ferry Terminal Name	70 non-null	object
49	Latitude	98143 non-null	float64
50	Longitude	98143 non-null	float64
51	Location	98143 non-null	object

dtypes: float64(5), int64(1), object(46)
memory usage: 44.1+ MB

Як ми бачимо, дані містять 52 колонки та 111069 записів. Якщо ми хочемо вивести цей DataFrame, то будуть показані тільки декілька перших рядків.

Виведемо перші 5 рядків таблиці:

```
complaints[:5]
```

```
>>
```

	Unique Key	Created Date	Closed Date	Agency	\
0	26589651	10/31/2013 02:08:41 AM	NaN	NYPD	
1	26593698	10/31/2013 02:01:04 AM	NaN	NYPD	
2	26594139	10/31/2013 02:00:24 AM	10/31/2013 02:40:32 AM	NYPD	
3	26595721	10/31/2013 01:56:23 AM	10/31/2013 02:21:48 AM	NYPD	
4	26590930	10/31/2013 01:53:44 AM	NaN	DOHMH	

	Agency Name	Complaint Type	\
0	New York City Police Department	Noise - Street/Sidewalk	
1	New York City Police Department	Illegal Parking	
2	New York City Police Department	Noise - Commercial	
3	New York City Police Department	Noise - Vehicle	
4	Department of Health and Mental Hygiene	Rodent	

	Descriptor	Location Type	Incident Zip	\
0	Loud Talking	Street/Sidewalk	11432	
1	Commercial Overnight Parking	Street/Sidewalk	11378	
2	Loud Music/Party	Club/Bar/Restaurant	10032	
3	Car/Truck Horn	Street/Sidewalk	10023	
4	Condition Attracting Rodents	Vacant Lot	10027	

	Incident Address	... Bridge Highway Name	Bridge Highway Direction	\
0	90-03 169 STREET	...	NaN	NaN
1	58 AVENUE	...	NaN	NaN
2	4060 BROADWAY	...	NaN	NaN
3	WEST 72 STREET	...	NaN	NaN
4	WEST 124 STREET	...	NaN	NaN

	Road Ramp	Bridge Highway Segment	Garage Lot Name	Ferry Direction	\
0	NaN	NaN	NaN	NaN	
1	NaN	NaN	NaN	NaN	
2	NaN	NaN	NaN	NaN	
3	NaN	NaN	NaN	NaN	
4	NaN	NaN	NaN	NaN	

	Ferry Terminal Name	Latitude	Longitude	\
0	NaN	40.708275	-73.791604	
1	NaN	40.721041	-73.909453	
2	NaN	40.843330	-73.939144	

```

3           NaN  40.778009 -73.980213
4           NaN  40.807691 -73.947387

                                Location
0  (40.70827532593202, -73.79160395779721)
1  (40.721040535628305, -73.90945306791765)
2  (40.84332975466513, -73.93914371913482)
3  (40.7780087446372, -73.98021349023975)
4  (40.80769092704951, -73.94738703491433)

[5 rows x 52 columns]

```

Переглянемо записи лише по одному із стовпців (наприклад, 'Complaint Type' - вид скарги):

```

complaints['Complaint Type'][:5]

>>
0    Noise - Street/Sidewalk
1          Illegal Parking
2    Noise - Commercial
3    Noise - Vehicle
4              Rodent

```

До речі, порядок вказування індексів не важливий:

```

complaints[:5]['Complaint Type']

>>
0    Noise - Street/Sidewalk
1          Illegal Parking
2    Noise - Commercial
3    Noise - Vehicle
4              Rodent

```

Тепер переглянемо лише тип скарги та район, відкинувши всі інші колонки.

```

complaints[['Complaint Type', 'Borough']]

>>

              Complaint Type    Borough
0    Noise - Street/Sidewalk    QUEENS

```

```

1           Illegal Parking      QUEENS
2      Noise - Commercial  MANHATTAN
3           Noise - Vehicle  MANHATTAN
4                Rodent  MANHATTAN
...
111064  Maintenance or Facility  BROOKLYN
111065           Illegal Parking      QUEENS
111066  Noise - Street/Sidewalk  MANHATTAN
111067      Noise - Commercial  BROOKLYN
111068      Blocked Driveway  BROOKLYN

[111069 rows x 2 columns]

```

Як бачимо, виводяться тільки перші і останні 5 записів.

Виведемо натомість перші 10.

```

complaints[['Complaint Type', 'Borough']][:10]

>>

```

	Complaint Type	Borough
0	Noise - Street/Sidewalk	QUEENS
1	Illegal Parking	QUEENS
2	Noise - Commercial	MANHATTAN
3	Noise - Vehicle	MANHATTAN
4	Rodent	MANHATTAN
5	Noise - Commercial	QUEENS
6	Blocked Driveway	QUEENS
7	Noise - Commercial	QUEENS
8	Noise - Commercial	MANHATTAN
9	Noise - Commercial	BROOKLYN

Тепер визначимо, який тип скарг зустрічається найчастіше. Для цього у *pandas* є вбудований метод `value_counts()`.

```

complaints['Complaint Type'].value_counts()

>>

```

HEATING	14200
GENERAL CONSTRUCTION	7471
Street Light Condition	7117
DOF Literature Request	5797
PLUMBING	5373
...	

Stalled Sites	1
Ferry Permit	1
Municipal Parking Facility	1
Trans Fat	1
Snow	1

Виведемо 10 найбільш частих типів скарг:

```
complaint_counts = complaints['Complaint Type'].value_counts()
complaint_counts[:10]
```

```
>>
```

HEATING	14200
GENERAL CONSTRUCTION	7471
Street Light Condition	7117
DOF Literature Request	5797
PLUMBING	5373
PAINT - PLASTER	5149
Blocked Driveway	4590
NONCONST	3998
Street Condition	3473
Illegal Parking	3343

І для зручності побудуємо стовпчикову діаграму (рис. 1.4).

```
complaint_counts[:10].plot(kind='bar')
```

```
>>
```

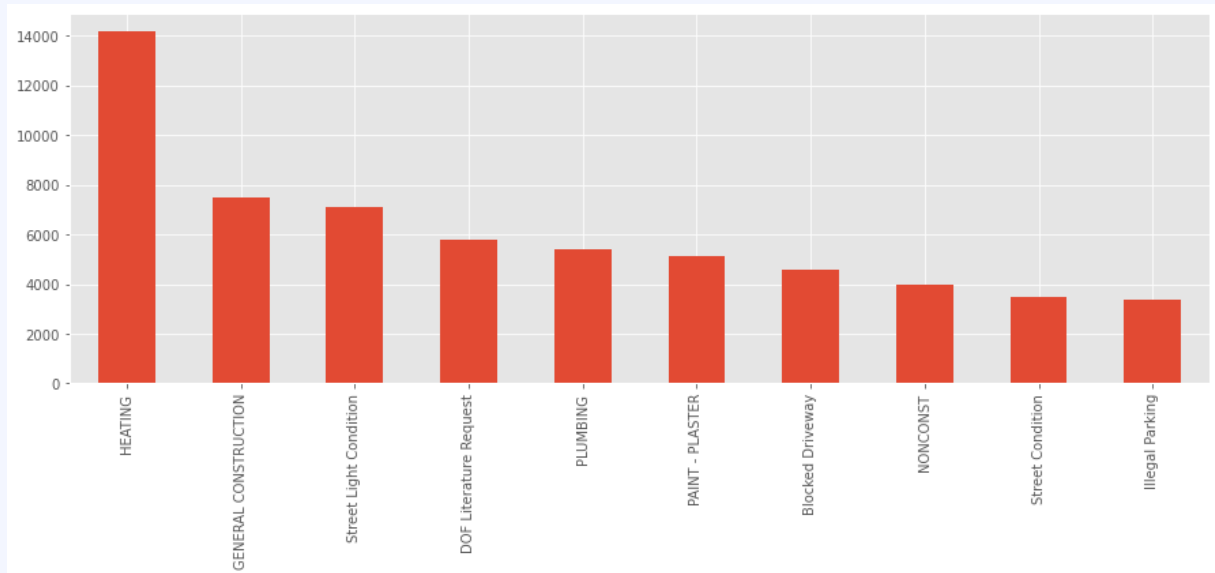


Рис. 1.4. Стовпчикова діаграма кількості скарг жителів США в залежності від типу скарги

Як бачимо, найчастіше жителі США скаржаться на опалення :)

Об'єднання даних та операції із записами

Повернемося до велосипедистів. З'ясуємо, чи багато людей люблять кататись на велосипеді у дощ. Для цього скористаємось даними з архіву погоди в Канаді.

Візьмемо дані за 2012 рік:

http://climate.weather.gc.ca/climate_data/bulk_data_e.html?format=csv&stationID=5415&Year={year}&Month={month}&timeframe=1&submit=Download+Data

Дані не обов'язково завантажувати на ПК. Можна автоматично відформатувати рядок стандартними методами Python, вказавши атрибути *month=3*, *year=2012* у відповідному методі *format()*.

Ми можемо використовувати *read_csv()* і просто вказувати URL як ім'я файлу. Також вкажемо, що будемо обробляти дати, і встановимо 'Date / Time (LST)' в якості індексів.

```
url_template = "http://climate.weather.gc.ca/climate_data/bulk_data_e.html?format=csv&stationID=5415&Year={year}&Month={month}&timeframe=1&submit=Download+Data"
url = url_template.format(month=3, year=2012)
weather_mar2012 = pd.read_csv(url, index_col='Date/Time (LST)', parse_dates=True, encoding='latin1')
```

```
weather_mar2012
```

```
>>
```

	Longitude (x)	Latitude (y)	\
Date/Time (LST)			
2012-03-01 00:00:00	-73.75	45.47	
2012-03-01 01:00:00	-73.75	45.47	
2012-03-01 02:00:00	-73.75	45.47	
...	...	...	
2012-03-31 21:00:00	-73.75	45.47	
2012-03-31 22:00:00	-73.75	45.47	
2012-03-31 23:00:00	-73.75	45.47	

	Station Name	Climate ID	Year	\
Date/Time (LST)				
2012-03-01 00:00:00	MONTREAL/PIERRE ELLIOTT TRUDEAU INTL A	7025250	2012	
2012-03-01 01:00:00	MONTREAL/PIERRE ELLIOTT TRUDEAU INTL A	7025250	2012	
2012-03-01 02:00:00	MONTREAL/PIERRE ELLIOTT TRUDEAU INTL A	7025250	2012	
...	...	...	...	

2012-03-31 21:00:00	MONTREAL/PIERRE ELLIOTT TRUDEAU INTL A	7025250	2012
2012-03-31 22:00:00	MONTREAL/PIERRE ELLIOTT TRUDEAU INTL A	7025250	2012
2012-03-31 23:00:00	MONTREAL/PIERRE ELLIOTT TRUDEAU INTL A	7025250	2012

Date/Time (LST)	Month	Day	Time (LST)	Temp (°C)	Temp Flag	...	\
2012-03-01 00:00:00	3	1	00:00	-5.5	NaN	...	
2012-03-01 01:00:00	3	1	01:00	-5.7	NaN	...	
2012-03-01 02:00:00	3	1	02:00	-5.4	NaN	...	
...	...	...	...	...	...	...	
2012-03-31 21:00:00	3	31	21:00	2.6	NaN	...	
2012-03-31 22:00:00	3	31	22:00	2.7	NaN	...	
2012-03-31 23:00:00	3	31	23:00	1.5	NaN	...	

Date/Time (LST)	Wind Spd Flag	Visibility (km)	Visibility Flag	\
2012-03-01 00:00:00	NaN	4.0	NaN	
2012-03-01 01:00:00	NaN	2.4	NaN	
2012-03-01 02:00:00	NaN	4.8	NaN	
...	...	...	...	
2012-03-31 21:00:00	NaN	25.0	NaN	
2012-03-31 22:00:00	NaN	25.0	NaN	
2012-03-31 23:00:00	NaN	25.0	NaN	

Date/Time (LST)	Stn Press (kPa)	Stn Press Flag	Hmdx	Hmdx Flag	\
2012-03-01 00:00:00	100.97	NaN	NaN	NaN	
2012-03-01 01:00:00	100.87	NaN	NaN	NaN	
2012-03-01 02:00:00	100.80	NaN	NaN	NaN	
...	...	...	...	...	
2012-03-31 21:00:00	100.86	NaN	NaN	NaN	
2012-03-31 22:00:00	100.82	NaN	NaN	NaN	
2012-03-31 23:00:00	100.79	NaN	NaN	NaN	

Date/Time (LST)	Wind Chill	Wind Chill Flag	Weather
2012-03-01 00:00:00	-13.0	NaN	Snow
2012-03-01 01:00:00	-13.0	NaN	Snow
2012-03-01 02:00:00	-13.0	NaN	Snow
...	...	...	...
2012-03-31 21:00:00	NaN	NaN	Clear
2012-03-31 22:00:00	NaN	NaN	Clear
2012-03-31 23:00:00	NaN	NaN	Clear

[744 rows x 29 columns]

Побудуємо графік температури (рис. 1.5).

```
weather_mar2012["Temp (°C)"].plot(figsize=(15, 5))
```

```
>>
```

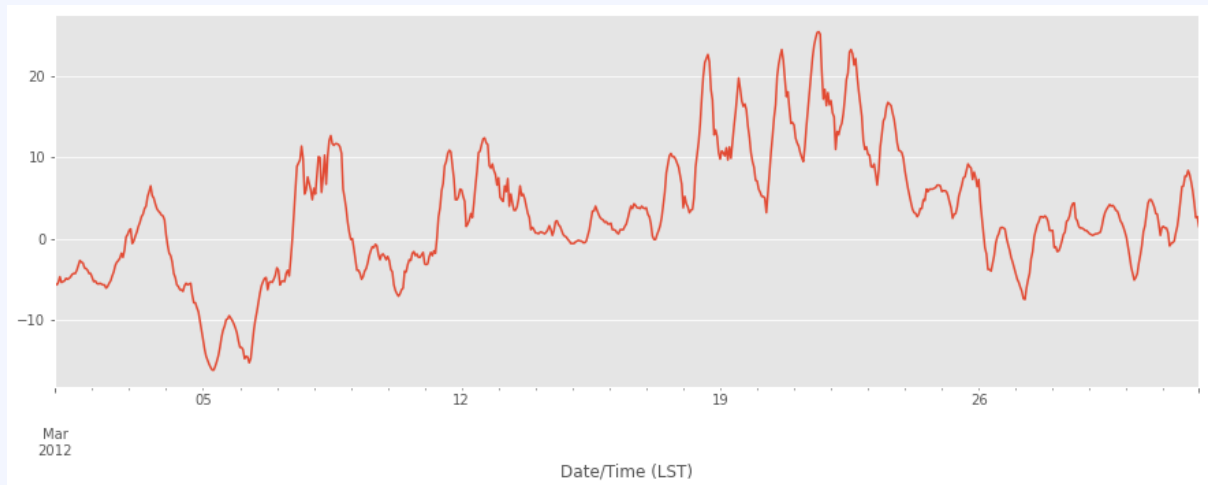


Рис. 1.5. Розподіл температури в березні 2012 року

Назви стовпців можна змінити для зручності (наприклад, "Temp (°C)" – не дуже зручно).

```
weather_mar2012.columns = [u'Longitude', u'Latitude', u'Station',  
u'Climate ID', u'Year', u'Month', u'Day', u'Time', u'Temp (C)',  
u'Temp Flag', u'Dew Point Temp (C)', u'Dew Point Temp Flag',  
u'Rel Hum (%)', u'Rel Hum Flag', u'Precip. Amount (mm)',  
u'Precip. Amount Flag', u'Wind Dir (10s deg)', u'Wind Dir Flag',  
u'Wind Spd (km/h)', u'Wind Spd Flag', u'Visibility (km)',  
u'Visibility Flag', u'Stn Press (kPa)', u'Stn Press Flag', u'Hmdx',  
u'Hmdx Flag', u'Wind Chill', u'Wind Chill Flag', u'Weather']
```

Є багато стовпців, які майже повністю порожні. Видалимо їх за допомогою методу *dropna()*.

Аргумент *axis = 1* у *dropna()* означає "видалити стовпці, а не рядки", і *how = 'any'* означає "видалити стовпець, якщо хоча б одне значення порожнє".

```
weather_mar2012 = weather_mar2012.dropna(axis=1, how='any')
```

```
weather_mar2012[:5]
```

```
>>
```

Date/Time (LST)	Longitude	Latitude	\
2012-03-01 00:00:00	-73.75	45.47	
2012-03-01 01:00:00	-73.75	45.47	
2012-03-01 02:00:00	-73.75	45.47	
2012-03-01 03:00:00	-73.75	45.47	
2012-03-01 04:00:00	-73.75	45.47	

Date/Time (LST)	Station	Climate ID	Year	\
2012-03-01 00:00:00	MONTREAL/PIERRE ELLIOTT TRUDEAU INTL A	7025250	2012	
2012-03-01 01:00:00	MONTREAL/PIERRE ELLIOTT TRUDEAU INTL A	7025250	2012	
2012-03-01 02:00:00	MONTREAL/PIERRE ELLIOTT TRUDEAU INTL A	7025250	2012	
2012-03-01 03:00:00	MONTREAL/PIERRE ELLIOTT TRUDEAU INTL A	7025250	2012	
2012-03-01 04:00:00	MONTREAL/PIERRE ELLIOTT TRUDEAU INTL A	7025250	2012	

Date/Time (LST)	Month	Day	Time	Temp (C)	Dew Point Temp (C)	\
2012-03-01 00:00:00	3	1	00:00	-5.5	-9.7	
2012-03-01 01:00:00	3	1	01:00	-5.7	-8.7	
2012-03-01 02:00:00	3	1	02:00	-5.4	-8.3	
2012-03-01 03:00:00	3	1	03:00	-4.7	-7.7	
2012-03-01 04:00:00	3	1	04:00	-5.4	-7.8	

Date/Time (LST)	Rel Hum (%)	Wind Spd (km/h)	Visibility (km)	\
2012-03-01 00:00:00	72	24	4.0	
2012-03-01 01:00:00	79	26	2.4	
2012-03-01 02:00:00	80	28	4.8	
2012-03-01 03:00:00	79	28	4.0	
2012-03-01 04:00:00	83	35	1.6	

Date/Time (LST)	Stn Press (kPa)	Weather
2012-03-01 00:00:00	100.97	Snow
2012-03-01 01:00:00	100.87	Snow
2012-03-01 02:00:00	100.80	Snow
2012-03-01 03:00:00	100.69	Snow
2012-03-01 04:00:00	100.62	Snow

Стовпці Longitude / Latitude / Station / Climate ID / Year / Month / Day / Time для нашої задачі зайві. Видалимо їх за допомогою методу *drop()*.


```
weather_mar2012 = weather_mar2012.drop(weather_mar2012.iloc[:, 0:8], axis=1)
weather_mar2012[:3]
```

```
>>
```

Date/Time (LST)	Temp (C)	Dew Point Temp (C)	Rel Hum (%)	\
2012-03-01 00:00:00	-5.5	-9.7	72	
2012-03-01 01:00:00	-5.7	-8.7	79	
2012-03-01 02:00:00	-5.4	-8.3	80	

Date/Time (LST)	Wind Spd (km/h)	Visibility (km)	Stn Press (kPa)	Weather
2012-03-01 00:00:00	24	4.0	100.97	Snow
2012-03-01 01:00:00	26	2.4	100.87	Snow
2012-03-01 02:00:00	28	4.8	100.80	Snow

Побудуємо графік температури (рис. 1.6). Для цього згрупуємо дані за часом доби, який "витягнемо" з індексів (так само, як ми робили для велосипедистів за днями тижня). Тільки рахувати будемо не суму, а медіану.

```
import numpy as np
```

```
temperatures = weather_mar2012[[u'Temp (C)']].copy()
temperatures.loc[:, 'Hour'] = weather_mar2012.index.hour
temperatures.groupby('Hour').aggregate(np.median).plot()
```

```
>>
```

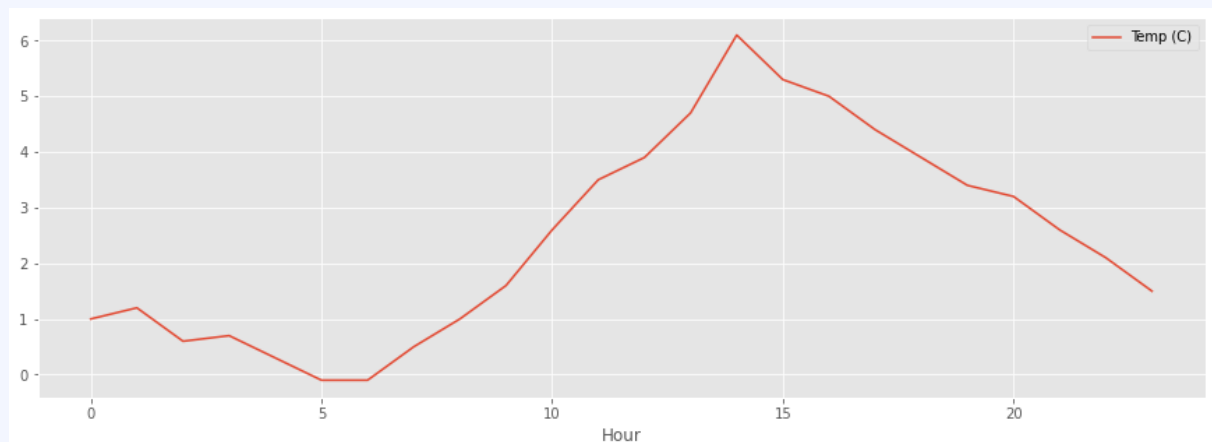


Рис. 1.6. Розподіл температури в залежності від часу доби

Як бачимо, найвища температура за медіаною припадає на 14:00.

Дані за місяць ми навчилися обробляти. А як отримати дані за рік?

Для цього можемо створити функцію - парсер:

```
def download_weather_month(year, month):
    url = url_template.format(year=year, month=month)
    weather_data = pd.read_csv(url, index_col='Date/Time (LST)',
                                parse_dates=True, encoding='latin1')
    weather_data.columns = [u'Longitude', u'Latitude', u'Station',
u'Climate ID', u'Year', u'Month', u'Day', u'Time', u'Temp (C)',
u'Temp Flag', u'Dew Point Temp (C)', u'Dew Point Temp Flag',
u'Rel Hum (%)', u'Rel Hum Flag', u'Precip. Amount (mm)',
u'Precip. Amount Flag', u'Wind Dir (10s deg)', u'Wind Dir Flag',
u'Wind Spd (km/h)', u'Wind Spd Flag', u'Visibility (km)',
u'Visibility Flag', u'Stn Press (kPa)', u'Stn Press Flag', u'Hmdx',
u'Hmdx Flag', u'Wind Chill', u'Wind Chill Flag', u'Weather']

    weather_data = weather_data.dropna(axis=1)
    weather_data = weather_data.drop(weather_data.iloc[:, 0:8], axis=1)
    return weather_data
```

Перевіримо:

```
download_weather_month(2012, 1)[:5]
```

```
>>
```

	Temp (C)	Dew Point Temp (C)	Rel Hum (%)	\
Date/Time (LST)				
2012-01-01 00:00:00	-1.8	-3.9	86	
2012-01-01 01:00:00	-1.8	-3.7	87	
2012-01-01 02:00:00	-1.8	-3.4	89	
2012-01-01 03:00:00	-1.5	-3.2	88	
2012-01-01 04:00:00	-1.5	-3.3	88	

	Wind Spd (km/h)	Visibility (km)	Stn Press (kPa)	\
Date/Time (LST)				
2012-01-01 00:00:00	4	8.0	101.24	
2012-01-01 01:00:00	4	8.0	101.24	
2012-01-01 02:00:00	7	4.0	101.26	
2012-01-01 03:00:00	6	4.0	101.27	
2012-01-01 04:00:00	7	4.8	101.23	

	Weather
Date/Time (LST)	
2012-01-01 00:00:00	Fog
2012-01-01 01:00:00	Fog
2012-01-01 02:00:00	Freezing Drizzle,Fog

```
2012-01-01 03:00:00 Freezing Drizzle,Fog
2012-01-01 04:00:00 Fog
```

Згенеруємо список з DataFrame для кожного місяця:

```
data_by_month = [download_weather_month(2012, i) for i in range(1, 13)]
```

І об'єднаємо все в один DataFrame.

```
weather_2012 = pd.concat(data_by_month)
weather_2012
```

```
>>
```

Date/Time (LST)	Temp (C)	Dew Point Temp (C)	Rel Hum (%)	\
2012-01-01 00:00:00	-1.8	-3.9	86	
2012-01-01 01:00:00	-1.8	-3.7	87	
2012-01-01 02:00:00	-1.8	-3.4	89	
2012-01-01 03:00:00	-1.5	-3.2	88	
2012-01-01 04:00:00	-1.5	-3.3	88	
...	...	...	...	
2012-12-31 19:00:00	0.1	-2.7	81	
2012-12-31 20:00:00	0.2	-2.4	83	
2012-12-31 21:00:00	-0.5	-1.5	93	
2012-12-31 22:00:00	-0.2	-1.8	89	
2012-12-31 23:00:00	0.0	-2.1	86	

Date/Time (LST)	Wind Spd (km/h)	Visibility (km)	Stn Press (kPa)	\
2012-01-01 00:00:00	4	8.0	101.24	
2012-01-01 01:00:00	4	8.0	101.24	
2012-01-01 02:00:00	7	4.0	101.26	
2012-01-01 03:00:00	6	4.0	101.27	
2012-01-01 04:00:00	7	4.8	101.23	
...	...	...	...	
2012-12-31 19:00:00	30	9.7	100.13	
2012-12-31 20:00:00	24	9.7	100.03	
2012-12-31 21:00:00	28	4.8	99.95	
2012-12-31 22:00:00	28	9.7	99.91	
2012-12-31 23:00:00	30	11.3	99.89	

Date/Time (LST)	Weather
2012-01-01 00:00:00	Fog
2012-01-01 01:00:00	Fog
2012-01-01 02:00:00	Freezing Drizzle,Fog

```

2012-01-01 03:00:00 Freezing Drizzle,Fog
2012-01-01 04:00:00 Fog
...
2012-12-31 19:00:00 Snow
2012-12-31 20:00:00 Snow
2012-12-31 21:00:00 Snow
2012-12-31 22:00:00 Snow
2012-12-31 23:00:00 Snow

```

```
[8784 rows x 7 columns]
```

Ми отримали дані за весь рік.

Для того, щоб кожного разу не парсити дані, збережемо їх у форматі csv.

```
weather_2012.to_csv('../weather_2012.csv')
```

І наостанок, знайдемо найбільш сніжний місяць.

Для початку імпортуємо щойно створений набір даних, щоб перевірити його правильність.

```

weather_2012 = pd.read_csv('../weather_2012.csv', parse_dates=True,
                             index_col='Date/Time (LST)')

```

```
weather_2012[:5]
```

```
>>
```

Date/Time (LST)	Temp (C)	Dew Point Temp (C)	Rel Hum (%)	\
2012-01-01 00:00:00	-1.8	-3.9	86	
2012-01-01 01:00:00	-1.8	-3.7	87	
2012-01-01 02:00:00	-1.8	-3.4	89	
2012-01-01 03:00:00	-1.5	-3.2	88	
2012-01-01 04:00:00	-1.5	-3.3	88	

Date/Time (LST)	Wind Spd (km/h)	Visibility (km)	Stn Press (kPa)	\
2012-01-01 00:00:00	4	8.0	101.24	
2012-01-01 01:00:00	4	8.0	101.24	
2012-01-01 02:00:00	7	4.0	101.26	
2012-01-01 03:00:00	6	4.0	101.27	
2012-01-01 04:00:00	7	4.8	101.23	

Date/Time (LST)	Weather
2012-01-01 00:00:00	Fog

```
2012-01-01 01:00:00      Fog
2012-01-01 02:00:00  Freezing Drizzle,Fog
2012-01-01 03:00:00  Freezing Drizzle,Fog
2012-01-01 04:00:00      Fog
```

Ми бачимо, що стовпець 'Weather' містить текстовий опис погоди за годину. Припустимо, що сніжна погода в описі містить "Snow".

Pandas дозволяє здійснювати операції над рядковими даними, дозволяючи ефективно обробляти стовпці.

```
weather_description = weather_2012['Weather']
is_snowing = weather_description.str.contains('Snow')
```

Цей код повертає булевий вектор.

```
is_snowing[:5]

>>

Date/Time (LST)
2012-01-01 00:00:00    False
2012-01-01 01:00:00    False
2012-01-01 02:00:00    False
2012-01-01 03:00:00    False
2012-01-01 04:00:00    False
```

Для аналізу вектору зручніше побудувати графік (рис. 1.7), перетворивши дані на float.

```
is_snowing = is_snowing.astype(int)
is_snowing.plot()

>>
```

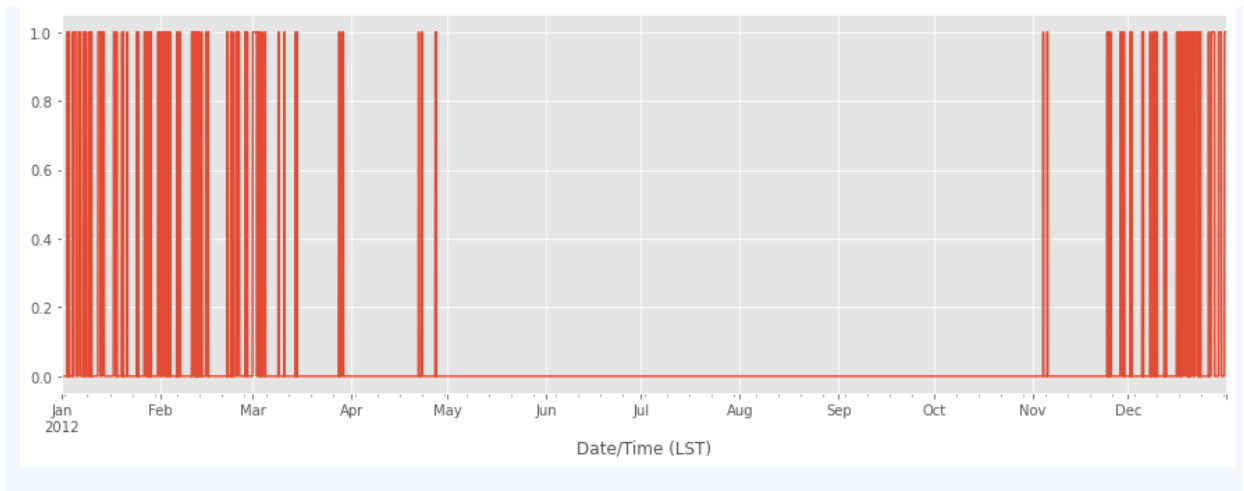


Рис. 1.7. Графік розподілу снігу за місяцями протягом 2012 року

Можна побудувати графік медіанної температури за кожен місяць (рис. 1.8). Для цього потрібно використати метод *resample()*. Параметр 'M' вказує, що ми беремо дані за кожен місяць.

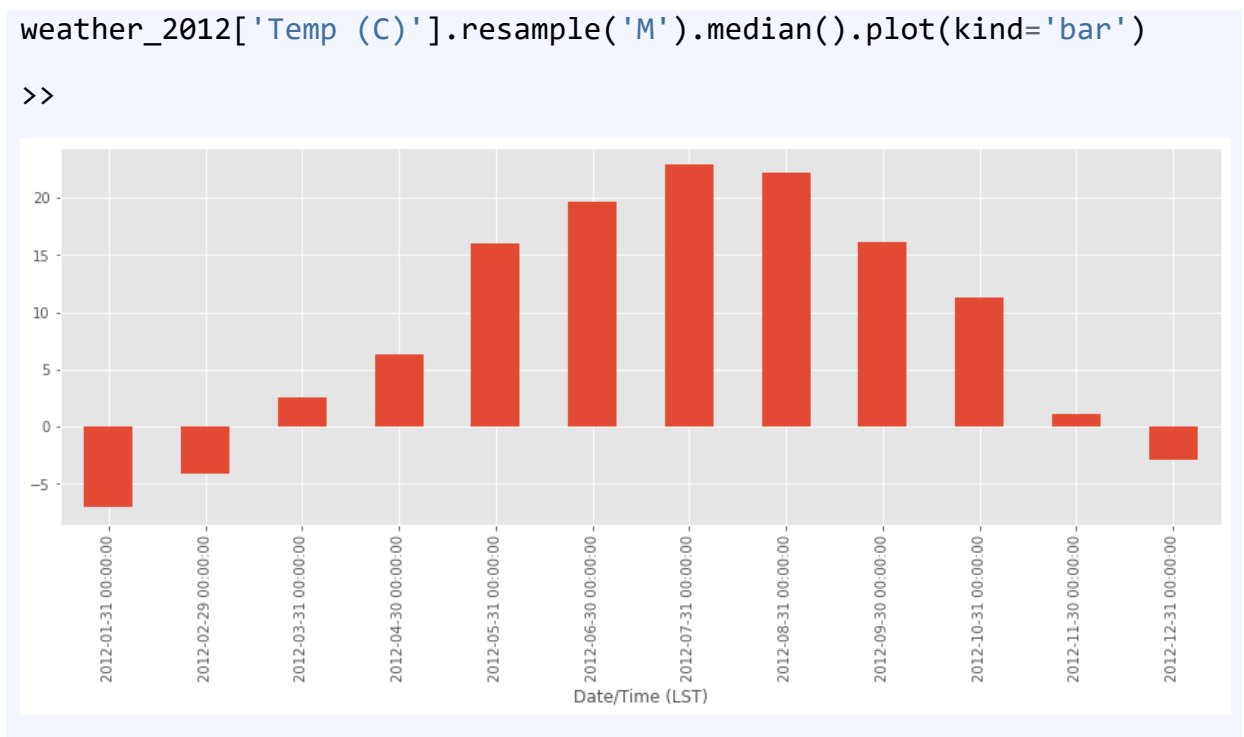


Рис. 1.8. Графік медіанної температури за місяцями протягом 2012 року

Бачимо, що липень та серпень виявились найбільш теплими місяцями (хто б міг подумати).

Стосовно снігу, можна вважати, що ми маємо справу з векторами з нулів і одиниць, замість True і False:

```
is_snowing.astype(int)[:5]
```

```
Date/Time (LST)
2012-01-01 00:00:00    0
2012-01-01 01:00:00    0
2012-01-01 02:00:00    0
2012-01-01 03:00:00    0
2012-01-01 04:00:00    0
```

А потім використовувати *resample()*, щоб знайти відсоток часу, коли йшов сніг (рис. 1.9).

```
is_snowing.astype(int).resample('M').mean()
```

```
Date/Time (LST)
2012-01-31    0.240591
2012-02-29    0.162356
2012-03-31    0.087366
2012-04-30    0.015278
2012-05-31    0.000000
2012-06-30    0.000000
2012-07-31    0.000000
2012-08-31    0.000000
2012-09-30    0.000000
2012-10-31    0.000000
2012-11-30    0.038889
2012-12-31    0.251344
Freq: M, Name: Weather, dtype: float64
```

```
is_snowing.astype(int).resample('M').mean().plot(kind='bar')
```

```
>>
```

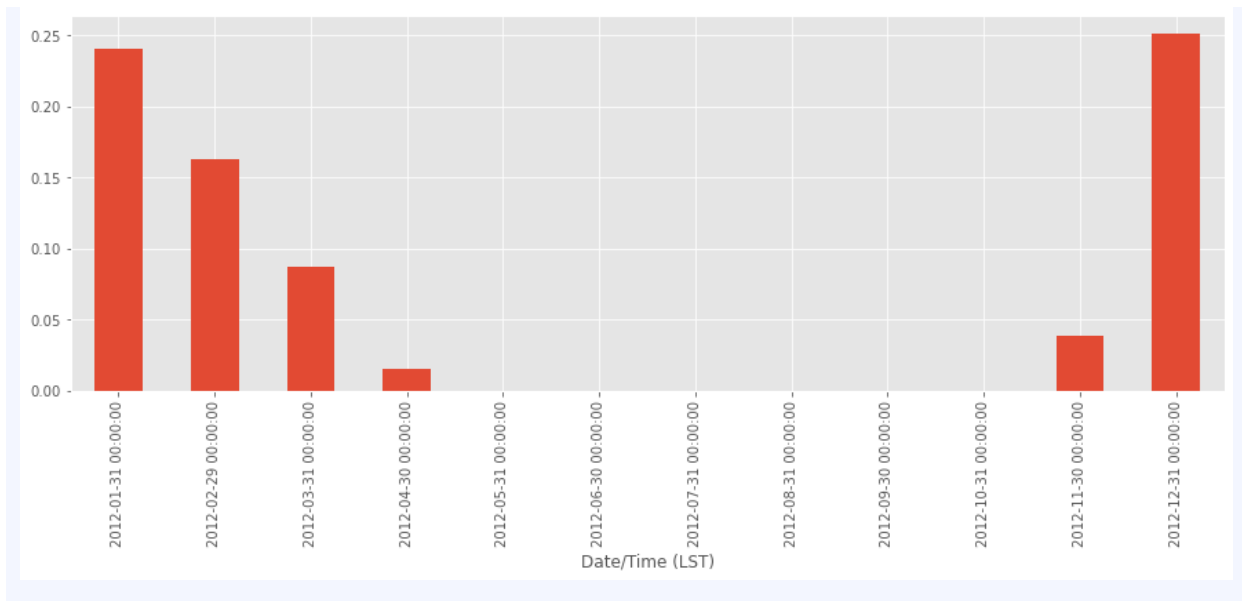


Рис. 1.9. Графік, що показує відсоток випадання снігу за місяцями протягом 2012р.

З графіку робимо висновок, що найбільш часто сніг йшов у грудні.

Тепер об'єднаємо це зі статистикою по температурі в один DataFrame та побудуємо суміщений графік (рис. 1.10).

```
temperature = weather_2012['Temp (C)'].resample('M').median()
is_snowing = weather_2012['Weather'].str.contains('Snow')
snowiness = is_snowing.astype(int).resample('M').mean()
```

```
temperature.name = "Температура"
snowiness.name = "Сніжність"
```

```
stats = pd.concat([temperature, snowiness], axis=1)
stats
>>
```

Date/Time (LST)	Температура	Сніжність
2012-01-31	-7.05	0.240591
2012-02-29	-4.10	0.162356
2012-03-31	2.60	0.087366
2012-04-30	6.30	0.015278
2012-05-31	16.05	0.000000
2012-06-30	19.60	0.000000
2012-07-31	22.90	0.000000
2012-08-31	22.20	0.000000
2012-09-30	16.10	0.000000
2012-10-31	11.30	0.000000
2012-11-30	1.05	0.038889
2012-12-31	-2.85	0.251344


```
stats.plot(kind='bar')
```

```
>>
```

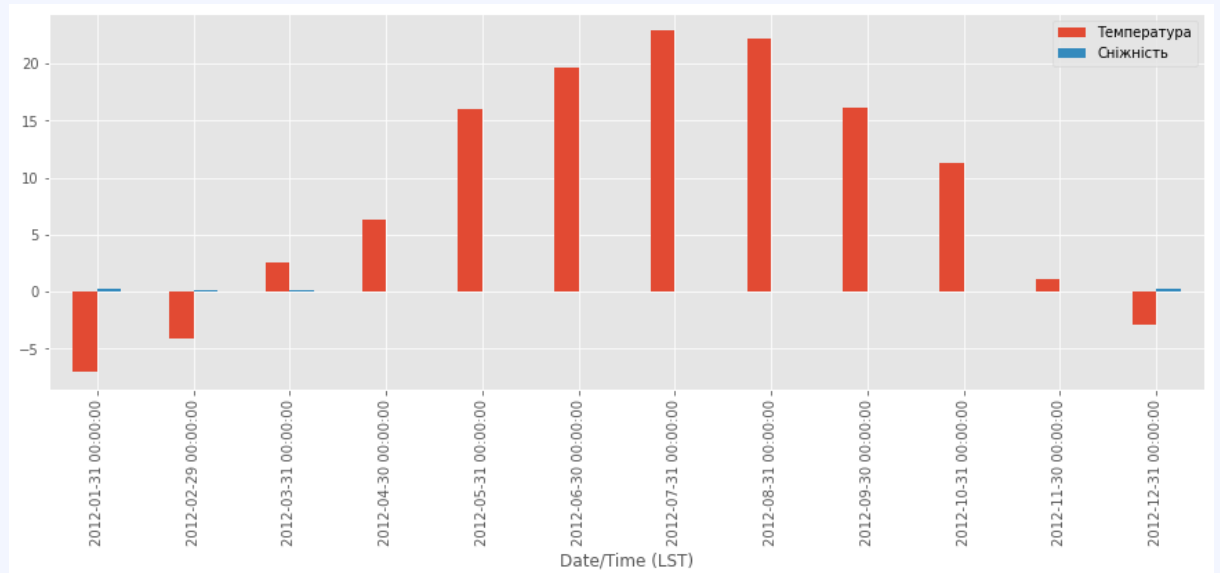
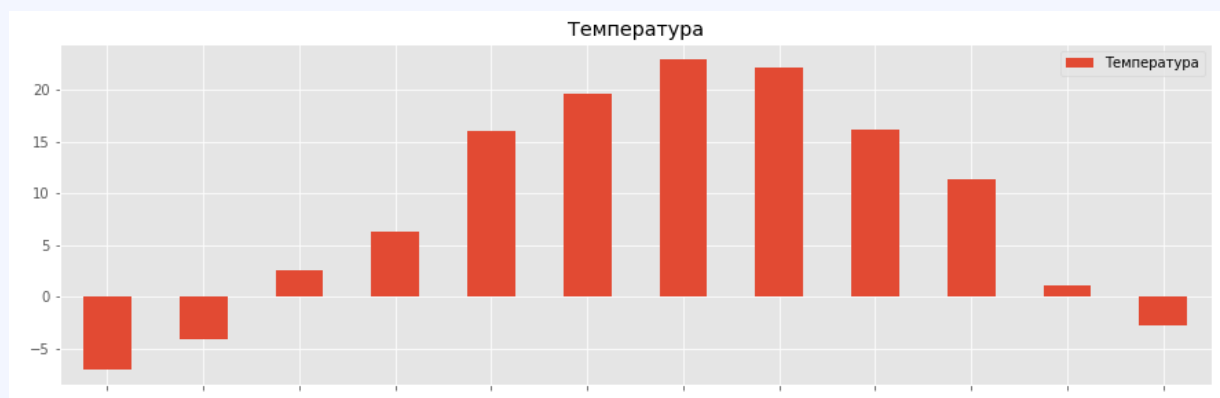


Рис. 1.10. Графік розподілу температури та рівня снігу за місяцями протягом 2012 року

Бачимо, що масштаби не співпадають. Тому краще зробити на двох окремих графіках (рис. 1.11):

```
stats.plot(kind='bar', subplots=True, figsize=(15, 10))
```

```
>>
```



а

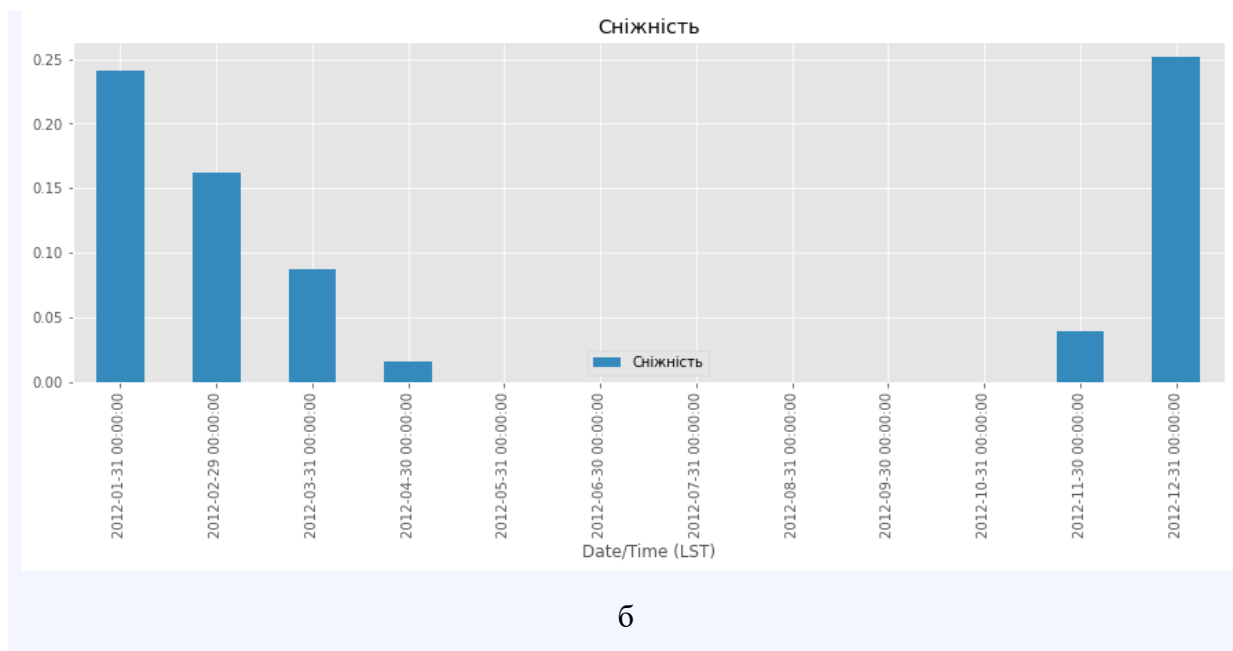


Рис. 1.11. Графік розподілу температури та рівня снігу за місяцями протягом 2012 року: а – розподіл температури; б – рівень снігу

1.4. Аналіз набору даних "Титанік"

Швидкий погляд на дані

Завантажимо набір даних з інформацією про пасажирів "Титаніка".

➤ Посилання:

<https://github.com/agconti/kaggle-titanic/raw/master/data/train.csv>

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns #розширення функціоналу matplotlib
import numpy as np
sns.set_style("ticks")

titanic_full_df = pd.read_csv("https://github.com/agconti/kaggle-titanic/
raw/master/data/train.csv", sep=",")
```

Визначимо розмірність таблиці.

```
titanic_full_df.shape

>>

(891, 12)
```

І переглянемо інформацію.

```
titanic_full_df.info()

>>

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
#   Column          Non-Null Count  Dtype
---  -
0   PassengerId     891 non-null   int64
1   Survived        891 non-null   int64
2   Pclass          891 non-null   int64
3   Name            891 non-null   object
4   Sex             891 non-null   object
5   Age             714 non-null   float64
6   SibSp           891 non-null   int64
7   Parch           891 non-null   int64
8   Ticket          891 non-null   object
```

```
9   Fare          891 non-null   float64
10  Cabin          204 non-null   object
11  Embarked       889 non-null   object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

- PassengerId – ідентифікатор пасажирів
 - Survival – поле в якому зазначено врятувалась людина (1) чи ні (0)
- Pclass – містить соціально-економічний статус:
 - високий
 - середній
 - низький
- Name – ім'я пасажирів
- Sex – стать пасажирів
- Age – вік
- SibSp – містить інформацію про кількість родичів 2-го порядку (чоловік, дружина, брати, сестри)
- Parch – містить інформацію про кількість родичів на борту 1-го порядку (мати, батько, діти)
- Ticket – номер квитка
- Fare – ціна квитка
- Cabin – каюта
- Embarked – порт посадки
 - C – Cherbourg
 - Q – Queenstown
 - S – Southampton

Переглянемо статистичні характеристики набору даних:

```
titanic_full_df.describe()
```

```
>>
```

	PassengerId	Survived	Pclass	Age	SibSp
count	891.000000	891.000000	891.000000	714.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008
std	257.353842	0.486592	0.836071	14.526497	1.102743
min	1.000000	0.000000	1.000000	0.420000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000
50%	446.000000	0.000000	3.000000	28.000000	0.000000
75%	668.500000	1.000000	3.000000	38.000000	1.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000

	Parch	Fare
count	891.000000	891.000000
mean	0.381594	32.204208
std	0.806057	49.693429
min	0.000000	0.000000
25%	0.000000	7.910400
50%	0.000000	14.454200
75%	0.000000	31.000000
max	6.000000	512.329200

А також перші записи в таблиці.

```
titanic_full_df.head()
```

```
>>
```

	PassengerId	Survived	Pclass	\
0	1	0	3	
1	2	1	1	
2	3	1	3	
3	4	1	1	
4	5	0	3	

	Name	Sex	Age	SibSp	\
0	Braund, Mr. Owen Harris	male	22.0	1	
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	
2	Heikkinen, Miss. Laina	female	26.0	0	
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	
4	Allen, Mr. William Henry	male	35.0	0	

	Parch	Ticket	Fare	Cabin	Embarked
0	0	A/5 21171	7.2500	NaN	S
1	0	PC 17599	71.2833	C85	C
2	0	STON/O2. 3101282	7.9250	NaN	S

3	0	113803	53.1000	C123	S
4	0	373450	8.0500	NaN	S

І останні записи в таблиці.

```
titanic_full_df.tail()
```

```
>>
```

	PassengerId	Survived	Pclass	Name
\				
886	887	0	2	Montvila, Rev. Juozas
887	888	1	1	Graham, Miss. Margaret Edith
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"
889	890	1	1	Behr, Mr. Karl Howell
890	891	0	3	Dooley, Mr. Patrick

	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
886	male	27.0	0	0	211536	13.00	NaN	S
887	female	19.0	0	0	112053	30.00	B42	S
888	female	NaN	1	2	W./C. 6607	23.45	NaN	S
889	male	26.0	0	0	111369	30.00	C148	C
890	male	32.0	0	0	370376	7.75	NaN	Q

Переглянемо список пасажирів, які вижили.

```
titanic_full_df[titanic_full_df['Survived']==True]
```

```
>>
```

	PassengerId	Survived	Pclass	\
1	2	1	1	
2	3	1	3	
3	4	1	1	
8	9	1	3	
9	10	1	2	
..	...	...	...	
875	876	1	3	
879	880	1	1	
880	881	1	2	
887	888	1	1	
889	890	1	1	

	Name	Sex	Age	SibSp
\				
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1
2	Heikkinen, Miss. Laina	female	26.0	0
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1

8	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0
9	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1
..	...	...	...	...
875	Najib, Miss. Adele Kiamie "Jane"	female	15.0	0
879	Potter, Mrs. Thomas Jr (Lily Alexenia Wilson)	female	56.0	0
880	Shelley, Mrs. William (Imanita Parrish Hall)	female	25.0	0
887	Graham, Miss. Margaret Edith	female	19.0	0
889	Behr, Mr. Karl Howell	male	26.0	0

	Parch	Ticket	Fare	Cabin	Embarked
1	0	PC 17599	71.2833	C85	C
2	0	STON/O2. 3101282	7.9250	NaN	S
3	0	113803	53.1000	C123	S
8	2	347742	11.1333	NaN	S
9	0	237736	30.0708	NaN	C
..	...	...	...	...	...
875	0	2667	7.2250	NaN	C
879	1	11767	83.1583	C50	C
880	1	230433	26.0000	NaN	S
887	0	112053	30.0000	B42	S
889	0	111369	30.0000	C148	C

[342 rows x 12 columns]

Підрахуємо кількість пасажирів, які вижили та загинули.

```
titanic_full_df["Survived"].value_counts()

>>

0    549
1    342
```

Переглянемо декілька довільних записів.

```
titanic_full_df.loc[442 : 450 : 2, ["Age", "Sex"]]

>>

   Age  Sex
442  25.0  male
444   NaN  male
446  13.0  female
448   5.0  female
450  36.0  male
```

Виведемо список унікальних значень в стовпці "Embarked".

```
titanic_full_df["Embarked"].unique()

>>
array(['S', 'C', 'Q', nan], dtype=object)
```

А також неунікальних:

```
titanic_full_df["Embarked"].nunique()

>>
3
```

Попередня обробка даних

Додамо ще один стовпець, який назвемо "Relatives" – родичі. Нехай він буде містити загальну кількість родичів пасажирів.

```
titanic_full_df["Relatives"] = titanic_full_df["SibSp"] + titanic_full_df["Parch"]
```

Переглянемо результат:

```
titanic_full_df[["SibSp", "Parch", "Relatives"]].head()

>>
   SibSp  Parch  Relatives
0      1     0          1
1      1     0          1
2      0     0          0
3      1     0          1
4      0     0          0
```

Замінімо числові значення в стовпці "Pclass" на рядкові.

```
titanic_full_df["Pclass"].replace({1: "Мажори", 2: "Середній клас", 3:
"Роботяги"}, inplace=True)
titanic_full_df["Pclass"].value_counts()

>>
Роботяги      491
Мажори        216
Середній клас  184
```

Відсортуємо записи за вартістю квитка.


```
titanic_full_df.sort_values(by="Fare", ascending=False)
```

	PassengerId	Survived	Pclass	Name \
258	259	1	Мажори	Ward, Miss. Anna
737	738	1	Мажори	Lesurer, Mr. Gustave J
679	680	1	Мажори	Cardeza, Mr. Thomas Drake Martinez
88	89	1	Мажори	Fortune, Miss. Mabel Helen
27	28	0	Мажори	Fortune, Mr. Charles Alexander
..	...	...	...	...
633	634	0	Мажори	Parr, Mr. William Henry Marsh
413	414	0	Середній клас	Cunningham, Mr. Alfred Fleming
822	823	0	Мажори	Reuchlin, Jonkheer. John George
732	733	0	Середній клас	Knight, Mr. Robert J
674	675	0	Середній клас	Watson, Mr. Ennis Hastings

	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked \
258	female	35.0	0	0	PC 17755	512.3292	NaN	C
737	male	35.0	0	0	PC 17755	512.3292	B101	C
679	male	36.0	0	1	PC 17755	512.3292	B51 B53 B55	C
88	female	23.0	3	2	19950	263.0000	C23 C25 C27	S
27	male	19.0	3	2	19950	263.0000	C23 C25 C27	S
..	...	...	...	...	...	...	...	...
633	male	NaN	0	0	112052	0.0000	NaN	S
413	male	NaN	0	0	239853	0.0000	NaN	S
822	male	38.0	0	0	19972	0.0000	NaN	S
732	male	NaN	0	0	239855	0.0000	NaN	S
674	male	NaN	0	0	239856	0.0000	NaN	S

	Relatives
258	0
737	0
679	1
88	5
27	5
..	...
633	0
413	0
822	0
732	0
674	0

```
[891 rows x 13 columns]
```

Визначимо кількість пропущених даних.

```
titanic_full_df.isnull().sum()

>>
PassengerId      0
Survived          0
Pclass           0
Name             0
Sex              0
Age             177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin           687
Embarked         2
Relatives        0
```

Видалимо пропущені записи із колонок *Age* та *Embarked*

```
titanic_full_df = titanic_full_df.dropna(subset=["Age", "Embarked"])
```

Тепер перевіримо наявність пропусків.

```
titanic_full_df.isnull().any()

>>
PassengerId      False
Survived          False
Pclass           False
Name             False
Sex              False
Age             False
SibSp            False
Parch            False
Ticket           False
Fare             False
Cabin            True
Embarked         False
Relatives        False
```

Заповнимо пропущені дані.

```
titanic_full_df = titanic_full_df.fillna("Невідомо")
titanic_full_df.head()

>>
```

	PassengerId	Survived	Pclass	\
0	1	0	Роботяги	
1	2	1	Мажори	
2	3	1	Роботяги	
3	4	1	Мажори	
4	5	0	Роботяги	

	Name	Sex	Age	SibSp	\
0	Braund, Mr. Owen Harris	male	22.0	1	
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	
2	Heikkinen, Miss. Laina	female	26.0	0	
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	
4	Allen, Mr. William Henry	male	35.0	0	

	Parch	Ticket	Fare	Cabin	Embarked	Relatives
0	0	A/5 21171	7.2500	Невідомо	S	1
1	0	PC 17599	71.2833	C85	C	1
2	0	STON/O2. 3101282	7.9250	Невідомо	S	0
3	0	113803	53.1000	C123	S	1
4	0	373450	8.0500	Невідомо	S	0

Аналіз даних

Визначимо середній вік пасажирів

```
titanic_full_df["Age"].mean()

>>

29.64209269662921
```

Створимо таблицю розподілу за статтю пасажирів, які вижили або загинули.

```
titanic_full_df[["Sex", "Survived"]].pivot_table(index=["Sex"],
columns=["Survived"], aggfunc=len)

>>
```

Survived	0	1
Sex		
female	64	195
male	360	93

А тепер ще і визначимо їх середній вік.

```
titanic_full_df[["Sex", "Survived", "Age"]].pivot_table(values=["Age"],  
index=["Sex"], columns=["Survived"], aggfunc="mean")
```

```
>>
```

	Age	
Survived	0	1
Sex		
female	25.046875	28.630769
male	31.618056	27.276022

Згрупуємо пасажирів за класом та визначимо середній вік у кожному класі.

```
titanic_full_df.groupby("Pclass").mean()["Age"]
```

```
>>
```

Pclass	
Мажори	38.105543
Роботяги	25.140620
Середній клас	29.877630

Переглянемо усереднені дані за класом "Роботяги"

```
titanic_full_df.groupby("Pclass").mean().loc["Роботяги"]
```

```
>>
```

PassengerId	441.219718
Survived	0.239437
Age	25.140620
SibSp	0.585915
Parch	0.456338
Fare	13.229435
Relatives	1.042254

Побудуємо гістограми для всіх колонок, де це можливо (рис. 1.12).

```
titanic_full_df.hist()
```

```
>>
```

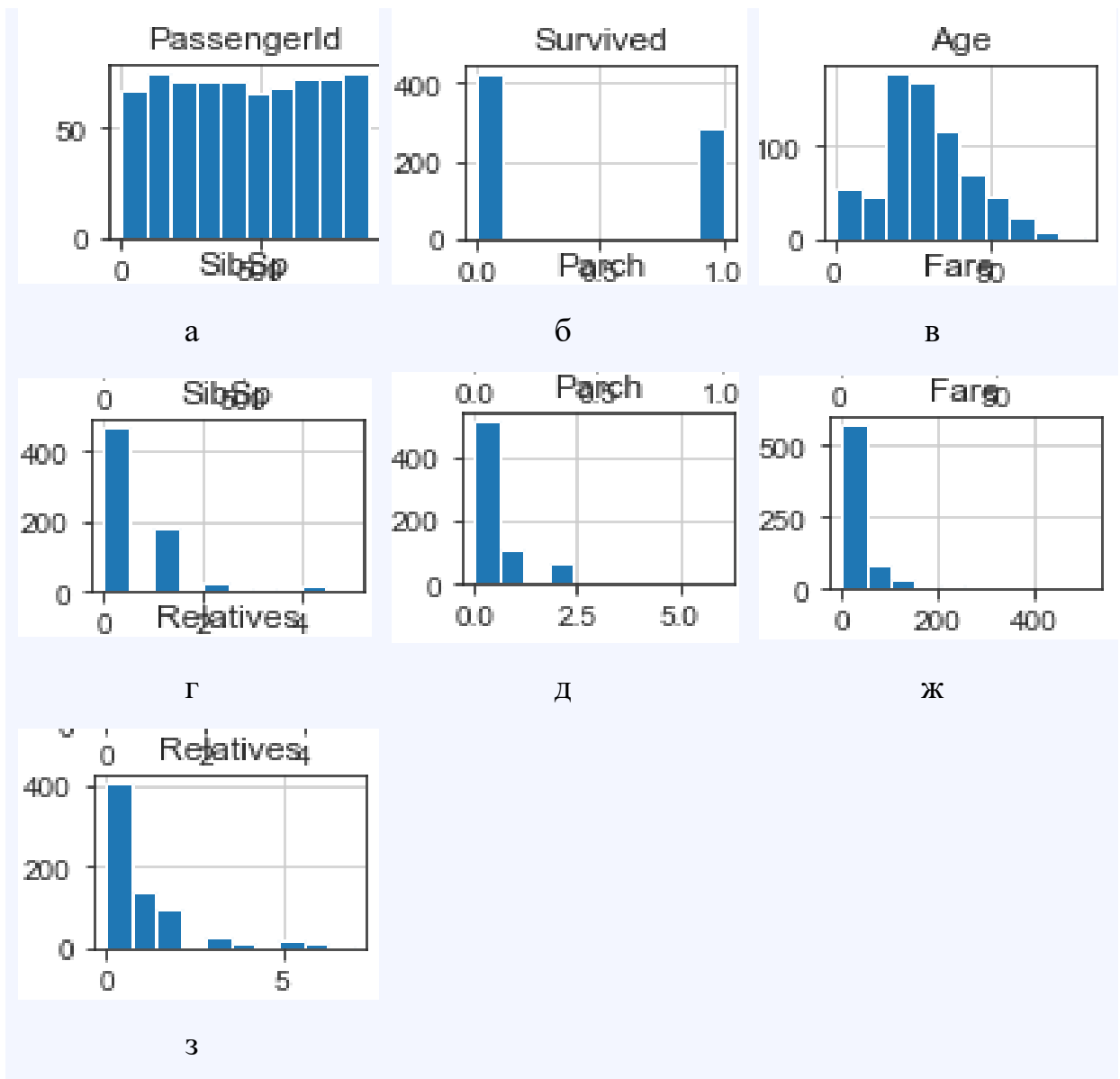


Рис. 1.12. а – ідентифікатор пасажирів; б – кількість пасажирів, які вижили та загинули; в – вік пасажирів; г – кількість родичів 2-го порядку; д – кількість родичів 1-го порядку; ж – вартість квитка; з – загальна кількість родичів

Окрема гістограма розподілу за віком (рис. 1.13).

```
titanic_full_df["Age"].hist()
>>
```

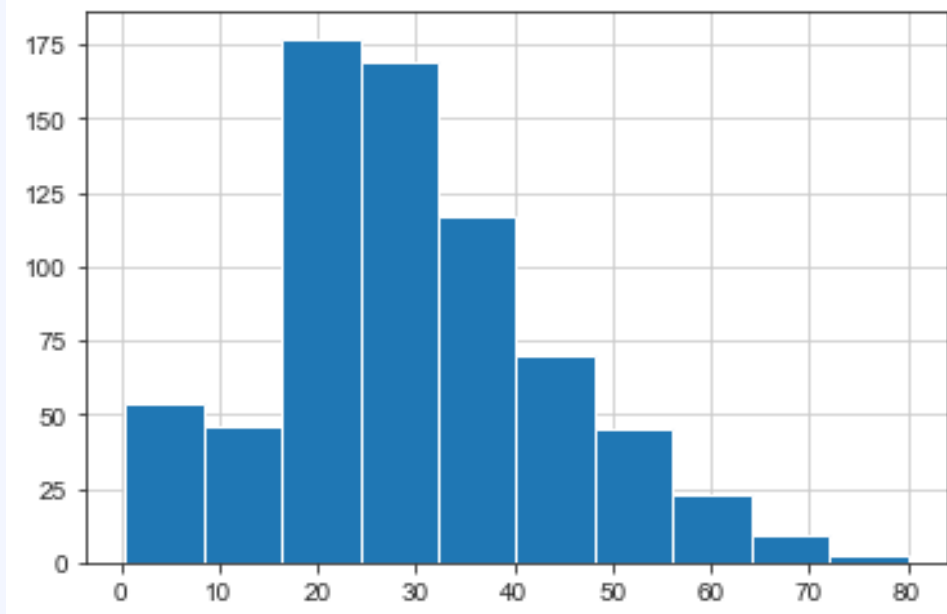


Рис. 1.13. Розподіл пасажирів за віком

Кругова діаграма розподілу за статтю (рис. 1.14):

```
titanic_full_df["Sex"].value_counts().plot(kind="pie", figsize=(5, 5),
fontSize=16);
```

```
>>
```

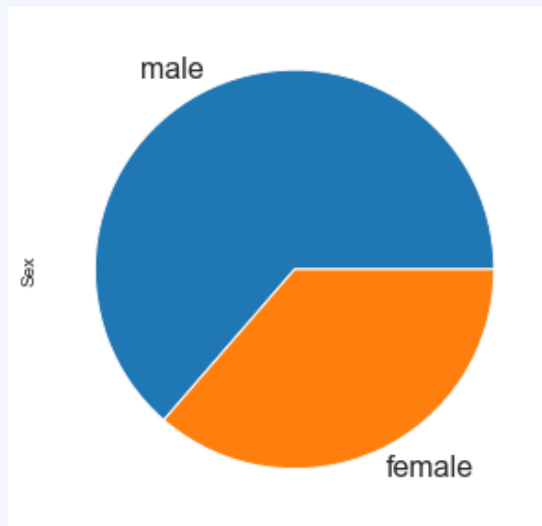


Рис. 1.14. Розподіл пасажирів за статтю

Те саме для класів (рис. 1.15):

```
titanic_full_df["Pclass"].value_counts().plot(kind="pie", figsize=(5, 5),
fontSize=16);
```

```
>>
```



Рис. 1.15. Розподіл пасажирів за класами

Стовпчикова діаграма за кількістю виживших та їх статтю (рис. 1.16):

```
titanic_full_df[["Sex", "Survived"]].pivot_table(index=["Sex"],
columns=["Survived"], aggfunc=len).plot(kind="bar");
```

>>

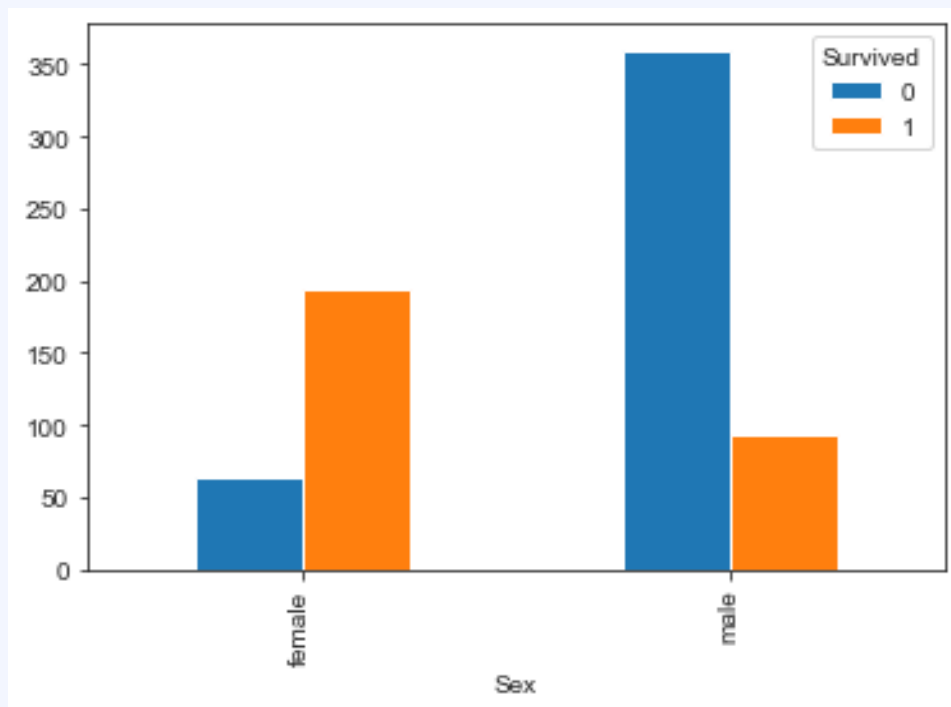


Рис. 1.16. Кількість пасажирів, що вижили та загинули з врахуванням статі

Графік летальності за віком та статтю (рис. 1.17):

```
titanic_full_df[["Age", "Survived"]].pivot_table(index=["Age"],  
columns=["Survived"], aggfunc=len).plot();
```

```
>>
```

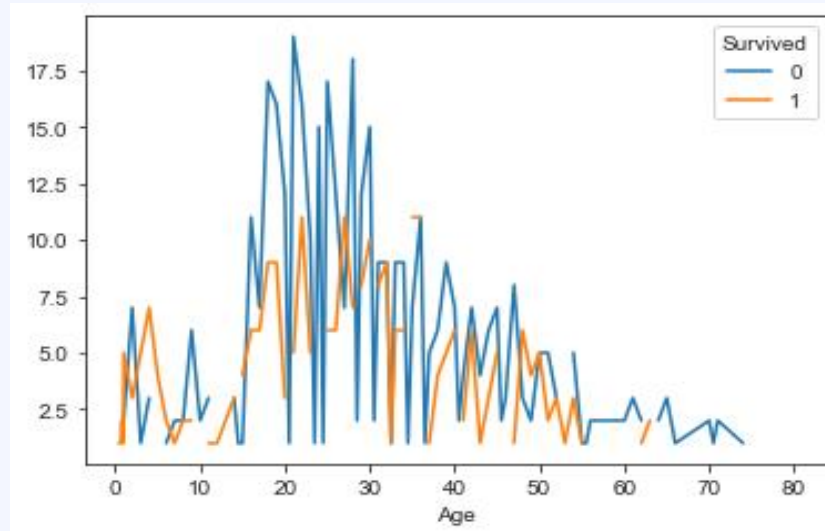


Рис. 1.17. Графік летальності за віком та статтю

1.5 Завдання для самостійного виконання

Загальні завдання для всіх варіантів:

Швидкий погляд на дані:

1. Завантажте набір даних.
2. Виведіть заголовок таблиці (перші 5 записів).
3. Скільки рядків і стовпців в таблиці?
4. Які назви стовпців?
5. Які типи даних у різних стовпців?
6. Скільки в кожному з них унікальних значень?
7. Скільки пропущених значень?

Варіант 1

Набір даних: злочинність в Лос-Анджелесі (la-crime.csv).

Аналіз даних:

1. У наборі даних є інформація про вік, стать, і походження кожної жертви. Люди якого походження найчастіше є жертвами злочинів?
2. Чи вірно, що жінки частіше виявляються жертвами в порівнянні з чоловіками?
3. Вивчіть розподіл кількості злочинів за віком. Яка тенденція? Люди якого віку найчастіше піддаються злочинам? Чи є локальні мінімуми? Використовуйте тип графіків hist.
4. Як відрізняється ймовірність для жінок і чоловіків стати жертвою в залежності від віку? Побудуйте візуалізацію.
5. В якому віковому проміжку чоловіки частіше стають жертвами злочинів?

Походження:

A - Other Asian	H - Hispanic / Latin / Mexican	P - Pacific Islander	Z - Asian Indian
B - Black	I - American Indian / Alaskan Native	S - Samoan	
C - Chinese	J - Japanese	U - Hawaiian	
D - Cambodian	K - Korean	V - Vietnamese	
F - Filipino	L - Laotian	W - White	
G - Guamanian	O - Other	X - Unknown	

Варіант 2

Набір даних: дорожня поліція Стенфорда (police_project.csv).

Аналіз даних:

1. З'ясуйте, чоловіки чи жінки частіше перевищують швидкість.
2. Порівняйте, наскільки часто для чоловіків та жінок проводився обшук (search_conducted).
3. Проаналізуйте найбільш часті причини (search_type) для оголошення в розшук. Візуалізуйте дані за допомогою стовпчикової діаграми.
4. Як часто водіїв розшукують для проведення обшуку (Protective Frisk)? Додайте до таблиці новий стовпець 'frisk', який буде містити булеві дані про проведення обшуку (True, якщо обшук проводився).
5. Побудуйте графік кількості зупинок за роками. В якому році було найменше зупинок водіїв?

Варіант 3

Набір даних: злочинність в Лос-Анджелесі (la-crime.csv).

Аналіз даних:

1. Визначте 10 найпоширеніших злочинів в LA. Побудуйте графік.
2. Від яких злочинів частіше потерпають жінки, а від яких чоловіки?
3. Люди якого походження найчастіше піддаються злочинам?
4. Відсортуйте райони по кількості злочинів. Побудуйте графік, що показує найбезпечніші і небезпечні райони.

5. Люди якого походження найчастіше страждають від злочинів в кожному з районів?

Походження:

A - Other Asian	H - Hispanic / Latin / Mexican	P - Pacific Islander	Z - Asian Indian
B - Black	I - American Indian / Alaskan Native	S - Samoan	
C - Chinese	J - Japanese	U - Hawaiian	
D - Cambodian	K - Korean	V - Vietnamese	
F - Filipino	L - Laotian	W - White	
G - Guamanian	O - Other	X - Unknown	

Варіант 4

Набір даних: дорожня поліція Стенфорда (police_project.csv).

Аналіз даних:

1. Перевірте, білих чи темношкірих людей частіше зупиняє поліція.
2. З'ясуйте, як частота зупинок через наркотики залежить від часу доби. Побудуйте графік.
3. Чи правда, що більшість зупинок водіїв трапляється вночі? Побудуйте графік типу hist для розподілу кількості зупинок за часом.
4. Виявіть хибні дані в стовпці 'stop_duration' та замініть їх на NaN.
5. Визначте середній час зупинки для кожної з причин зупинки (violation_raw). Побудуйте графік у вигляді стовпчикової діаграми.

Варіант 5

Набір даних: продажі відеоігор (vg-sales.csv).

Аналіз даних:

1. Знайдіть ТОП-10 ігор за кількістю продажів на платформі PC за всі роки (1980-2020).
2. Побудуйте графік продажів гри Super Mario Bros. в Європі за роками.

3. Визначте, ігри якого жанру найчастіше продавались в Північній Америці у 2010 році. Підрахуйте сумарну кількість проданих ігор цього жанру.
4. Побудуйте кругову діаграму, яка відображає долю проданих ігор 5 найпопулярніших видавництв (Publisher) по відношенню до загальної кількості проданих ігор за останні 10 років.
5. Визначте середню кількість всіх проданих ігор у світі за кожні 5 років, починаючи з 1980. Побудуйте графік hist.

Варіант 6

Набір даних: ринок смартфонів в Україні (phones_data.csv).

Аналіз даних:

1. Побудуйте стовпчикову діаграму розподілу цін (best price) на смартфони.
2. Визначте, як залежить ціна на смартфон від діагоналі екрану.
3. Знайдіть ТОП-5 найдешевших смартфонів з найбільшою ємністю акумулятора.
4. Створіть кругову діаграму розподілу долі виробників смартфонів на ринку (візьміть лише перші 10 виробників з найбільшою кількістю смартфонів).
5. Який смартфон виробника Samsung є найбільш популярним в цінovій категорії 10-15 тис. грн?

Варіант 7

Набір даних: продажі відеоігор (vg-sales.csv).

Аналіз даних:

1. Знайдіть ТОП-10 видавництв (Publisher) ігор за кількістю продажів у Європі за всі роки (1980-2020).

2. Визначте гру, яка найкраще продавалась у Японії в 2015 році. Підрахуйте долю її продажів у Японії по відношенню до загальної кількості проданих ігор в цьому регіоні та порівняйте з аналогічним показником цієї гри для інших регіонів.
3. Побудуйте суміщений графік (стовпчикову діаграму) кількості проданих ігор на PC та PS4 за 2015-2020 роки. Порівняйте, як змінювалась популярність даних платформ.
4. Побудуйте кругову діаграму, яка показуватиме долю популярності кожного жанру ігор в Європі у 2020 році.
5. Порівняйте рівень світової популярності ігор серії “Call of Duty” та “Battlefield” протягом 2005-2020 років. Побудуйте графік динаміки їх популярності за вказані роки.

Варіант 8

Набір даних: проекти на Netflix (netflix_titles.csv).

Аналіз даних:

1. Побудуйте стовпчикову діаграму розподілу кількості проектів за телевізійним рейтингом (TV-MA, TV-14, TV-PG тощо).
2. Чи правда, що останніми роками Netflix більше фокусується на серіалах (TV Show), ніж на фільмах? Проаналізуйте останні 5 років.
3. Визначте, яка країна випустила найбільше проектів на Netflix у 2020 році. А яка найменше?
4. В якому місяці на Netflix зазвичай виходить найбільше серіалів?
5. Визначте 5 найбільш популярних жанрів на Netflix та побудуйте відповідну кругову діаграму їх розподілу.

Варіант 9

Набір даних: ринок смартфонів в Україні (phones_data.csv).

Аналіз даних:

1. Визначте 10 найбільш дорогих смартфонів на ринку з датою виходу у 2020 році (порівнювати за категорією best price).
2. Чи правда, що найбільш дорогий смартфон виробника Prestigio коштує менше, ніж найдешевший смартфон від Apple?
3. Побудуйте стовпчикову діаграму розподілу кількості смартфонів на ринку в залежності від виробника.
4. Визначте, як змінювалась ємність акумулятора смартфонів від Samsung (з ціною вище 15 тис. грн) за період з 2015 по 2020 рік.
5. Порівняйте середню популярність смартфонів на базі Android та iOS за період з 2015 по 2020 рік.

Варіант 10

Набір даних: проекти на Netflix (netflix_titles.csv).

Аналіз даних:

1. Визначте, проектів якого жанру було найбільше випущено на Netflix у 2020 році.
2. Визначте ТОП-10 країн за кількістю проектів на Netflix. Побудуйте діаграму їх розподілу.
3. Яка країна випустила найбільше старих фільмів (дата прем'єри до 1960 року), доступних на Netflix?
4. Фільми на Netflix частіше виходять весною чи восени? А серіали?
5. В якому році на Netflix було випущено найбільшу кількість фільмів?

Варіант 11

Набір даних: додатки в Google Play Store (googleplaystore.csv).

Аналіз даних:

1. Визначте ТОП-10 додатків, які мають найбільше завантажень. Побудуйте стовпчикову діаграму.
2. Чи правда, що безкоштовні додатки з категорії «Ігри» мають більший середній рейтинг, ніж аналогічні платні додатки?
3. Побудуйте кругову діаграму розподілу частки додатків за віковими групами (Content rating).
4. Складіть список додатків, які не оновлювались більш ніж 7 років.
5. Порівняйте середню вагу (в мегабайтах) безкоштовних додатків із довільних 5 категорій.

Варіант 12

Набір даних: додатки в Google Play Store (googleplaystore.csv).

Аналіз даних:

1. Визначте ТОП-10 категорій, додатки в яких мають найбільшу кількість завантажень. Побудуйте стовпчикову діаграму.
2. Чи правда, що безкоштовні додатки з категорії «Шопінг» мають більшу кількість завантажень, ніж платні додатки з категорії «Освіта»?
3. Проаналізуйте, чи змінюється частка завантажень платних додатків у категорії «Ігри» в залежності від вікової групи?
4. Визначте 5 категорій, в яких платні додатки мають найбільшу кількість відгуків.
5. Знайдіть платні додатки в категорії «Сім'я», які не оновлювались більш ніж 5 років.

ПРАКТИКУМ 2. НЕЙРОННІ МЕРЕЖІ В KERAS

2.1 Загальні відомості

Keras — відкрита нейромережева бібліотека, написана мовою Python. Вона здатна працювати поверх TensorFlow, Microsoft Cognitive Toolkit, R, Theano та інших модулів. Keras дозволяє легко проводити експерименти з мережами глибинного навчання. Основними перевагами є зручність в користуванні, модульність та розширюваність.

Автор бібліотеки - François Chollet, Deep learning researcher at Google.

- Повна документація за бібліотекою: <https://keras.io/api/>

Keras працює з усіма відомими архітектурами нейронних мереж.

Список модулів Keras:

- Layers – містить набір прошарків нейронних мереж
- Data preprocessing – для попередньої обробки даних
- Optimizers – набір оптимізаторів
- Metrics – набір метрик
- Losses – набір функцій втрат (критеріїв якості)
- Built-in small datasets – вбудовані набори даних
- та інші...

Послідовність кроків для створення нейронної мережі:

- 1) Описати архітектуру мережі
- 2) Описати вхідні значення
- 3) Описати умови навчання (Compilation)
- 4) Навчити (кілька разів?)
- 5) Оцінити якість моделі
- 6) Застосувати

Приклад створення нейромережі

Розглянемо приклад. Необхідно створити нейронну мережу для класифікації мобільних телефонів за ціновою категорією (всього є 4 класи: 0 – найдешевші моделі, 1 – більш дорогі ... 3 – найбільш дорогі).

Перш за все, потрібно імпортувати необхідні модулі.

```
# Для роботи з даними
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split

# Для роботи з Keras
from tensorflow.keras.models import Sequential # модель прямого
розповсюдження
from tensorflow.keras.layers import Dense # повнозв'язні прошарки
```

Наступний код для оптимізації відеокарт (щоб працювало без помилок).

```
import tensorflow as tf
physical_devices = tf.config.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(physical_devices[0], True)
```

Завантажуємо навчальні дані.

```
data = pd.read_csv("../phones_price.csv")
data.head()
```

```
>>
```

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	\
0	842	0	2.2	0	1	0	7	0.6	
1	1021	1	0.5	1	0	1	53	0.7	
2	563	1	0.5	1	2	1	41	0.9	
3	615	1	2.5	0	0	0	10	0.8	
4	1821	1	1.2	0	13	1	44	0.6	

	mobile_wt	n_cores	...	px_height	px_width	ram	sc_h	sc_w	talk_time	\
0	188	2	...	20	756	2549	9	7	19	
1	136	3	...	905	1988	2631	17	3	7	
2	145	5	...	1263	1716	2603	11	2	9	
3	131	6	...	1216	1786	2769	16	8	11	
4	141	2	...	1208	1212	1411	8	2	15	

	three_g	touch_screen	wifi	price_range
0	0	0	1	1
1	1	1	0	2
2	1	1	0	2
3	1	0	0	2
4	1	1	0	1

[5 rows x 21 columns]

Перевіряємо, наскільки збалансована кількість зразків кожного класу. В ідеальному випадку, доля зразків кожного класу має бути приблизно однаковою.

```
data['price_range'].value_counts(normalize=True)
```

0	0.25
1	0.25
2	0.25
3	0.25

Як бачимо, у нас 21 стовпець даних. З них перші 20 - змінні X, останній стовпець – клас телефону (цінова категорія), Y.

Запишемо стовпці з характеристиками об'єктів до змінної X.

```
X = data.drop('price_range', axis = 1)
X
```

>>

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	\
0	842	0	2.2	0	1	0	7	
1	1021	1	0.5	1	0	1	53	
2	563	1	0.5	1	2	1	41	
...	...	...	...	...	..	...	...	
1997	1911	0	0.9	1	1	1	36	
1998	1512	0	0.9	0	4	1	46	
1999	510	1	2.0	1	5	1	45	

	m_dep	mobile_wt	n_cores	pc	px_height	px_width	ram	sc_h	sc_w	\
0	0.6	188	2	2	20	756	2549	9	7	
1	0.7	136	3	6	905	1988	2631	17	3	
2	0.9	145	5	6	1263	1716	2603	11	2	
...	...	...	...	..	...	...	...	...	...	
1997	0.7	108	8	3	868	1632	3057	9	1	
1998	0.1	145	5	5	336	670	869	18	10	
1999	0.9	168	6	16	483	754	3919	19	4	

	talk_time	three_g	touch_screen	wifi
0	19	0	0	1
1	7	1	1	0
2	9	1	1	0
...	...	...	...	...
1997	5	1	1	0
1998	19	1	1	1
1999	2	1	1	1

[2000 rows x 20 columns]

А також створимо змінну з правильними відповідям Y. Для цього переведемо стовпець *price_range* з початкової таблиці до формату **one hot encoding**.

```
Y = pd.get_dummies(data.price_range, prefix='Price range')
Y
```

```
>>
```

	Price range_0	Price range_1	Price range_2	Price range_3
0	0	1	0	0
1	0	0	1	0
2	0	0	1	0
3	0	0	1	0
4	0	1	0	0
...	...	...	...	...
1995	1	0	0	0
1996	0	0	1	0
1997	0	0	0	1
1998	1	0	0	0
1999	0	0	0	1

[2000 rows x 4 columns]

Розділимо отриманий набір даних на навчальну і тестову множини.

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.15)
```

Виконаємо стандартизацію значень змінних X, привівши кожен стовпець до нульового середнього значення та одиничної дисперсії.

```
mean = X_train.mean(axis=0)
std = X_train.std(axis=0)
```

```
X_train -= mean
X_train /= std
X_test -= mean
X_test /= std
```

Створюємо послідовну модель:

```
model = Sequential()
```

Додаємо перший прихований прошарок *Dense*. В якості аргументів спочатку вказуємо кількість нейронів в прихованому прошарку, потім розмірність вхідного вектору (скільки характеристик *X*), а також активаційну функцію. В Keras доступні такі активаційні функції:

- `relu`
- `sigmoid`
- `softmax`
- `softplus`
- `softsign`
- `tanh`
- `selu`
- `elu`
- `exponential`

Використовуємо *relu*.

```
model.add(Dense(10, input_dim=X_train.shape[1], activation="relu"))
```

Додаємо другий прихований прошарок. Вказуємо тільки кількість нейронів у ньому та активаційну функцію *relu*.

```
model.add(Dense(5, activation="relu"))
```

Додаємо вихідний прошарок. Кількість нейронів у ньому має відповідати кількості класів (якщо використовується формат one hot encoding). Активаційна функція для задачі класифікації – *softmax*.

```
model.add(Dense(4, activation="softmax"))
```

Описуємо умови навчання. Обов'язково потрібно задати оптимізатор, критерій якості (функцію помилки, яку будемо мінімізувати) та метрику (показник, за яким зручно оцінювати якість навчання).

В Keras доступні оптимізатори:

- SGD
- RMSprop
- Adam
- Adadelta
- Adagrad
- Adamax
- Nadam
- Ftrl

Зараз найчастіше використовується *Adam*.

В якості функцій втрат в Keras доступні різні варіанти, але для задач класифікації використовуємо такі:

- *binary_crossentropy* - для бінарної класифікації (коли всього два класи)
- *categorical_crossentropy* - для категоріальної класифікації (коли багато класів)

Метрики, які найчастіше використовуються для класифікації:

- *Accuracy* (доля правильних відповідей серед всіх зразків)
- *Precision* (як багато з обраних елементів дійсно є правильними?)

- *Recall* (як багато елементів було обрано серед загальної кількості елементів, які потрібно було обрати)

Обираємо метрику *accuracy*.

```
model.compile(optimizer='adam', loss='categorical_crossentropy',  
              metrics=['accuracy'])
```

Навчаємо модель. Лог навчання запишемо до змінної *history*. Необхідно обов'язково вказати наступні аргументи: навчальний набір даних (*X_train*), правильні відповіді для цього набору (*Y_train*), кількість епох *epochs*, обсяг вибірки валідації *validation_split* (по відношенню до тестової вибірки). Також вкажемо розмір батчу *batch_size* та режим відображення *verbose=2*, щоб виводити повну інформацію про навчання.

```
history = model.fit(X_train, Y_train, epochs=50,  
                    validation_split=0.15, batch_size=16, verbose=2)  
  
>>  
  
Epoch 1/50  
91/91 - 0s - loss: 1.4100 - accuracy: 0.2512 - val_loss: 1.4019 -  
val_accuracy: 0.2118  
Epoch 2/50  
91/91 - 0s - loss: 1.3870 - accuracy: 0.2893 - val_loss: 1.3944 -  
val_accuracy: 0.2196  
Epoch 3/50  
91/91 - 0s - loss: 1.3742 - accuracy: 0.3190 - val_loss: 1.3827 -  
val_accuracy: 0.2392  
...  
  
Epoch 48/50  
91/91 - 0s - loss: 0.0863 - accuracy: 0.9785 - val_loss: 0.1605 -  
val_accuracy: 0.9333  
Epoch 49/50  
91/91 - 0s - loss: 0.0853 - accuracy: 0.9758 - val_loss: 0.1527 -  
val_accuracy: 0.9373  
Epoch 50/50  
91/91 - 0s - loss: 0.0834 - accuracy: 0.9827 - val_loss: 0.1499 -  
val_accuracy: 0.9451
```

Оцінюємо якість моделі на тестовій множині. Запишемо результат до змінних *loss* та *accuracy*.

```
loss, accuracy = model.evaluate(X_test, Y_test, verbose=0)
print(loss, accuracy)

>>
0.18817807734012604 0.9200000166893005
```

Виводимо графік з історією навчання (рис. 2.1) для оцінки ефективності навчання.

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()

>>
```

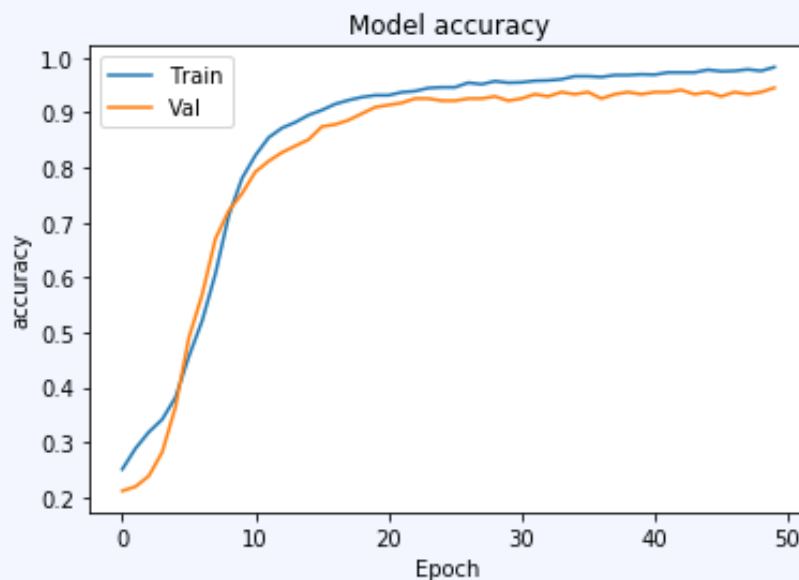


Рис. 2.1. Якість навчання для кожної епохи:

— навчальна множина; — валідаційна множина

Використовуємо модель для класифікації одного зразка з тестової вибірки.

```
# Зразок треба подати як список, тому робимо зріз
sample = X_test[0:1]
```

```
prediction = model.predict(sample)
print(prediction)

>>

[[4.3346232e-30  3.1874181e-24  1.7326867e-05  9.9998271e-01]]
```

Оформимо красиво:

```
score = np.max(prediction)
prediction = np.argmax(prediction)

print(f'Передбачено: {prediction} з достовірністю {score*100:.2f}%',
      f'Повинно бути: {np.argmax(Y_test[0:1])}', sep='\n')

Передбачено: 3 з достовірністю 100.00%
Повинно бути: 3
```

Збережемо модель.

```
model.save('mobile_price.h5')
```


2.1 Завдання для самостійного виконання

Загальні завдання для всіх варіантів:

1. Завантажити набір даних.
2. Вивести заголовок таблиці (перші 5 записів).
3. В якому форматі мітки класів?
4. В якому форматі характеристики об'єктів? Чи є характеристики у вигляді тексту? Якщо так, факторизувати дані в таких стовпцях.
5. Наскільки збалансовані класи?
6. Створити навчальний набір предикторів X та цілей Y . Перетворити мітки класів до формату One hot encoding.
7. Розбити набір даних на навчальну та тестову множини.
8. Стандартизувати предиктори X .
9. Створити модель нейронної мережі прямого розповсюдження. Обрати архітектуру, активаційні функції, оптимізатор, критерії якості та метрики. Якщо даних мало, під час навчання не використовувати валідаційну множину.
10. Навчити нейронну мережу (кілька разів?).
11. Оцінити якість роботи навченої мережі на тестовій множині.
12. Побудувати графік залежності критерію якості (або метрики) від номеру ітерації навчання. Проаналізувати графік і, за потреби, перенавчити мережу, врахувавши отриману інформацію.
13. Подати на вхід навченої мережі довільний зразок із тестової вибірки. За результатами вивести наступне повідомлення: «Передбачено клас {мітка класу} з достовірністю { P }%».

Підказка: для факторизації можна використати наступний код:

```
to_factor = ['назва1', 'назва2']
```

```
X[to_factor] = X[to_factor].apply(lambda col: pd.factorize(col, sort=True)[0])
```

де X – набір даних (предикторів); назва1, назва2 – імена стовпців, дані в яких необхідно факторизувати (замінити текст цілими числами). Кількість таких стовпців може бути різною.

Варіант 1

Набір даних: цитруси (citrus.csv).

Опис даних: В наборі містяться дані про розміри та колір цитрусових фруктів. За цими характеристиками потрібно розпізнати, апельсин це чи грейпфрут. Мітки класів наведені у стовпці «name».

Кількість класів: 2 (апельсин/грейпфрут).

Варіант 2

Набір даних: відмови від мобільного оператора (churn_b.csv).

Опис даних: В наборі містяться дані про користувачів французького оператора Orange Telecom. За цими даними потрібно визначити, чи існує ризик відмови користувача від послуг даного оператора. Мітки класів наведені у стовпці «Churn». Стовпець «State» необхідно видалити із набору даних перед аналізом. Дані в деяких інших стовпцях треба факторизувати.

Кількість класів: 2 (Є ризик (true) / немає ризику (false)).

Варіант 3

Набір даних: клієнти кредитного агентства (clients_b.csv).

Опис даних: В наборі містяться дані про потенційних клієнтів кредитного агентства. За цими даними потрібно визначити, чи існує ризик того, що клієнт не поверне кредит. Мітки класів наведені у стовпці «bad_client_target». Дані в деяких інших стовпцях треба факторизувати.

Кількість класів: 2 (Клієнт надійний (0) / клієнт ненадійний (1)).

Варіант 4

Набір даних: маркетингова кампанія банку (bank_b.csv).

Опис даних: В наборі містяться дані про потенційних вкладників банку. За цими даними потрібно визначити, чи існує ймовірність, що клієнт зробить депозит. Мітки класів наведені у стовпці «у». Дані в деяких інших стовпцях треба факторизувати.

Кількість класів: 2 (Зробить внесок (yes) / не зробить внесок (no)).

Варіант 5

Набір даних: здоров'я ембріону (fetal_health_b.csv).

Опис даних: В наборі містяться дані про біологічні показники ембріонів. За цими даними потрібно визначити стан здоров'я плоду. Мітки класів наведені у стовпці «fetal_health».

Кількість класів: 3 (Нормальне (1) / тривожне (2) / патологічне (3)).

Варіант 6

Набір даних: погода на Різдво в Чикаго (ChicagoWeatherChristmas_new.csv).

Опис даних: В наборі містяться дані про історію погоди в Чикаго на Різдво. За цими даними потрібно визначити, яка ймовірність зустріти сніжне Різдво в залежності від характеристик погоди. Мітки класів наведені у стовпці «White Christmas». Стовпець «Year» необхідно видалити із набору даних перед аналізом

Кількість класів: 3 (Без снігу (FALSE) / Невідомо (Not Defined) / зі снігом (TRUE)).

Варіант 7

Набір даних: махінації з кредитними картками (creditcard_b.csv).

Опис даних: В наборі містяться зашифровані дані про операції з кредитними картками. За цими даними потрібно навчитись виявляти підозрілі транзакції. Мітки класів наведені у стовпці «Class».

Кількість класів: 2 (Безпечна транзакція (0) / небезпечна транзакція (1)).

Варіант 8

Набір даних: доходи населення (income_b.csv).

Опис даних: В наборі містяться дані про різних жителів США. За цими даними потрібно визначити клас, до якого належить людина – багата вона чи ні. Мітки класів наведені у стовпці «income_>50K». Дані в деяких інших стовпцях треба факторизувати.

Кількість класів: 2 (Дохід менше 50K (0) / Дохід більше 50K (1)).

Варіант 9

Набір даних: покупці автомобілів (autocustomer.csv).

Опис даних: В наборі містяться дані про потенційних покупців автомобілів в одному із автосалонів. За цими даними потрібно визначити сегмент, до якого належить покупець. Мітки класів наведені у стовпці «Segmentation». Дані в деяких інших стовпцях треба факторизувати. Стовпець «ID» необхідно видалити із набору даних перед аналізом.

Кількість класів: 4 (A / B / C / D).

Варіант 10

Набір даних: хлопчики та дівчата (gender_classification.csv).

Опис даних: В наборі містяться дані про біометричні показники людей різної статі. За цими даними потрібно визначити стать людини. Мітки класів наведені у стовпці «gender».

Кількість класів: 2 (Чоловік (Male) / Жінка (Female)).

Варіант 11

Набір даних: покемони (pokedex_b.csv).

Опис даних: В наборі містяться дані про покемонів. За цими даними потрібно навчитись визначати, чи є покемон легендарним. Мітки класів наведені у стовпці

«is_legendary». Дані в деяких інших стовпцях треба факторизувати. Стовпці «pokedex_number» та «name» необхідно видалити із набору даних перед аналізом
Кількість класів: 2 (Легендарний (1) / Звичайний (0)).

Варіант 12

Набір даних: музика (music.csv).

Опис даних: В наборі містяться дані про музичні композиції. За цими даними потрібно навчитись визначати, до якого жанру належить композиція. Мітки класів наведені у стовпці «label». Стовпець «filename» необхідно видалити із набору даних перед аналізом

Кількість класів: 10 (hiphop/blues/metal/jazz/rock/country/disco/pop/reggae/classical).

ПРАКТИКУМ 3. ОПТИМІЗАЦІЯ НЕЙРОННИХ МЕРЕЖ

3.1. Загальні відомості

Часто буває необхідно оптимізувати архітектуру та інші параметри створених нейронних мереж. Завдяки вдалому підбору зовнішніх параметрів мережі можна покращити якість її роботи. Серед підходів, які найчастіше використовуються для оптимізації нейромережових моделей, можна виділити наступні:

- Підготовка навчальної вибірки
- Дострокова зупинка навчання (early stopping)
- Оптимізація архітектури
- Нормалізація вихідних даних окремих прошарків мережі (Batch normalization)
- Регуляризація нейронів (Dropout)
- Регуляризація ваг (weight regularizers)
- Підбір оптимальних критеріїв якості та метрик
- Підбір характеристик оптимізаторів
- Підбір характеристик ініціалізації ваг (weight initializers)

Підготовка навчальної вибірки

Розглянемо приклад підготовки даних. Необхідно створити нейронну мережу для класифікації деяких осіб за рівнем доходу. Класів всього два: дохід більше 50 тис. у.о. (`income_>50K == 1`) та менше 50 тис. у.о. (`income_>50K == 0`). Для кожної особи є набір характеристик, за якими пропонується визначити рівень доходів (більше 50 тис. або менше). Всі особливості змінних у даних необхідно врахувати у подальшому аналізі.

Перш за все, імпортуємо необхідні модулі.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
```

```
from keras.models import Sequential
from keras.layers import Dense
```

Наступний код для оптимізації відеокарт (щоб працювало без помилок)

```
import tensorflow as tf
physical_devices = tf.config.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(physical_devices[0], True)
```

Завантажуємо дані та переглянемо заголовок таблиці.

```
data = pd.read_csv("../income.csv")
data.head()
```

```
>>
```

	age	workclass	fnlwgt	education	educational-num	marital-status	\
0	67	Private	366425	Doctorate	16	Divorced	
1	17	Private	244602	12th	8	Never-married	
2	31	Private	174201	Bachelors	13	Married-civ-spouse	
3	58	State-gov	110199	7th-8th	4	Married-civ-spouse	
4	25	State-gov	149248	Some-college	10	Never-married	

	occupation	relationship	race	gender	capital-gain	capital-loss	\
0	Exec-managerial	Not-in-family	White	Male	99999	0	
1	Other-service	Own-child	White	Male	0	0	
2	Exec-managerial	Husband	White	Male	0	0	
3	Transport-moving	Husband	White	Male	0	0	
4	Other-service	Not-in-family	Black	Male	0	0	

	hours-per-week	native-country	income_>50K
0	60	United-States	1
1	15	United-States	0
2	40	United-States	1
3	40	United-States	0
4	40	United-States	0

Як бачимо, характеристики об'єктів в наборі даних можуть бути як числовими (загальний капітал, кількість робочих годин за тиждень, рівень витрат тощо), так і рядковими (професія, сімейний стан, стать тощо).

Переглянемо загальну інформацію про набір даних.

```
data.info()
```

```
>>
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 43957 entries, 0 to 43956
Data columns (total 15 columns):
#   Column                Non-Null Count  Dtype
---  -
0   age                   43957 non-null  int64
1   workclass             41459 non-null  object
2   fnlwgt               43957 non-null  int64
3   education            43957 non-null  object
4   educational-num      43957 non-null  int64
5   marital-status       43957 non-null  object
6   occupation           41451 non-null  object
7   relationship         43957 non-null  object
8   race                 43957 non-null  object
9   gender               43957 non-null  object
10  capital-gain         43957 non-null  int64
11  capital-loss         43957 non-null  int64
12  hours-per-week      43957 non-null  int64
13  native-country      43194 non-null  object
14  income_>50K        43957 non-null  int64
dtypes: int64(7), object(8)
memory usage: 5.0+ MB

```

Бачимо, що всього в наборі є дані про 43957 об'єктів (осіб). Окрім того, якщо подивимось уважно, то деякі об'єкти мають пропущені характеристики (число ненульових записів в стовпці не співпадає з загальною кількістю об'єктів).

Відповідно, дізнаємось кількість пропущених значень для кожної характеристики.

```

data.isnull().sum()

>>
age                0
workclass          2498
fnlwgt             0
education          0
educational-num    0
marital-status     0
occupation         2506
relationship       0
race               0
gender             0
capital-gain       0
capital-loss       0

```



```
hours-per-week      0
native-country      763
income_>50K         0
```

Як бачимо, пропущені значення є в стовпцях *workclass*, *occupation* та *native-country*. Оскільки кожен рядок таблиці являє собою окрему людину, ми не можемо просто взяти і замінити пропущені дані нулями або середніми значеннями. Тому єдиний варіант - прибрати з таблиці записи із пропущеними даними.

```
clear_data = data.dropna(subset=["workclass", "occupation", "native-
country"])
clear_data.isnull().sum()

>>

age                0
workclass          0
fnlwgt            0
education          0
educational-num    0
marital-status     0
occupation         0
relationship       0
race              0
gender             0
capital-gain       0
capital-loss       0
hours-per-week     0
native-country     0
income_>50K        0
dtype: int64
```

Тепер пропущених даних немає - можна продовжувати роботу.

Перевіряємо, наскільки збалансована кількість зразків кожного класу. В ідеальному випадку, доля зразків кожного класу має бути приблизно однаковою.

```
clear_data['income_>50K'].value_counts(normalize=True)

>>

0    0.752204
1    0.247796
```

Як бачимо, кількість об'єктів кожного класу незбалансована. Це може призвести до того, що мережа під час навчання значно більше "уваги" приділятиме об'єктам того класу, зразків якого більше. За такої незбалансованості навіть може виникнути ситуація, коли до навчальної вибірки не потрапить жодного зразка класу, об'єктів якого менше. З цим треба щось робити.

Балансування вибірки

Існують різні підходи до виправлення балансу між класами. Наприклад, можна ввести так звану "вагу класу" - коефіцієнти для кожного класу, які визначатимуть ціну помилки. Ціна помилки для класу, зразків якого в навчальних даних менше, буде більшою. Відповідно, такі зразки будуть сильніше впливати на критерій якості, що змусить мережу більше уваги приділяти саме цим зразкам. Однак, це призведе до розширення діапазону значень функції помилки. Деякі оптимізатори, величина кроку яких залежить від величини градієнту (наприклад, стандартний SGD), можуть спрацювати некоректно.

Детальніше з використанням зважених класів можна ознайомитись за посиланням:

- https://www.tensorflow.org/tutorials/structured_data/imbalanced_data#calculate_class_weights

У подальшому ми будемо використовувати більш прості і очевидні підходи, такі як:

- штучне збільшення кількості зразків (oversampling) класу, об'єктів якого менше у вибірці;
- штучне зменшення (undersampling) кількості зразків класу, об'єктів якого більше у вибірці.

Коли який підхід використовувати?

- Якщо загальна кількість зразків "пригніченого" класу невелика (до 200-300), то використовуємо oversampling - копіюємо існуючі зразки стільки разів, скільки потрібно для зрівняння з "домінуючим" класом.
- Якщо кількість зразків "пригніченого" класу достатня (від 300 і більше), можна видалити потрібну кількість випадкових об'єктів "домінуючого" класу для виправлення балансу.

Подивимось, скільки у нас зразків кожного класу в абсолютній величині.

```
clear_data['income_>50K'].value_counts()

>>
0      30635
1      10092
```

Зразків класу 1 в три рази менше, ніж класу 0, але достатньо для застосування підходу undersampling. Зробимо це.

Створимо два набори даних (по одному для кожного класу). В перший набір запишемо лише зразки класу 0, в другий - класу 1. Під час формування набору зразків класу 0 вкажемо, що з початкового набору необхідно взяти випадкові 10500 об'єктів, що приблизно буде дорівнювати кількості зразків у класі 1. Потім об'єднаємо ці два набори в один.

```
# Undersampling
class0 = clear_data[clear_data['income_>50K']==0].sample(n=10500)
class1 = clear_data[clear_data['income_>50K']==1]

balanced_data = pd.concat([class0,class1])
balanced_data['income_>50K'].value_counts(normalize=True)

>>
0      0.509907
1      0.490093
```

Тепер кількість зразків в обох класах збалансована.

Якби ми використовували підхід oversampling, потрібно було б виконати аналогічні операції, але копіювати набір зразків класу 1 три рази. Приклад коду:

```
# Oversampling
'''
class0 = clear_data[clear_data['income_>50K']==0]
class1 = clear_data[clear_data['income_>50K']==1]
class1 = pd.concat([class1]*3)

balanced_data = pd.concat([class0,class1])
balanced_data['income_>50K'].value_counts(normalize=True)
'''

>>

0    0.502947
1    0.497053
```

Факторизація

Як ми вже з'ясували, в наборі даних деякі змінні представлені у вигляді рядків. Подивимось знову на заголовок таблиці.

```
balanced_data.head()

>>
```

	age	workclass	fnlwgt	education	educational-num	\
224	60	Private	167670	Bachelors	13	
9419	59	Private	113838	Bachelors	13	
36762	43	Private	233851	Bachelors	13	
23171	27	Self-emp-not-inc	334132	Assoc-acdm	12	
16259	34	Private	299383	HS-grad	9	

	marital-status	occupation	relationship	race	gender	\
224	Married-civ-spouse	Prof-specialty	Husband	White	Male	
9419	Widowed	Prof-specialty	Not-in-family	White	Female	
36762	Divorced	Adm-clerical	Not-in-family	White	Female	
23171	Never-married	Prof-specialty	Not-in-family	White	Female	
16259	Never-married	Craft-repair	Not-in-family	Black	Male	

	capital-gain	capital-loss	hours-per-week	native-country	income_>50K
224	0	0	35	United-States	0
9419	4650	0	37	United-States	0
36762	0	0	40	United-States	0
23171	0	0	78	United-States	0
16259	0	0	40	United-States	0

Щоб перетворити рядкові дані на числові, використовується операція **факторизації**. Факторизація замінює однакові слова або словосполучення відповідними чисельними значеннями. Таким чином, характеристика об'єкту стає *фактором*, який може бути врахованим математично.

В Pandas з цією метою використовується функція *factorize()*, яку необхідно застосувати для кожного стовпця з текстовими змінними. В нашому випадку, факторизувати необхідно дані у стовпцях *workclass*, *education*, *marital-status*, *occupation*, *relationship*, *race*, *gender*, *native-country*. Зробимо це за допомогою наступного коду.

```
to_factor = ['workclass', 'education', 'marital-status', 'occupation',  
'relationship', 'race', 'gender', 'native-country']
```

```
factor_data = balanced_data.copy()  
factor_data[to_factor] = factor_data[to_factor].apply(lambda col:  
pd.factorize(col, sort=True)[0])  
factor_data.head()
```

```
>>
```

	age	workclass	fnlwgt	education	educational-num	marital-status	\
224	60	2	167670	9	13	2	
9419	59	2	113838	9	13	6	
36762	43	2	233851	9	13	0	
23171	27	4	334132	7	12	4	
16259	34	2	299383	11	9	4	

	occupation	relationship	race	gender	capital-gain	capital-loss	\
224	9	0	4	1	0	0	
9419	9	1	4	0	4650	0	
36762	0	1	4	0	0	0	
23171	9	1	4	0	0	0	
16259	2	1	2	1	0	0	

	hours-per-week	native-country	income_>50K
224	35	37	0
9419	37	37	0
36762	40	37	0
23171	78	37	0
16259	40	37	0

Однак, якщо ми подивимось уважніше, в стовпці *educational-num* вже містяться факторизовані дані про освіту особи. Відповідно, стовпець *educational-num* та факторизований нами стовпець *education* по суті дублюють один одного. Іншими словами, вони є **колінеарними**. Тому один із цих стовпців можна взагалі прибрати з даних.

```
final_data = factor_data.drop('education', axis = 1)
final_data.head()
```

```
>>
```

	age	workclass	fnlwgt	educational-num	marital-status	occupation	\
224	60	2	167670	13	2	9	
9419	59	2	113838	13	6	9	
36762	43	2	233851	13	0	0	
23171	27	4	334132	12	4	9	
16259	34	2	299383	9	4	2	

	relationship	race	gender	capital-gain	capital-loss	hours-per-week	\
224	0	4	1	0	0	35	
9419	1	4	0	4650	0	37	
36762	1	4	0	0	0	40	
23171	1	4	0	0	0	78	
16259	1	2	1	0	0	40	

	native-country	income_>50K
224	37	0
9419	37	0
36762	37	0
23171	37	0
16259	37	0

Класи збалансовані, текстові змінні факторизовані, пропущені значення та колінеарні змінні видалені. Тепер можна створити і навчити нейронну мережу за стандартною процедурою.

```
Y = pd.get_dummies(final_data['income_>50K'], prefix='Class')
X = final_data.drop(['income_>50K'], axis = 1)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
                                                    test_size=0.15)
```

```

mean = X_train.mean(axis=0)
std = X_train.std(axis=0)

X_train -= mean
X_train /= std
X_test -= mean
X_test /= std

model = Sequential()
model.add(Dense(10, input_dim=X_train.shape[1], activation="relu"))
model.add(Dense(2, activation="softmax"))
model.compile(optimizer='adam', loss='binary_crossentropy',
              metrics=['accuracy'])

history = model.fit(X_train, Y_train, epochs=30,
                    validation_split=0.15, batch_size=20)

loss, accuracy = model.evaluate(X_test, Y_test, verbose=0)
print(loss, accuracy)

>>

Train on 14877 samples, validate on 2626 samples
Epoch 1/30
14877/14877 [=====] - 3s 182us/step - loss:
0.5293 - accuracy: 0.7364 - val_loss: 0.4578 - val_accuracy: 0.7768
Epoch 2/30
14877/14877 [=====] - 2s 155us/step - loss:
0.4382 - accuracy: 0.7872 - val_loss: 0.4291 - val_accuracy: 0.8012
Epoch 3/30
14877/14877 [=====] - 2s 145us/step - loss:
0.4148 - accuracy: 0.8049 - val_loss: 0.4134 - val_accuracy: 0.8111
...
Epoch 28/30
14877/14877 [=====] - 2s 142us/step - loss:
0.3874 - accuracy: 0.8182 - val_loss: 0.4002 - val_accuracy: 0.8199
Epoch 29/30
14877/14877 [=====] - 2s 147us/step - loss:
0.3875 - accuracy: 0.8190 - val_loss: 0.3992 - val_accuracy: 0.8161
Epoch 30/30
14877/14877 [=====] - 2s 146us/step - loss:
0.3876 - accuracy: 0.8182 - val_loss: 0.3986 - val_accuracy: 0.8184
0.3751888842828145 0.827128529548645

```

Регуляризація

Регуляризація – комплекс дій, спрямованих на уникнення перенавчання мережі. У випадку використання регуляризації модель навмисно спрощують під час навчання або обмежують її параметри, накладаючи деякі штрафи за перенавчання. Це дозволяє покращити якість навчання та пришвидшити збіжність оптимізатора. Як наслідок, мережа буде швидше навчатись та більш якісно апроксимувати дані. Особливо помітним вплив регуляризації є у навчанні глибинний мереж з десятками і сотнями прошарків, тоді як більш поверхневі моделі самі по собі є достатньо простими для уникнення перенавчання. Однак, іноді все ж вдається на декілька відсотків підвищити показники якості роботи нейромережі.

Для розгляду різноманітних підходів до регуляризації, повернемося до набору даних з класифікацією мобільних телефонів за ціновими діапазонами.

```
data = pd.read_csv("../phones_price.csv")
data.head()
```

```
>>
```

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	\
0	842	0	2.2	0	1	0	7	0.6	
1	1021	1	0.5	1	0	1	53	0.7	
2	563	1	0.5	1	2	1	41	0.9	
3	615	1	2.5	0	0	0	10	0.8	
4	1821	1	1.2	0	13	1	44	0.6	

	mobile_wt	n_cores	...	px_height	px_width	ram	sc_h	sc_w	talk_time	\
0	188	2	...	20	756	2549	9	7	19	
1	136	3	...	905	1988	2631	17	3	7	
2	145	5	...	1263	1716	2603	11	2	9	
3	131	6	...	1216	1786	2769	16	8	11	
4	141	2	...	1208	1212	1411	8	2	15	

	three_g	touch_screen	wifi	price_range
0	0	0	1	1
1	1	1	0	2
2	1	1	0	2
3	1	0	0	2
4	1	1	0	1

```
[5 rows x 21 columns]
```



```

X = data.drop('price_range', axis = 1)
Y = pd.get_dummies(data.price_range, prefix='Price range')

X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
                                                    test_size=0.15)

mean = X_train.mean(axis=0)
std = X_train.std(axis=0)

X_train -= mean
X_train /= std
X_test -= mean
X_test /= std

model = Sequential()
model.add(Dense(10, input_dim=X_train.shape[1], activation="relu"))
model.add(Dense(5, activation="relu"))
model.add(Dense(4, activation="softmax"))

model.compile(optimizer='adam', loss='categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(X_train, Y_train, epochs=50,
                   validation_split=0.15, batch_size=16)
loss, accuracy = model.evaluate(X_test, Y_test, verbose=0)
print(loss, accuracy)

>>

Train on 1445 samples, validate on 255 samples
Epoch 1/50
1445/1445 [=====] - 0s 257us/step - loss:
1.3842 - accuracy: 0.2803 - val_loss: 1.3206 - val_accuracy: 0.3529
Epoch 2/50
1445/1445 [=====] - 0s 185us/step - loss:
1.3162 - accuracy: 0.3315 - val_loss: 1.2721 - val_accuracy: 0.3882
Epoch 3/50
1445/1445 [=====] - 0s 185us/step - loss:
1.2525 - accuracy: 0.4249 - val_loss: 1.2161 - val_accuracy: 0.4627
...

Epoch 48/50
1445/1445 [=====] - 0s 183us/step - loss:
0.0749 - accuracy: 0.9827 - val_loss: 0.1879 - val_accuracy: 0.9176
Epoch 49/50
1445/1445 [=====] - 0s 184us/step - loss:
0.0721 - accuracy: 0.9841 - val_loss: 0.1836 - val_accuracy: 0.9216
Epoch 50/50

```

```
1445/1445 [=====] - 0s 183us/step - loss: 0.0702 - accuracy: 0.9862 - val_loss: 0.1820 - val_accuracy: 0.9137
```

```
0.22537615448236464 0.8866666555404663
```

Виводимо графік з історією навчання (рис. 3.1) для оцінки ефективності навчання.

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()

>>
```

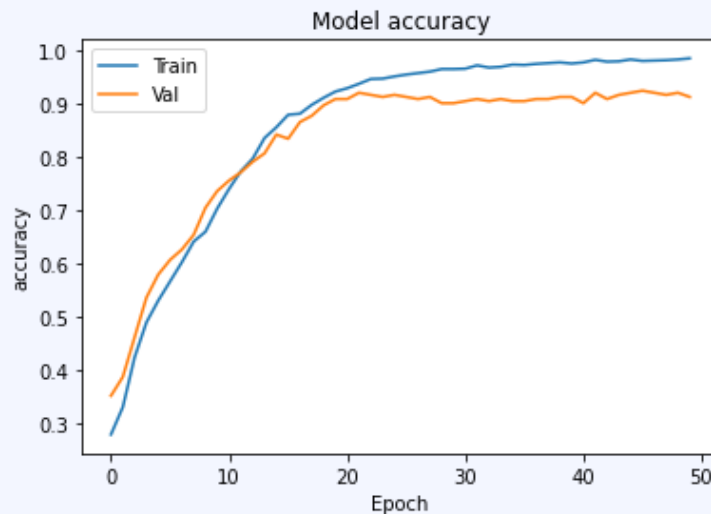


Рис. 3.1. Якість навчання для кожної епохи:

— навчальна множина; — валідаційна множина

Як бачимо, приблизно після 20 епохи почалося перенавчання мережі: ассигасу на валідаційній множині почала знижуватись (хоча потім і відновила своє середнє значення). Ассигасу на тестовій множині склала 88,6%. Відповідно, спробуємо застосувати різні підходи до регуляризації для виправлення даної ситуації і покращення якості роботи мережі.

Дострокова зупинка навчання (early stopping)

Keras дозволяє відслідковувати величину зміни критерію якості або метрики під час навчання. Якщо критерій якості не буде покращуватись декілька епох поспіль, існує можливість автоматично зупинити навчання. Це не дозволить мережі продовжувати підлаштовуватись під особливості навчальної вибірки, коли потрібна якість уже досягнута.

- Детальніше: https://keras.io/api/callbacks/early_stopping/

Як видно з документації, для реалізації цього методу використовується клас *EarlyStopping*. Ми повинні створити його екземпляр та вказати посилання на цей об'єкт в якості функції зворотного виклику для метода *model.fit()*. Основні параметри *EarlyStopping*, які нам знадобляться:

- *monitor* - критерій якості або метрика, яку потрібно відслідковувати
- *patience* - кількість епох без покращення результату, після яких навчання буде зупинено.

Використаємо цей підхід. Відслідковувати будемо метрику Ассурасу на валідаційній множині.

```
import tensorflow as tf
early = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy',
                                         patience=5)

model2 = Sequential()
model2.add(Dense(10, input_dim=X_train.shape[1], activation="relu"))
model2.add(Dense(5, activation="relu"))
model2.add(Dense(4, activation="softmax"))

model2.compile(optimizer='adam', loss='categorical_crossentropy',
               metrics=['accuracy'])

# Використовуємо callbacks=[early]
history2 = model2.fit(X_train, Y_train, epochs=50,
                     validation_split=0.15, batch_size=16, callbacks=[early])

loss, accuracy = model2.evaluate(X_test, Y_test, verbose=0)
print(loss, accuracy)
```

```

>>
Train on 1445 samples, validate on 255 samples
Epoch 1/50
1445/1445 [=====] - 0s 255us/step - loss:
1.4545 - accuracy: 0.2740 - val_loss: 1.3515 - val_accuracy: 0.3294
Epoch 2/50
1445/1445 [=====] - 0s 187us/step - loss:
1.3664 - accuracy: 0.3370 - val_loss: 1.3036 - val_accuracy: 0.3490
Epoch 3/50
1445/1445 [=====] - 0s 180us/step - loss:
1.3032 - accuracy: 0.3882 - val_loss: 1.2536 - val_accuracy: 0.3647
Epoch 4/50
...
Epoch 27/50
1445/1445 [=====] - 0s 180us/step - loss:
0.1442 - accuracy: 0.9599 - val_loss: 0.2200 - val_accuracy: 0.9059
Epoch 28/50
1445/1445 [=====] - 0s 181us/step - loss:
0.1379 - accuracy: 0.9640 - val_loss: 0.2132 - val_accuracy: 0.9059
Epoch 29/50
1445/1445 [=====] - 0s 177us/step - loss:
0.1307 - accuracy: 0.9654 - val_loss: 0.2069 - val_accuracy: 0.9098

0.21375454366207122 0.9133333563804626

```

Як бачимо, спрацював механізм early stopping, і навчання автоматично перервалось на 29 епосі. На тестовій множині ми отримали долю правильних відповідей 91,3%. Подивимось на графік навчання (рис. 3.2).

```

plt.plot(history2.history['accuracy'])
plt.plot(history2.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()

>>

```

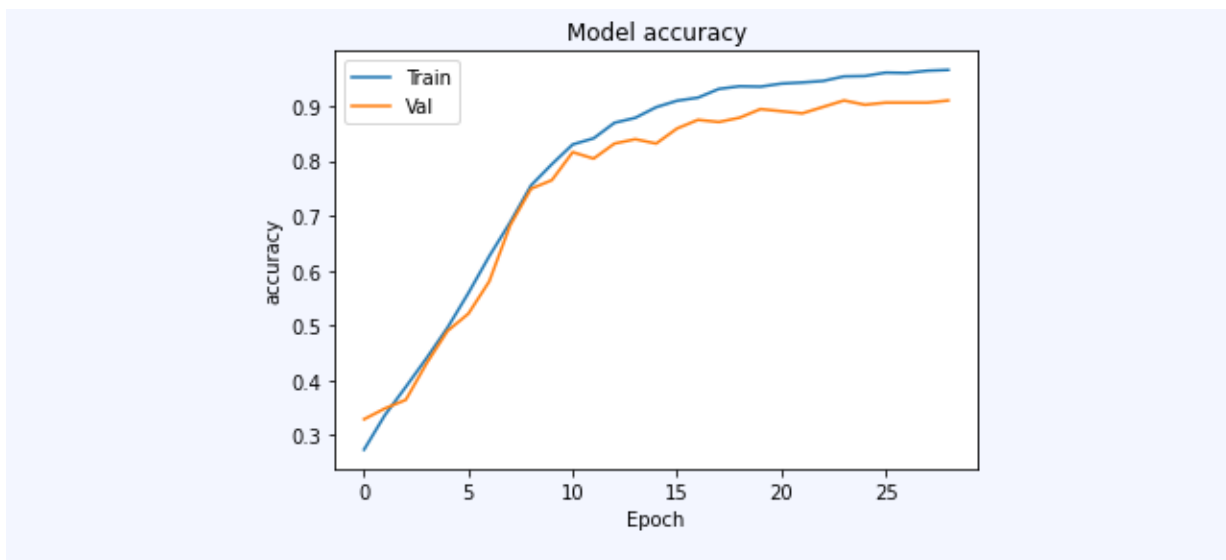


Рис. 3.2. Якість навчання для кожної епохи при застосуванні механізму дострокової зупинки навчання:

— навчальна множина; — валідаційна множина

Перенавчання немає.

Оптимізація архітектури

Іноді початково обрана архітектура мережі є занадто складною, що також може призводити до перенавчання. В такому випадку, доцільно зменшити складність мережі, відповідним чином змінивши кількість прошарків або нейронів у них.

Видалимо другий прихований прошарок з нашої мережі, а також зменшимо кількість нейронів у першому до 6. Цього буде достатньо для вирішення поставленого завдання.

```
model3 = Sequential()
model3.add(Dense(6, input_dim=X_train.shape[1], activation="relu"))
model3.add(Dense(4, activation="softmax"))

model3.compile(optimizer='adam', loss='categorical_crossentropy',
               metrics=['accuracy'])

history3 = model3.fit(X_train, Y_train, epochs=50,
                     validation_split=0.15, batch_size=16)
```

```

loss, accuracy = model3.evaluate(X_test, Y_test, verbose=0)
print(loss, accuracy)

>>

Train on 1445 samples, validate on 255 samples
Epoch 1/50
1445/1445 [=====] - 0s 237us/step - loss:
1.5003 - accuracy: 0.2948 - val_loss: 1.5053 - val_accuracy: 0.2667
Epoch 2/50
1445/1445 [=====] - 0s 178us/step - loss:
1.4122 - accuracy: 0.3260 - val_loss: 1.4386 - val_accuracy: 0.3020
Epoch 3/50
1445/1445 [=====] - 0s 165us/step - loss:
1.3521 - accuracy: 0.3488 - val_loss: 1.3864 - val_accuracy: 0.3412
...

Epoch 48/50
1445/1445 [=====] - 0s 156us/step - loss:
0.1550 - accuracy: 0.9744 - val_loss: 0.1934 - val_accuracy: 0.9647
Epoch 49/50
1445/1445 [=====] - 0s 156us/step - loss:
0.1515 - accuracy: 0.9723 - val_loss: 0.1897 - val_accuracy: 0.9647
Epoch 50/50
1445/1445 [=====] - 0s 157us/step - loss:
0.1475 - accuracy: 0.9730 - val_loss: 0.1868 - val_accuracy: 0.9569
0.17638297021389007 0.949999988079071

```

Виводимо графік з історією навчання (рис. 3.3) для оцінки ефективності навчання.

```

plt.plot(history3.history['accuracy'])
plt.plot(history3.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()

>>

```

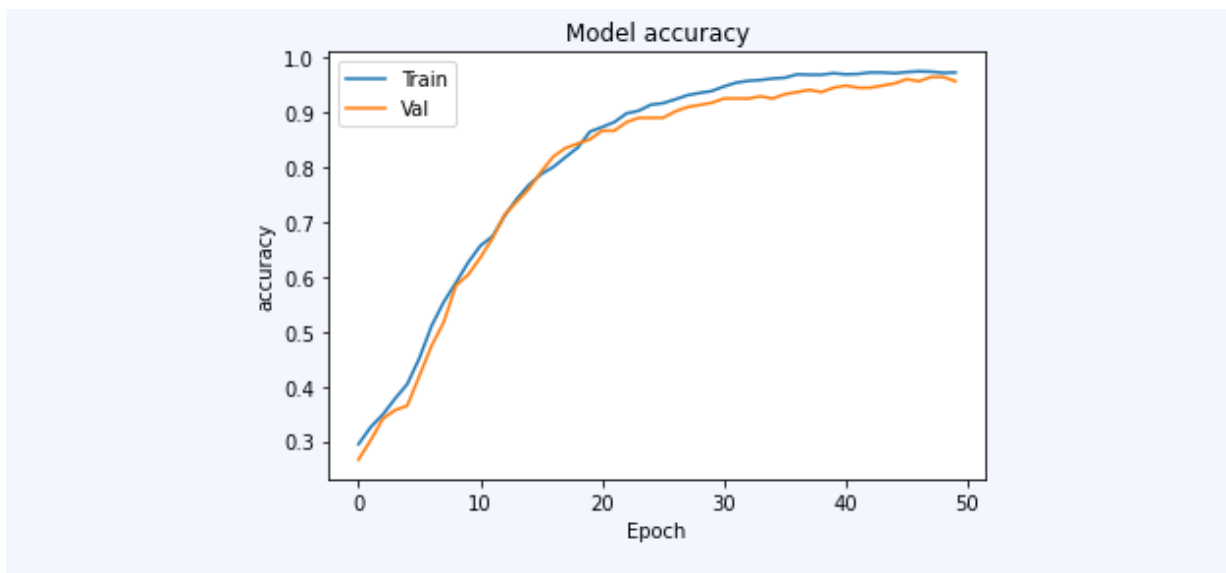


Рис. 3.3. Якість навчання для кожної епохи при оптимізації архітектури:

— навчальна множина; — валідаційна множина

Як бачимо, для навчання мережі знадобилося більше ітерацій, але водночас якість суттєво зросла до значення accuracy майже 94,5% на тестовій множині. Перенавчання немає.

Batch normalization

Використання прошарку Batch normalization дозволяє пришвидшити навчання та полегшити мережі пошук необхідних ознак. Математичне обґрунтування необхідності використання такого прошарку існує, але його ефективність на практиці часто ставиться під сумнів. Втім, Batch normalization все одно застосовують.

➤ Документація в Keras:

https://keras.io/api/layers/normalization_layers/batch_normalization/

Прошарок Batch normalization нормалізує вхідні дані для наступного прошарку. Для цього значення по кожному із входів прошарку стандартизуються методом приведення їх до нульового середнього та одиничної дисперсії. Для обчислення необхідних величин використовуються дані за один батч (міні-батч).

Використаємо Batch normalization у комбінації зі спрощеною архітектурою.

```

from keras.layers import BatchNormalization

model4 = Sequential()
model4.add(Dense(6, input_dim=X_train.shape[1], activation="relu"))
model4.add(BatchNormalization())
model4.add(Dense(4, activation="softmax"))

model4.compile(optimizer='adam', loss='categorical_crossentropy',
               metrics=['accuracy'])

history4 = model4.fit(X_train, Y_train, epochs=50,
                     validation_split=0.15, batch_size=16)

loss, accuracy = model4.evaluate(X_test, Y_test, verbose=0)
print(loss, accuracy)

>>

Train on 1445 samples, validate on 255 samples
Epoch 1/50
1445/1445 [=====] - 1s 370us/step - loss:
1.6787 - accuracy: 0.2325 - val_loss: 1.4582 - val_accuracy: 0.2980
Epoch 2/50
1445/1445 [=====] - 0s 231us/step - loss:
1.4983 - accuracy: 0.2865 - val_loss: 1.3770 - val_accuracy: 0.3765
Epoch 3/50
1445/1445 [=====] - 0s 228us/step - loss:
1.3761 - accuracy: 0.3315 - val_loss: 1.2973 - val_accuracy: 0.4000
...

Epoch 48/50
1445/1445 [=====] - 0s 227us/step - loss:
0.3119 - accuracy: 0.8803 - val_loss: 0.2209 - val_accuracy: 0.9333
Epoch 49/50
1445/1445 [=====] - 0s 232us/step - loss:
0.2911 - accuracy: 0.8858 - val_loss: 0.2117 - val_accuracy: 0.9451
Epoch 50/50
1445/1445 [=====] - 0s 227us/step - loss:
0.3167 - accuracy: 0.8775 - val_loss: 0.2071 - val_accuracy: 0.9490
0.19396755476792654 0.9566666483879089

```

Виводимо графік з історією навчання (рис. 3.4) для оцінки ефективності навчання.

```

plt.plot(history4.history['accuracy'])
plt.plot(history4.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('accuracy')

```



```
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()

>>
```

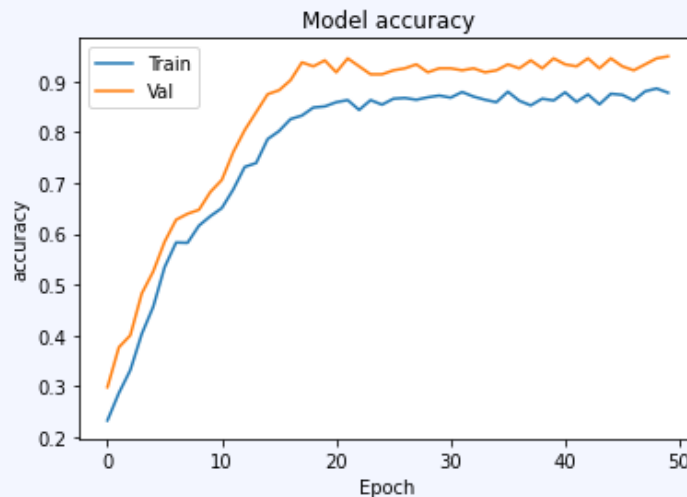


Рис. 3.4. Якість навчання для кожної епохи при застосуванні Batch normalization:

— навчальна множина; — валідаційна множина

Значення accuracy склало 95,6% на тестовій множині. Перенавчання немає.

На графіку можна помітити цікавий ефект - точність на валідаційній множині вище, ніж на навчальній. Це нормально, і вказує на те, що під час навчання використовувалась регуляризація.

Dropout

Прошарок Dropout дозволяє уникнути перенавчання шляхом відключення певної кількості випадкових нейронів попереднього прошарку. Під "відключенням" мається на увазі заміна їх вихідних значень на 0. Таким чином, під час навчання мережа гірше запам'ятовує конкретні навчальні зразки та краще їх узагальнює, виявляючи найбільш значущі взаємозв'язки у даних.

➤ Документація: https://keras.io/api/layers/regularization_layers/dropout/

Головним параметром прошарку Dropout є *rate* - доля нейронів, які необхідно відключити.

Dropout зазвичай застосовується у випадках, коли прошарок містить декілька десятків і більше нейронів. Однак, з навчальною метою, використаємо його для нашої моделі.

```
from keras.layers import Dropout

model5 = Sequential()
model5.add(Dense(6, input_dim=X_train.shape[1], activation="relu"))
model5.add(Dropout(0.2))
model5.add(Dense(4, activation="softmax"))

model5.compile(optimizer='adam', loss='categorical_crossentropy',
               metrics=['accuracy'])

history5 = model5.fit(X_train, Y_train, epochs=50,
                     validation_split=0.15, batch_size=16)

loss, accuracy = model5.evaluate(X_test, Y_test, verbose=0)
print(loss, accuracy)

>>

Train on 1445 samples, validate on 255 samples
Epoch 1/50
1445/1445 [=====] - 0s 287us/step - loss:
1.6374 - accuracy: 0.2360 - val_loss: 1.5357 - val_accuracy: 0.2471
Epoch 2/50
1445/1445 [=====] - 0s 203us/step - loss:
1.4873 - accuracy: 0.2879 - val_loss: 1.4358 - val_accuracy: 0.2706
Epoch 3/50
1445/1445 [=====] - 0s 171us/step - loss:
1.3809 - accuracy: 0.3384 - val_loss: 1.3656 - val_accuracy: 0.3373
...

Epoch 48/50
1445/1445 [=====] - 0s 167us/step - loss:
0.3666 - accuracy: 0.8526 - val_loss: 0.2306 - val_accuracy: 0.9490
Epoch 49/50
1445/1445 [=====] - 0s 167us/step - loss:
0.3395 - accuracy: 0.8713 - val_loss: 0.2231 - val_accuracy: 0.9569
Epoch 50/50
1445/1445 [=====] - 0s 167us/step - loss:
0.3757 - accuracy: 0.8457 - val_loss: 0.2222 - val_accuracy: 0.9608
0.240070184469223 0.9466666579246521
```

Виводимо графік з історією навчання (рис. 3.5) для оцінки ефективності навчання.

```
plt.plot(history5.history['accuracy'])
plt.plot(history5.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()
```

>>

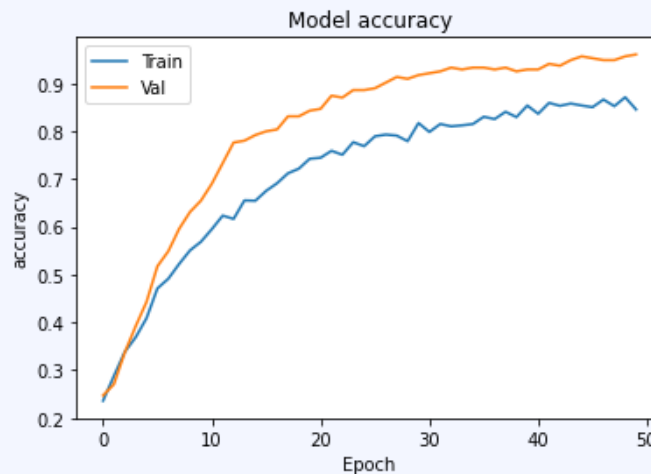


Рис. 3.5. Якість навчання для кожної епохи при застосуванні прошарку Dropout:

— навчальна множина; — валідаційна множина

Значення асигасу склало 94,6% на тестовій множині. Перенавчання немає. На графіку знову видно ефект регуляризації - точність на валідаційній множині вище, ніж на навчальній. У випадку Dropout це легко пояснити - під час навчання деякі нейрони вимикаються, і модель працює у спрощеному вигляді. Але під час валідації прошарок Dropout не використовується, тому нейронна мережа виконує задачу зі своєю повною потужністю, що призводить до зменшення помилки.

Регуляризація ваг (weight regularizers)

Встановлено, що великі значення ваг призводять до перенавчання. Це пояснюється тим, що такі значення "перетягують на себе увагу" мережі, водночас інші ознаки потенційно важливі ознаки будуть ігноруватись. Чим більші значення

ваг, тим більш нелінійною буде розділяюча функція (правило), яку шукає мережа під час навчання. Тут можна провести аналогію з апроксимацією поліномом - чим вище степінь полінома, тим більш нелінійною є апроксимуюча функція.

Щоб уникнути перенавчання внаслідок завеликих значень деяких ваг, використовуються так звані регуляризаційні доданки до значення критерію якості. Основні їх види такі:

- L1-регуляризація - під час розрахунку функції втрат до її значення додається зважена сума модулів синаптичних ваг мережі.
- L2-регуляризація - те саме, але додається зважена сума квадратів ваг мережі (використовується частіше).
- L1_L2-регуляризація - комбінація перших двох.

Додавання регуляризаційного доданку до значення функції втрат грає роль штрафу. Чим більше значення цього доданку (чим вище сума ваг), тим вищим буде значення функції втрат. Відповідно, щоб це значення зменшити, оптимізатор буде зменшувати ваги.

Регуляризація може застосовуватись як до основних ваг (kernel regularizer), так і для ваг зміщень (вільних членів) мережі (bias regularizer).

- Детальніше в документації: <https://keras.io/api/layers/regularizers/>

Застосуємо L2-регуляризацію для нашої моделі. Для цього необхідний вид регуляризації вказується як аргумент під час створення прошарку.

```
model6 = Sequential()
model6.add(Dense(6, input_dim=X_train.shape[1], activation="relu",
kernel_regularizer='l2'))
model6.add(Dense(4, activation="softmax", kernel_regularizer='l2'))

model6.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

history6 = model6.fit(X_train, Y_train, epochs=50,
validation_split=0.15, batch_size=16)
```

```

loss, accuracy = model6.evaluate(X_test, Y_test, verbose=0)
print(loss, accuracy)

>>

Train on 1445 samples, validate on 255 samples
Epoch 1/50
1445/1445 [=====] - 0s 243us/step - loss:
1.7506 - accuracy: 0.2201 - val_loss: 1.6760 - val_accuracy: 0.2549
Epoch 2/50
1445/1445 [=====] - 0s 169us/step - loss:
1.6007 - accuracy: 0.2464 - val_loss: 1.5548 - val_accuracy: 0.2902
Epoch 3/50
1445/1445 [=====] - 0s 158us/step - loss:
1.5076 - accuracy: 0.2830 - val_loss: 1.4683 - val_accuracy: 0.3333
...

Epoch 48/50
1445/1445 [=====] - 0s 157us/step - loss:
0.5217 - accuracy: 0.9716 - val_loss: 0.5279 - val_accuracy: 0.9608
Epoch 49/50
1445/1445 [=====] - 0s 157us/step - loss:
0.5211 - accuracy: 0.9696 - val_loss: 0.5286 - val_accuracy: 0.9569
Epoch 50/50
1445/1445 [=====] - 0s 157us/step - loss:
0.5205 - accuracy: 0.9702 - val_loss: 0.5283 - val_accuracy: 0.9608
0.5368138917287191 0.949999988079071

```

Виводимо графік з історією навчання (рис. 3.6) для оцінки ефективності навчання.

```

plt.plot(history6.history['accuracy'])
plt.plot(history6.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()

>>

```

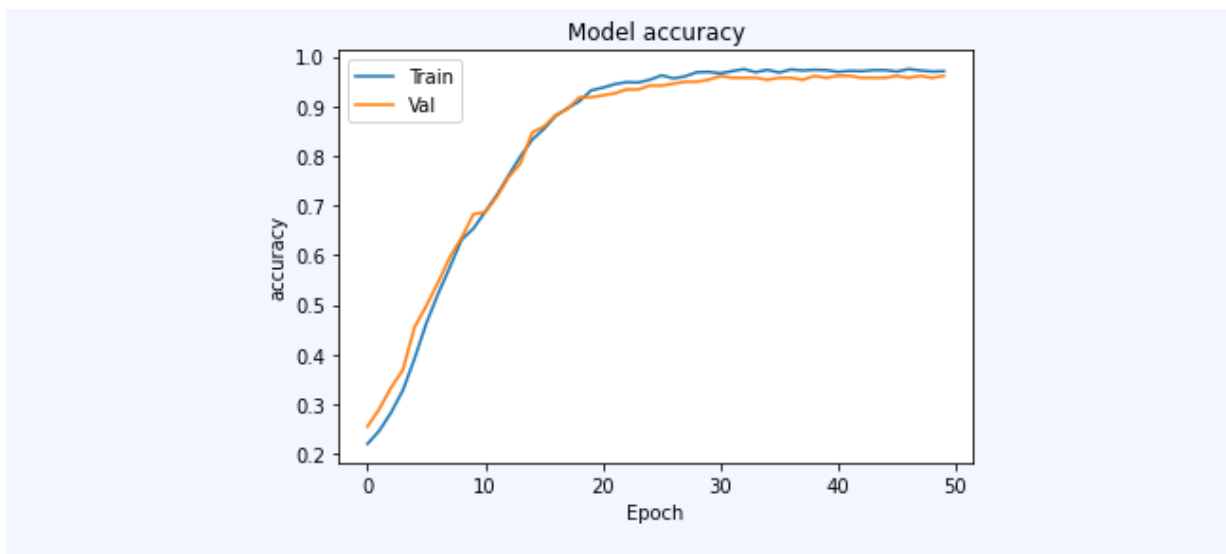


Рис. 3.6. Якість навчання для кожної епохи при застосуванні регуляризації ваг:

— навчальна множина; — валідаційна множина

Значення accuracy склало 94,9% на тестовій множині. Перенавчання немає.

Підбір оптимальних критеріїв якості та метрик

Обраний критерій якості напряму впливає на якість роботи нейронної мережі. Адже саме критерій якості оптимізується в ході виконання алгоритму зворотношопроширення помилки. Деякі критерії якості призначені для вирішення лише спеціалізованих задач, деякі є універсальними. Важливо розуміти, який критерій якості потрібно використати в конкретній ситуації.

- Доступні критерії якості в Keras: <https://keras.io/api/losses/>

Всього критерії якості в Keras діляться на три великі групи:

- Probabilistic losses – функції втрат для вирішення задач класифікації.
- Regression losses – функції втрат для вирішення задач регресії.
- Hinge losses for "maximum-margin" classification – функції втрат для розділових (маржевих) класифікаторів (застосовується рідко).

Якщо вирішується задача класифікації, використовують наступні критерії якості:

- **Binary Crossentropy** – якщо в задачі два класи.
- **Categorical Crossentropy** – якщо в задачі більше двох класів.
- **Sparse Categorical Crossentropy** – якщо мітки класів представлені звичайними числами (факторами), а не в категоріальному форматі One hot encoding (такий варіант не рекомендується).

Якщо вирішується задача регресії (прогнозування), то зазвичай використовують такі критерії якості:

- **Mean Squared Error** – середня сума квадратів помилок. Нестійка до рідкісних великий за модулем помилок - викидів.
- **Mean Absolute Error** – середня сума модулів помилок. Стійка до викидів, але повільніше збіжність.
- **Mean Absolute Percentage Error** – середня сума модулів помилок у відсотках (простими словами – відносна похибка). Зручна для інтерпретації.
- **Mean Squared Logarithmic Error** – середня сума логарифмів квадратів помилок. Зручна якщо величина змінюється в широкому діапазоні.

На практиці людині важко інтерпретувати значення критеріїв якості. Наприклад, важко зрозуміти, що означає "значення категоріальної крос-ентропії на тестовій множині складає 0,036". Набагато більш легким для сприйняття буде "доля правильних відповідей на тестовій множині складає 98%". Саме тому разом з критеріями якості в Keras застосовуються метрики - показники ефективності нейронної мережі, які людині легше сприйняти і осмислити.

- Доступні метрики в Keras: <https://keras.io/api/metrics/>

Для задач класифікації найчастіше використовуються:

- **Accuracy** (іноді ще **BinaryAccuracy**, **CategoricalAccuracy**) – доля правильних відповідей. Застосовується практично завжди.

- **Precision, Recall** – метрики, засновані на підрахунку кількості помилок I та II роду. Застосовуються у випадку несбалансованих класів.
- **MeanIoU** – застосовується для визначення якості сегментації зображень.

Для задач регресії метриками можуть виступати самі критерії якості.

Найчастіше використовуються:

- **Mean Absolute Error** – середня сума модулів помилок. Цю метрику легко інтерпретувати, адже одиниці її вимірювання співпадають з одиницями цілі.
- **Mean Absolute Percentage Error** – середня сума модулів помилок у відсотках.
- **RootMeanSquaredError** – аналог суми квадратів помилок, але додатково береться квадратний корінь. Знову ж таки, щоб узгодити одиниці вимірювання відповіді.

Спробуємо замінити метрику Accuracy на Categorical Accuracy у нашому прикладі. Ця метрика відрізняється від звичайної Accuracy тим, що можна додатково налаштувати деякі її параметри (але ми цього робити не будемо).


```

model7 = Sequential()
model7.add(Dense(6, input_dim=X_train.shape[1], activation="relu"))
model7.add(Dense(4, activation="softmax"))

model7.compile(optimizer='adam', loss='categorical_crossentropy',
               metrics=['categorical_accuracy'])

history7 = model7.fit(X_train, Y_train, epochs=50,
                     validation_split=0.15, batch_size=16)

loss, categ_accuracy = model7.evaluate(X_test, Y_test, verbose=0)
print(loss, categ_accuracy)

>>

Train on 1445 samples, validate on 255 samples
Epoch 1/50
1445/1445 [=====] - 0s 226us/step - loss: 1.5554 -
categorical_accuracy: 0.2429 - val_loss: 1.3968 - val_categorical_accuracy:
0.2824
Epoch 2/50
1445/1445 [=====] - 0s 156us/step - loss: 1.3957 -
categorical_accuracy: 0.2962 - val_loss: 1.2931 - val_categorical_accuracy:
0.3647
Epoch 3/50
1445/1445 [=====] - 0s 152us/step - loss: 1.2891 -
categorical_accuracy: 0.3779 - val_loss: 1.2167 - val_categorical_accuracy:
0.4078
...

Epoch 48/50
1445/1445 [=====] - 0s 151us/step - loss: 0.1697 -
categorical_accuracy: 0.9737 - val_loss: 0.2034 - val_categorical_accuracy:
0.9412
Epoch 49/50
1445/1445 [=====] - 0s 157us/step - loss: 0.1649 -
categorical_accuracy: 0.9709 - val_loss: 0.1992 - val_categorical_accuracy:
0.9451
Epoch 50/50
1445/1445 [=====] - 0s 151us/step - loss: 0.1607 -
categorical_accuracy: 0.9723 - val_loss: 0.1933 - val_categorical_accuracy:
0.9529
0.18950493037700653 0.953333331823349

```

Виводимо графік з історією навчання (рис. 3.7) для оцінки ефективності навчання.

```
plt.plot(history7.history['categorical_accuracy'])
plt.plot(history7.history['val_categorical_accuracy'])
plt.title('Model categorical_accuracy')
plt.ylabel('categorical_accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()

>>
```

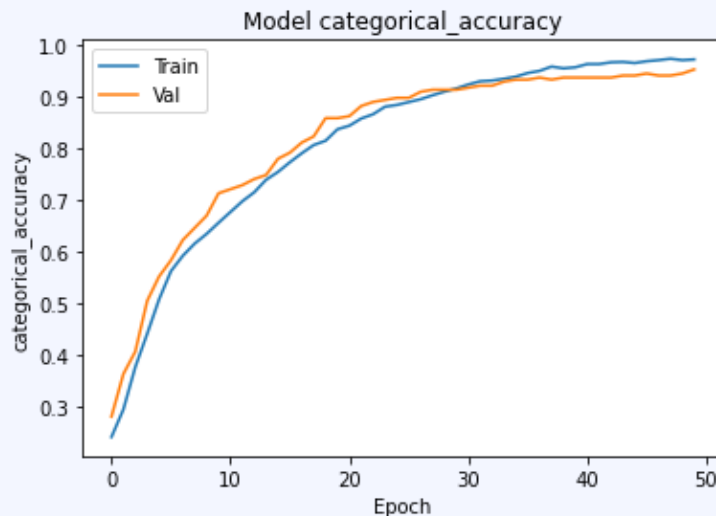


Рис. 3.7. Якість навчання для кожної епохи з метрикою Categorical Accuracy:

— навчальна множина; — валідаційна множина

Значення категоріальної долі правильних відповідей склало 95,3% на тестовій множині.

Підбір характеристик оптимізаторів

Не дивлячись на те, що на даний момент найбільш популярним оптимізатором є Adam, іноді буває необхідність замінити його або підлаштувати деякі параметри (наприклад, швидкість навчання learning rate). Для цього можна створити новий екземпляр класу потрібного оптимізатора, і вказати бажані атрибути замість стандартних. Можна навіть створити власний оптимізатор.

- Доступні оптимізатори в Keras: <https://keras.io/api/optimizers/>

Для того, щоб створити оптимізатор з власними налаштуваннями, необхідно в документації знайти опис відповідного класу оптимізатора та переглянути список доступних атрибутів.

➤ Наприклад, опис класу Adam: <https://keras.io/api/optimizers/adam/>

Використовуючи такий підхід, створимо оптимізатор Adam з нестандартними параметрами (змінимо learning rate) і застосуємо його в нашій моделі.

```
from keras import optimizers

my_adam = optimizers.Adam(learning_rate=0.01)

model8 = Sequential()
model8.add(Dense(6, input_dim=X_train.shape[1], activation="relu"))
model8.add(Dense(4, activation="softmax"))

model8.compile(optimizer=my_adam, loss='categorical_crossentropy',
               metrics=['accuracy'])

history8 = model8.fit(X_train, Y_train, epochs=50,
                     validation_split=0.15, batch_size=16)

loss, accuracy = model8.evaluate(X_test, Y_test, verbose=0)
print(loss, accuracy)

Train on 1445 samples, validate on 255 samples
Epoch 1/50
1445/1445 [=====] - 0s 211us/step - loss:
1.2799 - accuracy: 0.4014 - val_loss: 0.8947 - val_accuracy: 0.5922
Epoch 2/50
1445/1445 [=====] - 0s 153us/step - loss:
0.6723 - accuracy: 0.7322 - val_loss: 0.5069 - val_accuracy: 0.7961
Epoch 3/50
1445/1445 [=====] - 0s 149us/step - loss:
0.4253 - accuracy: 0.8671 - val_loss: 0.3715 - val_accuracy: 0.8706
...

Epoch 48/50
1445/1445 [=====] - 0s 168us/step - loss:
0.0614 - accuracy: 0.9730 - val_loss: 0.1665 - val_accuracy: 0.9333
Epoch 49/50
1445/1445 [=====] - 0s 158us/step - loss:
0.0614 - accuracy: 0.9730 - val_loss: 0.1895 - val_accuracy: 0.9333
Epoch 50/50
1445/1445 [=====] - 0s 163us/step - loss:
```

```
0.0753 - accuracy: 0.9696 - val_loss: 0.1548 - val_accuracy: 0.9412  
0.22023165668050448 0.9266666769981384
```

Виводимо графік з історією навчання (рис. 3.8) для оцінки ефективності навчання.

```
plt.plot(history8.history['accuracy'])  
plt.plot(history8.history['val_accuracy'])  
plt.title('Model accuracy')  
plt.ylabel('accuracy')  
plt.xlabel('Epoch')  
plt.legend(['Train', 'Val'], loc='upper left')  
plt.show()  
  
>>
```

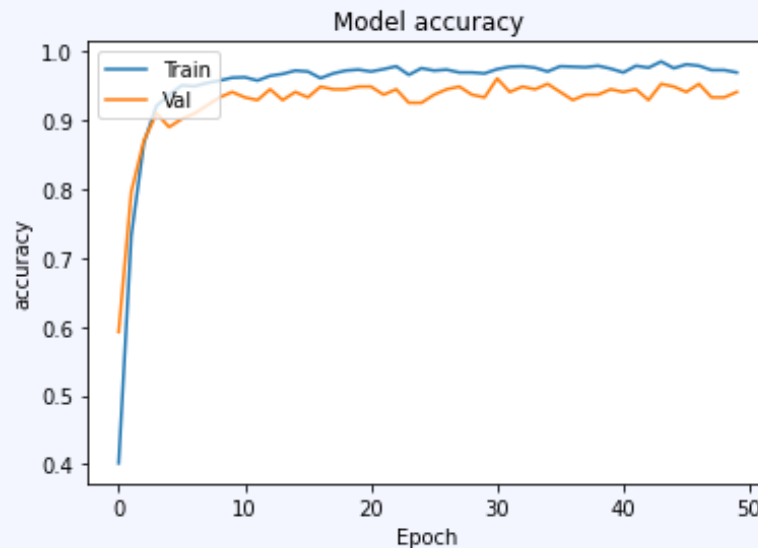


Рис. 3.8. Якість навчання для кожної епохи з модифікованим оптимізатором Adam ($learning_rate=0.01$): — навчальна множина; — валідаційна множина

Як бачимо, збільшивши швидкість навчання, оптимізатор швидше знайшов мінімум. Але при цьому є ризик пропустити інший мінімум, до чого може призвести занадто великий крок навчання - оптимізатор просто "перестрибне" необхідну область.

Окрім модифікацій параметрів оптимізатора Adam, можна для порівняння створити інший оптимізатор з використанням класичного методу градієнтного спуску з поправкою (моментом) Нестерова.

```

from keras import optimizers

my_SGD = optimizers.SGD(learning_rate=0.001, momentum=0.8,
                        nesterov=True)

model9 = Sequential()
model9.add(Dense(6, input_dim=X_train.shape[1], activation="relu"))
model9.add(Dense(4, activation="softmax"))

model9.compile(optimizer=my_SGD, loss='categorical_crossentropy',
               metrics=['accuracy'])

history9 = model9.fit(X_train, Y_train, epochs=100,
                     validation_split=0.15, batch_size=16)

loss, accuracy = model9.evaluate(X_test, Y_test, verbose=0)
print(loss, accuracy)

>>

Train on 1445 samples, validate on 255 samples
Epoch 1/100
1445/1445 [=====] - 0s 176us/step - loss:
1.5448 - accuracy: 0.3149 - val_loss: 1.4357 - val_accuracy: 0.3725
Epoch 2/100
1445/1445 [=====] - 0s 129us/step - loss:
1.4731 - accuracy: 0.3280 - val_loss: 1.3876 - val_accuracy: 0.3961
Epoch 3/100
1445/1445 [=====] - 0s 130us/step - loss:
1.4253 - accuracy: 0.3433 - val_loss: 1.3520 - val_accuracy: 0.4078
...

Epoch 98/100
1445/1445 [=====] - 0s 134us/step - loss:
0.2531 - accuracy: 0.9606 - val_loss: 0.2769 - val_accuracy: 0.9176
Epoch 99/100
1445/1445 [=====] - 0s 135us/step - loss:
0.2509 - accuracy: 0.9599 - val_loss: 0.2749 - val_accuracy: 0.9216
Epoch 100/100
1445/1445 [=====] - 0s 149us/step - loss:
0.2488 - accuracy: 0.9585 - val_loss: 0.2729 - val_accuracy: 0.9216
0.28417868455251055 0.9399999976158142

```

Виводимо графік з історією навчання (рис. 3.9) для оцінки ефективності навчання.

```
plt.plot(history9.history['accuracy'])
plt.plot(history9.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()

>>
```

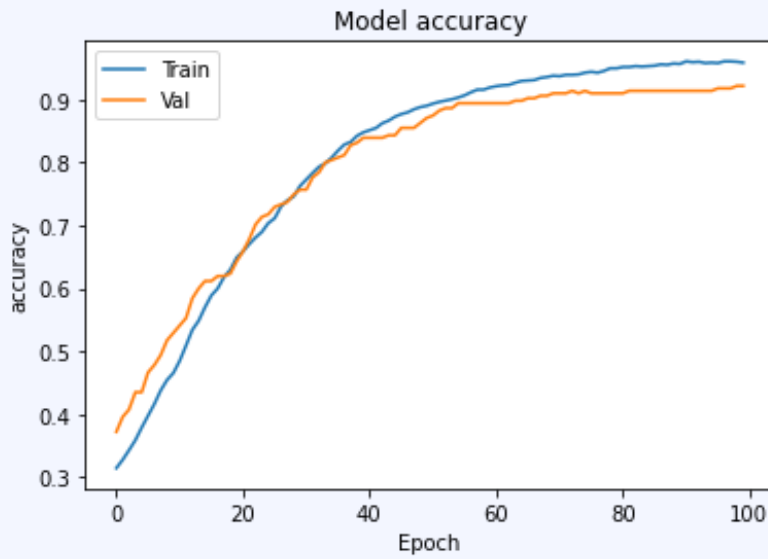


Рис. 3.9. Якість навчання для кожної епохи при застосуванні в якості оптимізатора методу градієнтного спуску з поправкою Нестерова (*learning_rate=0.01*):

— навчальна множина; — валідаційна множина

Як можна побачити, оптимізатору SGD знадобилось більше ітерацій навчання, щоб знайти мінімум. Водночас, показники якості роботи мережі суттєво не покращились. Ассурасу на тестовій множині склала майже 94%. Висновок - змінюємо параметри оптимізатора або використовуємо Adam.

Підбір характеристик ініціалізації ваг (weight initializers)

Ми вже знаємо, що перед початком навчання мережі необхідно задати початкові значення ваг - ініціалізувати їх. Але яким чином провести цю ініціалізацію? Варіантів багато.

➤ Документація: <https://keras.io/api/layers/initializers/>

Найчастіше ваги ініціалізуються випадковими маленькими числами. Ми можемо задати закон розподілу цих чисел або взагалі провести ініціалізацію константами (тільки для ваг зміщень *bias*!). Keras пропонує нам наступні варіанти:

- **Random Normal** – ініціалізація за нормальним законом розподілу. Використовується за замовчуванням.
- **Random Uniform** – ініціалізація за рівномірним законом розподілу.
- **Truncated Normal** – ініціалізація за нормальним законом розподілу, для якого введені граничні значення величин (усічений нормальний закон).
- **Zeros** – встановити всі початкові ваги в 0. Не сильно зрозуміло, навіщо.
- **Ones** – встановити всі початкові ваги в 1. Іноді застосовується в мережах LSTM.
- **Constant** – ініціалізувати константою. Можна використовувати для ініціалізації ваг зміщень. Має бути маленьким дійсним числом!

Але найчастіше нестандартна ініціалізація ваг використовується з метою доступу до параметру *seed* – зерна датчику випадкових чисел. Встановивши цей параметр, при кожній реініціалізації (повторному встановленню) початкових значень ваг їх величини будуть однаковими. Таким чином, запам'ятавши значення початкових ваг, можна в будь-який перенавчити мережу для отримання тих самих характеристик її якості (якщо і інші параметри навчання не змінюються). Оскільки початкові значення ваг залишаються однаковими, оптимізатор завжди починає роботу алгоритму з тієї самої початкової точки.

Застосуємо алгоритм ініціалізації ваг нейронів (kernel) **Truncated Normal**, а ваги зміщень (*bias*) ініціалізуємо константами. Також задамо параметр *seed* для відтворюваності результатів навчання у разі перезапуску. Ініціалізатори ваг вказуються як атрибути прошарку під час його створення.

```

from keras import initializers

init = initializers.TruncatedNormal(mean=0., stddev=0.05,
                                     seed=12345)
init_b = initializers.Constant(1e-3)

model10 = Sequential()
model10.add(Dense(6, input_dim=X_train.shape[1], activation="relu",
                  kernel_initializer=init, bias_initializer=init_b))
model10.add(Dense(4, activation="softmax", kernel_initializer=init,
                  bias_initializer=init_b))

model10.compile(optimizer='adam', loss='categorical_crossentropy',
                metrics=['accuracy'])

history10 = model10.fit(X_train, Y_train, epochs=50,
                       validation_split=0.15, batch_size=16)

loss, accuracy = model10.evaluate(X_test, Y_test, verbose=0)
print(loss, accuracy)

>>

Train on 1445 samples, validate on 255 samples
Epoch 1/50
1445/1445 [=====] - 0s 211us/step - loss:
1.3841 - accuracy: 0.3176 - val_loss: 1.3755 - val_accuracy: 0.4667
Epoch 2/50
1445/1445 [=====] - 0s 153us/step - loss:
1.3379 - accuracy: 0.5315 - val_loss: 1.2712 - val_accuracy: 0.5020
Epoch 3/50
1445/1445 [=====] - 0s 154us/step - loss:
1.1840 - accuracy: 0.4997 - val_loss: 1.0916 - val_accuracy: 0.5098
...

Epoch 48/50
1445/1445 [=====] - 0s 155us/step - loss:
0.1212 - accuracy: 0.9785 - val_loss: 0.1349 - val_accuracy: 0.9647
Epoch 49/50
1445/1445 [=====] - 0s 163us/step - loss:
0.1192 - accuracy: 0.9813 - val_loss: 0.1319 - val_accuracy: 0.9725
Epoch 50/50
1445/1445 [=====] - 0s 176us/step - loss:
0.1178 - accuracy: 0.9820 - val_loss: 0.1307 - val_accuracy: 0.9608
0.141209290822347 0.95333331823349

```

Виводимо графік з історією навчання (рис. 3.10) для оцінки ефективності навчання.


```
plt.plot(history10.history['accuracy'])
plt.plot(history10.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()

>>
```

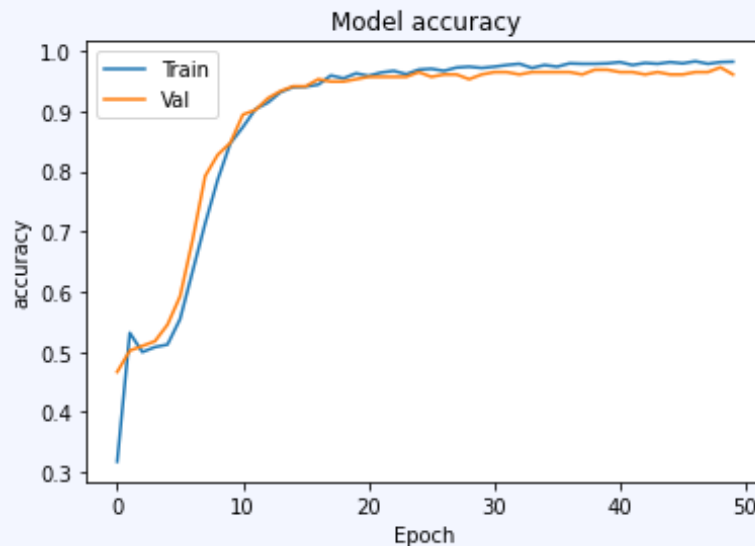


Рис. 3.10. Якість навчання для кожної епохи при застосуванні власної ініціалізації:

— навчальна множина; — валідаційна множина

Отримали Ассигасу на тестовій множині 95,3%. За графіком навчання видно, що на початкових ітераціях оптимізатор знаходився далеко від мінімуму, але зрештою вийшов з цієї ситуації. Це трапилось тому що ініціалізовані значення ваг були підібрані не зовсім вдало.

Підсумок

Коли який із розглянутих підходів застосовувати - питання без відповіді. Метод проб і помилок. Лише підбором і комбінацією описаних методів можна досягти оптимального результату.

Зазвичай, найбільший вплив на ефективність роботи мережі мають збалансована вибірка, правильно підібрана архітектура, регуляризація ваг та Batch normalization. Часто корисним є Dropout – якщо в мережі багато нейронів і вона має

складну архітектуру. Також не потрібно забувати і про early stopping. Спробуємо створити мережу із використанням цих підходів.

```
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense, Dropout, BatchNormalization
from keras import optimizers
from keras import initializers

early = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy',
                                          patience=5)

init = initializers.TruncatedNormal(mean=0., stddev=0.05,
                                     seed=98765)
init_b = initializers.Constant(1e-3)

model11 = Sequential()
model11.add(Dense(6, input_dim=X_train.shape[1], activation="relu",
                 kernel_initializer=init, bias_initializer=init_b,
                 kernel_regularizer='l2'))
model11.add(BatchNormalization())
model11.add(Dropout(0.2))
model11.add(Dense(4, activation="softmax", kernel_initializer=init,
                 bias_initializer=init_b, kernel_regularizer='l2'))
model11.compile(optimizer='adam', loss='categorical_crossentropy',
               metrics=['accuracy'])

history11 = model11.fit(X_train, Y_train, epochs=50,
                       validation_split=0.15, batch_size=16,
                       callbacks=[early])
loss, accuracy = model11.evaluate(X_test, Y_test, verbose=0)
print(loss, accuracy)

>>

Train on 1445 samples, validate on 255 samples
Epoch 1/50
1445/1445 [=====] - 1s 404us/step - loss:
1.3525 - accuracy: 0.3889 - val_loss: 1.3643 - val_accuracy: 0.5765
Epoch 2/50
1445/1445 [=====] - 0s 230us/step - loss:
1.1678 - accuracy: 0.5737 - val_loss: 1.2343 - val_accuracy: 0.5961
Epoch 3/50
1445/1445 [=====] - 0s 232us/step - loss:
0.9482 - accuracy: 0.6415 - val_loss: 1.0163 - val_accuracy: 0.7294
...
```

```

Epoch 17/50
1445/1445 [=====] - 0s 271us/step - loss:
0.4256 - accuracy: 0.8533 - val_loss: 0.2953 - val_accuracy: 0.9373
Epoch 18/50
1445/1445 [=====] - 0s 250us/step - loss:
0.4208 - accuracy: 0.8484 - val_loss: 0.2891 - val_accuracy: 0.9333
Epoch 19/50
1445/1445 [=====] - 0s 257us/step - loss:
0.3980 - accuracy: 0.8526 - val_loss: 0.2799 - val_accuracy: 0.9216
0.2740660031636556 0.96333333086967468

```

Виводимо графік з історією навчання (рис. 3.11) для оцінки ефективності навчання.

```

plt.plot(history11.history['accuracy'])
plt.plot(history11.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()

>>

```

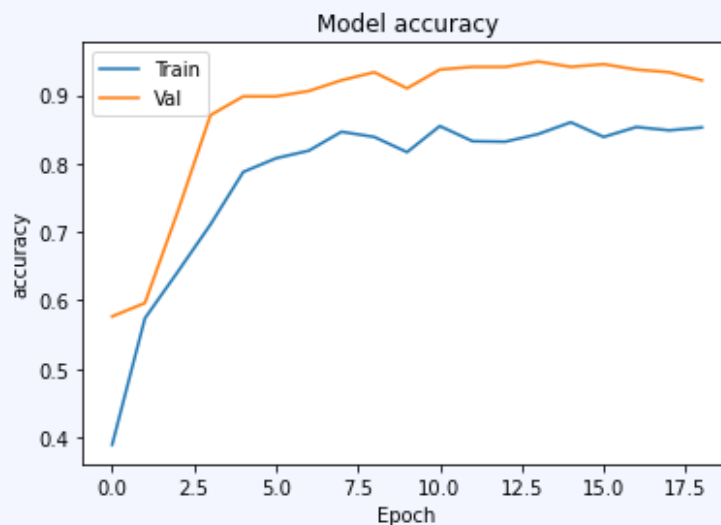


Рис. 3.11. Якість навчання для кожної епохи при одночасному застосуванні декількох методів оптимізації :

— навчальна множина; — валідаційна множина

Отже, в результаті ми отримали показник долі правильних відповідей на тестовій множині у розмірі 96,3%, що є набагато кращим у порівнянні з початковою моделлю (88,6%). Ось так "тюнінгують" нейронні мережі:)

Навчання пакету моделей

Оскільки кожен нейронну мережу необхідно навчати декілька разів (з метою уникнення локальних мінімумів), корисно робити це в циклі. Будемо створювати одразу кілька моделей, а потім з них обирати найкращу. Далі код без зайвих коментарів.

```
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense, Dropout, BatchNormalization
from keras import optimizers
from keras import initializers

# Вказуємо, скільки моделей ми хочемо навчити
NumberOfModels = 10
# Створюємо заготовки для відповідних списків
Model = []
History = []
Loss = []

# Оголошуємо необхідні налаштування
early = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy',
                                          patience=7)

init = initializers.TruncatedNormal(mean=0., stddev=0.05)
init_b = initializers.Constant(1e-3)

# Власне цикл
for i in range(NumberOfModels):
    Model.append(Sequential())
    Model[i].add(Dense(6, input_dim=X_train.shape[1],
                      activation="relu", kernel_initializer=init,
                      bias_initializer=init_b,
                      kernel_regularizer='l2'))
    Model[i].add(BatchNormalization())
    Model[i].add(Dropout(0.2))
    Model[i].add(Dense(4, activation="softmax",
                      kernel_initializer=init,
                      bias_initializer=init_b,
                      kernel_regularizer='l2'))
```

```

Model[i].compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

History.append(Model[i].fit(X_train, Y_train, epochs=50,
                           validation_split=0.15,
                           batch_size=16, callbacks=[early]))

Loss.append(Model[i].evaluate(X_test, Y_test, verbose=0))

# Обираємо найкращу модель за показником Accurasy на тестовій множині
best = max([Loss[i][1] for i in range(NumberOfModels)])
print('Best: ', best)
Train on 1445 samples, validate on 255 samples
Epoch 1/50
1445/1445 [=====] - 1s 414us/step - loss:
1.3066 - accuracy: 0.4353 - val_loss: 1.3423 - val_accuracy: 0.5176
Epoch 2/50
1445/1445 [=====] - 0s 238us/step - loss:
1.0320 - accuracy: 0.6062 - val_loss: 1.1641 - val_accuracy: 0.6275
Epoch 3/50
1445/1445 [=====] - 0s 238us/step - loss:
0.8541 - accuracy: 0.6644 - val_loss: 0.9619 - val_accuracy: 0.8549
...

Epoch 34/50
1445/1445 [=====] - 0s 234us/step - loss:
0.4049 - accuracy: 0.8630 - val_loss: 0.2619 - val_accuracy: 0.9490
Epoch 35/50
1445/1445 [=====] - 0s 243us/step - loss:
0.4120 - accuracy: 0.8540 - val_loss: 0.2606 - val_accuracy: 0.9333
Epoch 36/50
1445/1445 [=====] - 0s 244us/step - loss:
0.4109 - accuracy: 0.8505 - val_loss: 0.2512 - val_accuracy: 0.9451

Best:  0.9599999785423279

```

3.2 Завдання для самостійного виконання

Загальне завдання для всіх варіантів

На основі розглянутих підходів до оптимізації нейронних мереж модифікувати модель, розроблену під час виконання попередньої самостійної роботи. Експериментальним шляхом визначити методи або комбінацію методів та значень параметрів, які найбільше впливають на ефективність роботи нейронної мережі. Критерієм правильності виконання завдання буде помітне покращення якості роботи мережі на тестовій вибірці у порівнянні з аналогічним показником, отриманим під час виконання попередньої роботи.

ПРАКТИКУМ 4. НЕЙРОННІ МЕРЕЖІ В ЗАДАЧАХ ПРОГНОЗУВАННЯ (РЕГРЕСІЇ)

4.1. Загальні відомості

Задачі прогнозування відрізняються від задач класифікації тим, що цілі Y є не номерами класів, а мають кількісну шкалу (грн, \$, кг, В тощо). Відповідно, у вихідному прошарку мережі завжди буде лише один нейрон. Окрім того, до вихідного прошарку не застосовується жодна активаційна функція (або використовується лінійна). Прикладами регресії є прогнозування прибутку, прогноз погоди, прогнозування обсягів продажів або цін на товари. З технічної точки зору, нейронні мережі прямого розповсюдження для прогнозування нічим не відрізняються від аналогічних мереж для класифікації. Важливо лише правильно підібрати метрики та критерії якості і підготувати дані для аналізу.

- Критерії якості для задач регресії в Keras:

https://keras.io/api/losses/regression_losses/

- Метрики для задач регресії в Keras:

https://keras.io/api/metrics/regression_metrics/

Одним із прикладів задач регресії є прогнозування часових послідовностей (часових рядів).

Часовий ряд (англ. time series) – це ряд точок даних, проіндексованих (або перелічених, або відкладених на графіку) в хронологічному порядку. Найчастіше часовий ряд є послідовністю, взятою на рівновіддалених точках в часі, які йдуть одна за одною. Таким чином, він є послідовністю даних дискретного часу. Прикладами часових рядів є висоти океанських припливів, кількості сонячних плям, та щоденне значення вартості акцій на момент закриття торгів. (Wikipedia)

Найкраще в цих задачах себе проявляють мережі довгої короткострокової пам'яті (Long Short-Term Memory, LSTM). Але зараз ми навчимося вирішувати цю задачу із використанням вже знайомих нам Feedforward мереж.

Підготовка даних

Розглянемо набір даних про обсяг пасажирських перевезень у США за 1949 - 1960 роки. Цей приклад є класичним і найбільш демонстративним щоб гарно розібратись з усіма особливостями прогнозування рядів.

Задача: необхідно зробити прогноз обсягів пасажирських перевезень на 12 місяців вперед.

Для початку імпортуємо необхідні бібліотеки.

```
import numpy
import pandas as pd

import matplotlib
import matplotlib.pyplot as plt
matplotlib.style.use('ggplot')
# Розмір графіків
plt.rcParams['figure.figsize'] = (15, 5)
```

Завантажимо навчальну вибірку - часовий ряд, на основі якого будемо робити прогноз.

```
ser_g = pd.read_csv('../series_g.csv', sep=';', header=0)
```

Переглянемо заголовки і кінець таблиці:

```
ser_g.head()

>>
      date  series_g
0  JAN 1949        112
1  FEB 1949        118
2  MAR 1949        132
3  APR 1949        129
4  MAY 1949        121

ser_g.tail()

      date  series_g
139  AUG 1960        606
140  SEP 1960        508
141  OCT 1960        461
142  NOV 1960        390
143  DEC 1960        432
```


Як бачимо, дані організовані по місяцям. В стовпці *date* – місяць і рік, в стовпці *sreies_g* – обсяг пасажирських перевезень за відповідний час.

Переглянемо, скільки всього записів є в наборі даних.

```
ser_g.shape  
  
>>  
  
(144, 2)
```

Отже, маємо дані за 144 місяці.

Тепер побудуємо графік (рис. 4.1), щоб відповісти на 4 питання:

- Чи є тренд? Тренд - це загальна тенденція в поведінці ряду. Тренд описує, як змінюються середні значення ряду із часом. Якщо тренд є, потрібно оцінити його характер: зростаючий/спадаючий, лінійний/експоненціальний/затухаючий тощо.
- Чи є сезонність? Сезонність - періодичні зміни значень ряду з часом. Якщо є сезонність, то яка вона? Який період сезону? Сезонність адитивна (значення сезонних поправок мають постійну амплітуду відносно лінії тренду) чи мультиплікативна (значення сезонних поправок збільшуються/зменшуються пропорційно зростанню/спаданню тренду)? Якщо сезонність мультиплікативна, її потрібно привести до адитивної (наприклад, прологарифмувавши значення ряду).
- Чи змінює ряд свій характер? Зміну характеру можна виявити за різкою зміною тренду. Якщо ряд змінює свій характер, для прогнозування необхідно використовувати лише останню його ділянку, на якій характер ряду незмінний.
- Чи є викиди або пропущені значення? Викиди - аномально великі або маленькі значення в ряді. Якщо вони є, їх потрібно замінити більш "розумними" значеннями (найчастіше – середнім або медіаною між сусідніми відносно викиду спостереженнями).

```
# Графік щоб відповісти на 4 питання
ser_g.iloc[:,1].plot()

>>
```

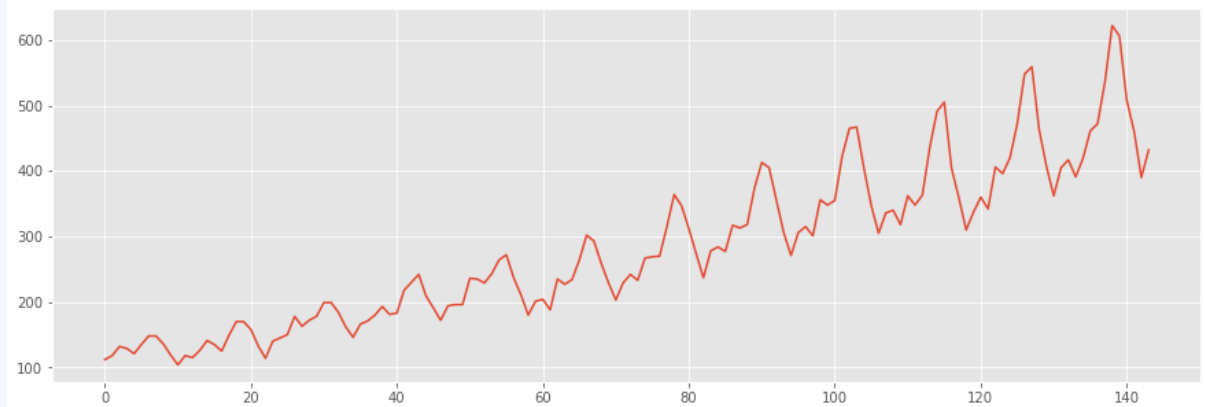


Рис. 4.1. Обсяг пасажирських перевезень у США за 1949 - 1960 роки (за 144 місяці)

Аналізуємо графік:

- Ряд має зростаючий лінійний тренд
- Ряд має мультиплікативну сезонність
- Ряд не змінює свій характер
- Викидів або пропущених значень немає

Оскільки сезонність мультиплікативна, її потрібно привести до адитивної, використавши логарифмування (рис. 4.2). Таким чином, модель буде вчитись робити прогноз для логарифму ряду, а не самого ряду.

```
# Потрібно прогнозувати логарифм
ser_g['log_y'] = numpy.log10(ser_g['series_g'])

fig = plt.figure(figsize=(12, 4))
ax1 = fig.add_subplot(121)
ser_g['series_g'].plot(ax=ax1)
ax1.set_title(u'Обсяг перевезень пасажирів')
ax1.set_ylabel(u'Тисяч осіб')

ax2 = fig.add_subplot(122)
pd.Series(ser_g['log_y']).plot(ax=ax2)
ax2.set_title(u'log10 від обсягу перевезень пасажирів')
ax2.set_ylabel(u'log10 від тисяч осіб')
```

```
Text(0, 0.5, 'log10 від тисяч осіб')
```

```
>>
```

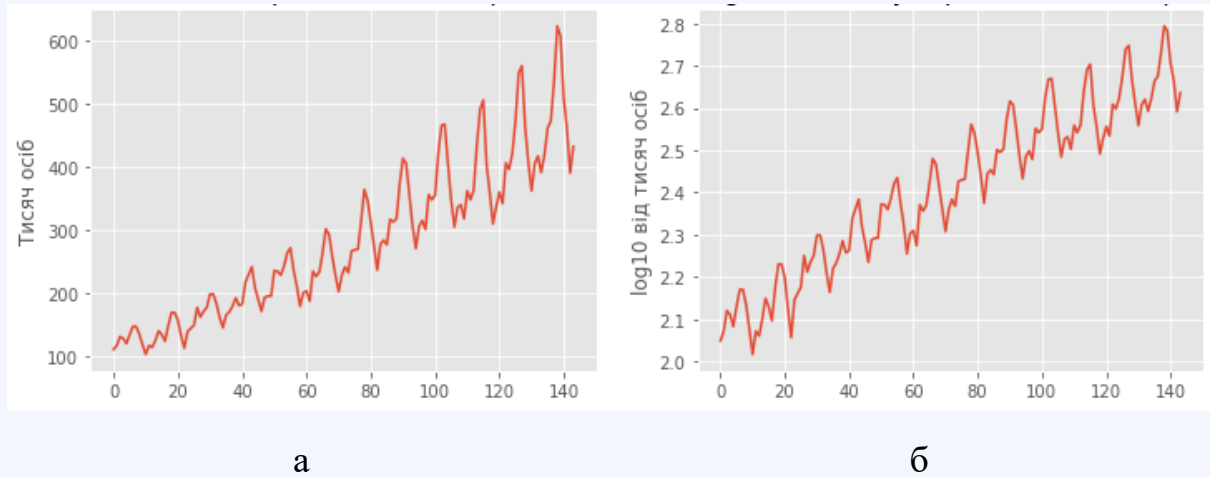


Рис. 4.2. Обсяг пасажирських перевезень у США за 144 місяці:

а – обсяг перевезень пасажирів; б – $\log_{10}$ від обсягу перевезень пасажирів

Тепер необхідно перетворити дані так, щоб отримати таблицю предикторів X та цілей Y . Оскільки тривалість сезону в нашому випадку складає 12 місяців (1 рік), то логічно буде за дванадцятьма попередніми місяцями робити прогноз на один наступний місяць. Тобто для одного рядка навчальних даних буде 12 стовпців X та один стовпець Y .

Під час навчання:

- За значеннями спостережень №1-№12 прогнозуємо спостереження №13 - це буде перша ітерація навчання (перший рядок навчальних даних). Предиктори X - спостереження №1-№12, ціль Y - спостереження №13.
- Зсуваємось на 1 місяць вперед. За значеннями спостережень №2-№13 прогнозуємо спостереження №14 (це буде другий рядок навчальних даних).
- Зсуваємось на 1 місяць вперед. За значеннями спостережень №3-№14 прогнозуємо спостереження № 15.

➤ Крок за кроком проходимо по всіх навчальних даних.

Виконаємо необхідні перетворення даних.

```
ser_g_2 = pd.DataFrame()

for i in range(12,0,-1):
    ser_g_2['t-'+str(i)] = ser_g.iloc[:,2].shift(i)

ser_g_2['t'] = ser_g.iloc[:,2].values

print(ser_g_2.head(13))

>>
```

	t-12	t-11	t-10	t-9	t-8	t-7	t-6 \
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN	NaN	NaN	2.049218
7	NaN	NaN	NaN	NaN	NaN	2.049218	2.071882
8	NaN	NaN	NaN	NaN	2.049218	2.071882	2.120574
9	NaN	NaN	NaN	2.049218	2.071882	2.120574	2.110590
10	NaN	NaN	2.049218	2.071882	2.120574	2.110590	2.082785
11	NaN	2.049218	2.071882	2.120574	2.110590	2.082785	2.130334
12	2.049218	2.071882	2.120574	2.110590	2.082785	2.130334	2.170262

	t-5	t-4	t-3	t-2	t-1	t
0	NaN	NaN	NaN	NaN	NaN	2.049218
1	NaN	NaN	NaN	NaN	2.049218	2.071882
2	NaN	NaN	NaN	2.049218	2.071882	2.120574
3	NaN	NaN	2.049218	2.071882	2.120574	2.110590
4	NaN	2.049218	2.071882	2.120574	2.110590	2.082785
5	2.049218	2.071882	2.120574	2.110590	2.082785	2.130334
6	2.071882	2.120574	2.110590	2.082785	2.130334	2.170262
7	2.120574	2.110590	2.082785	2.130334	2.170262	2.170262
8	2.110590	2.082785	2.130334	2.170262	2.170262	2.133539
9	2.082785	2.130334	2.170262	2.170262	2.133539	2.075547
10	2.130334	2.170262	2.170262	2.133539	2.075547	2.017033
11	2.170262	2.170262	2.133539	2.075547	2.017033	2.071882
12	2.170262	2.133539	2.075547	2.017033	2.071882	2.060698

Відрізаємо перші 12 рядків.

```
ser_g_4 = ser_g_2[12:]
ser_g_4.head()
>>
```

	t-12	t-11	t-10	t-9	t-8	t-7	t-6	\
12	2.049218	2.071882	2.120574	2.110590	2.082785	2.130334	2.170262	
13	2.071882	2.120574	2.110590	2.082785	2.130334	2.170262	2.170262	
14	2.120574	2.110590	2.082785	2.130334	2.170262	2.170262	2.133539	
15	2.110590	2.082785	2.130334	2.170262	2.170262	2.133539	2.075547	
16	2.082785	2.130334	2.170262	2.170262	2.133539	2.075547	2.017033	

	t-5	t-4	t-3	t-2	t-1	t
12	2.170262	2.133539	2.075547	2.017033	2.071882	2.060698
13	2.133539	2.075547	2.017033	2.071882	2.060698	2.100371
14	2.075547	2.017033	2.071882	2.060698	2.100371	2.149219
15	2.017033	2.071882	2.060698	2.100371	2.149219	2.130334
16	2.071882	2.060698	2.100371	2.149219	2.130334	2.096910

Навчальна таблиця даних готова. Тепер розділяємо предиктори та цілі.

```
# Цілі - вектор y
y = ser_g_4['t']
# Предиктори - таблиця X
X = ser_g_4.drop('t', axis=1)
```

Розділяємо дані на навчальну і тестову множини. Тестова множина у випадку прогнозування часових рядів - останні спостереження. Оскільки нам важливо отримати точний прогноз саме для останнього періоду часу.

```
# Знадаємо, скільки рядків в навчальній таблиці
ser_g_4.shape

(132, 13)

# Візьмемо останні 12 спостережень в якості тестової вибірки
X_train = X[:120]
y_train = y[:120]
X_test = X[120:]
y_test = y[120:]

# Все добре?
print(ser_g_4.shape)
print(X_train.shape)
```

```
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
>>
```

```
(132, 13)
(120, 12)
(120,)
(12, 12)
(12,)
```

```
# Все добре?
```

```
X_train.head()
```

	t-12	t-11	t-10	t-9	t-8	t-7	t-6	\
12	2.049218	2.071882	2.120574	2.110590	2.082785	2.130334	2.170262	
13	2.071882	2.120574	2.110590	2.082785	2.130334	2.170262	2.170262	
14	2.120574	2.110590	2.082785	2.130334	2.170262	2.170262	2.133539	
15	2.110590	2.082785	2.130334	2.170262	2.170262	2.133539	2.075547	
16	2.082785	2.130334	2.170262	2.170262	2.133539	2.075547	2.017033	

	t-5	t-4	t-3	t-2	t-1
12	2.170262	2.133539	2.075547	2.017033	2.071882
13	2.133539	2.075547	2.017033	2.071882	2.060698
14	2.075547	2.017033	2.071882	2.060698	2.100371
15	2.017033	2.071882	2.060698	2.100371	2.149219
16	2.071882	2.060698	2.100371	2.149219	2.130334

Навчальні дані готові. Можемо створити і навчити нейронну мережу. В якості критерію якості використаємо *mean_squared_error*, метрика - *mean_absolute_percentage_error*. Активаційна функція вихідного нейрону - лінійна *activation='linear'*. Розділяти вибірку на батчі не будемо (даних і так мало) - *batch_size=None*.

```
from keras.models import Sequential
from keras.layers import Dense
```

```
Using TensorFlow backend.
```

```
# Створюємо модель
```

```
model = Sequential()
model.add(Dense(8, input_dim=12, activation='relu'))
model.add(Dense(1, activation='linear'))
```

```

# Компілюємо модель
model.compile(loss='mean_squared_error', optimizer='adam',
              metrics=['mean_absolute_percentage_error'])

# Навчаємо модель
model.fit(X_train, y_train, epochs=300, batch_size=None)

>>

Epoch 1/300
120/120 [=====] - 0s 708us/step - loss: 4.6182
- mean_absolute_percentage_error: 88.9544
Epoch 2/300
120/120 [=====] - 0s 108us/step - loss: 3.7469
- mean_absolute_percentage_error: 80.0813
Epoch 3/300
120/120 [=====] - 0s 117us/step - loss: 2.9702
- mean_absolute_percentage_error: 71.2797

...
Epoch 298/300
120/120 [=====] - 0s 92us/step - loss: 0.0019
- mean_absolute_percentage_error: 1.4825
Epoch 299/300
120/120 [=====] - 0s 108us/step - loss: 0.0019
- mean_absolute_percentage_error: 1.4824
Epoch 300/300
120/120 [=====] - 0s 100us/step - loss: 0.0019
- mean_absolute_percentage_error: 1.4823

```

Виконаємо грубу оцінку якості моделі на тестових даних. Грубу - тому що таким способом ми уникаємо "накопичення помилки", оскільки використовуємо заздалегідь відомі значення всіх предикторів.

```

scores = model.evaluate(X_test, y_test)
print("\n rude MAPE: %.2f%%" % (scores[1]))

>>

12/12 [=====] - 0s 1ms/step

rude MAPE: 0.96%

```

Обчислюємо грубий прогноз на тестових даних.

```
false_predictions = model.predict(X_test)
```

Для правильної оцінки якості роботи мережі на тестових даних необхідно реалізувати наступний алгоритм:

- Взяти перший набір предикторів із тестових даних. В нашому випадку, це спостереження №109-№120. За їх значеннями спрогнозувати спостереження №121.
- Взяти спостереження №110-№121 (спрогнозоване на попередньому кроці). За ними спрогнозувати спостереження №122.
- Взяти спостереження №110-№122 (два останні - спрогнозовані на попередніх кроках). За ними спрогнозувати спостереження №123.
- Повторювати ці дії стільки разів, скільки значень потрібно спрогнозувати. В нашому випадку - 12.
- Лише після цього оцінити значення метрики, порахувавши помилки як різницю між реальними цілями y та спрогнозованими значеннями. Це треба зробити за власною формулою, а не стандартним методом *evaluate()*!

Напишемо власну функцію для прогнозування за описаним алгоритмом.

```
def make_prediction(X_predict, nb_of_predictions):  
    predictions = numpy.array([])  
  
    for i in range (nb_of_predictions):  
        y_predicted = model.predict(X_predict)  
        predictions = numpy.append(predictions, y_predicted)  
  
        X_predict = numpy.roll(X_predict, -1)  
        X_predict[0][-1] = y_predicted  
  
    return predictions
```

Проведемо оцінку якості моделі за метрикою MAPE.

```
# Перетворимо перший зразок із тестової вибірки на масив Numpy  
X_predict = numpy.array(X_test[:1])
```



```
# Використаємо написану функцію для отримання правильного прогнозу  
predictions = make_prediction(X_predict, len(X_test))
```

Отримаємо значення MAPE на тестових даних.

```
y_test = numpy.array(y_test)  
  
MAPE = 100*sum(numpy.abs(y_test - predictions) / numpy.maximum(y_test,  
1e-20))/len(y_test)  
print(f"\n MAPE: {MAPE:.2f} %")  
  
MAPE: 1.13 %
```

Обчислюємо допасування ("підгонку") моделі. Підгонка показує, наскільки якісно модель робить прогнози на навчальній множині. Порівнявши результати роботи мережі з правильними значеннями Y , можна візуально (на графіку) оцінити, наскільки близькими є прогнози моделі до реальних даних, на яких вона навчалась.

```
predictions_train = model.predict(X_train)
```

Згадуємо розміри таблиць і будуємо графіки (рис. 4.3).

```
print(X_train.shape)  
print(y_train.shape)  
print(X_test.shape)  
print(y_test.shape)  
  
>>  
  
(120, 12)  
(120,)  
(12, 12)  
(12,)
```

- Було 144 спостереження
- Відкинули 12, стало 132
- train 120
- test 12

```

# Графік з результатами

plt.rcParams['figure.figsize'] = (15, 5)

# numpy.arange([start, ]stop, [step, ]dtype=None)

x2 = numpy.arange(0, 120, 1)

x3 = numpy.arange(120, 132, 1)

# реальні дані (початковий ряд без відкинутих і тестових значень)
plt.plot(x2, y_train, color='blue')

# підгонка
plt.plot(x2, predictions_train, color='green')

# реальні дані на тестовій множині
plt.plot(x3, y_test, color='blue')

# грубий прогноз на тестовій множині
plt.plot(x3, false_predictions, color='purple')

# правильний прогноз на тестовій множині
plt.plot(x3, predictions, color='red')

>>

```

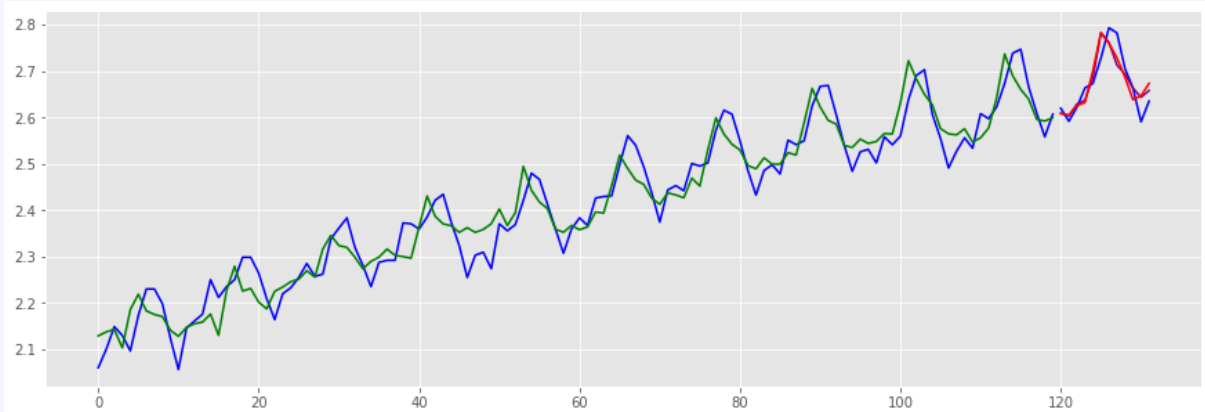


Рис. 4.3. Обсяг пасажирських перевезень у США за 144 місяці:

blue – початкові дані; *green* – апроксимація моделлю; *purple* – грубий прогноз;
red – правильний прогноз

Якщо результати роботи мережі нас задовольняють, можна використати навчену модель для створення реального прогнозу. Якщо ні - перенавчаємо. Реальний прогноз є продовженням початкового ряду на певну кількість спостережень. В нашому випадку - на 12 місяців вперед. Для створення прогнозу використаємо раніше написану нами функцію *make_prediction()*. Кількість предикторів X має відповідати кількості стовпців X в таблиці даних, на якій навчалась мережа. В нашому випадку, в якості першого набору предикторів X необхідно подати дані за останні 12 місяців з початкового ряду.

```
# Прогнозуємо на 12 місяців вперед
nb_of_predictions = 12

# Перетворюємо дані за останні 12 місяців в масиви Numpy
X_real_prediction = numpy.array(ser_g_4.iloc[-12:,1])
X_real_prediction = numpy.expand_dims(X_real_prediction, axis = 1)
X_real_prediction = numpy.transpose(X_real_prediction)

# Робимо прогноз
real_predictions = make_prediction(X_real_prediction,
                                   nb_of_predictions)
```

Побудуємо суміщений графік (рис. 4.4), на якому покажемо початковий ряд та прогноз на наступні 12 місяців.

```
# Дані з початкового ряду
x_past = numpy.arange(0, len(ser_g_4))
y_past = ser_g_4.iloc[:,1]

# Додаємо координати X для прогнозу
x_pred = numpy.arange(len(ser_g_4), len(ser_g_4) + nb_of_predictions)

# Початковий ряд
plt.plot(x_past, y_past, color='blue')

# Прогноз
plt.plot(x_pred, real_predictions, color='red')

>>
```

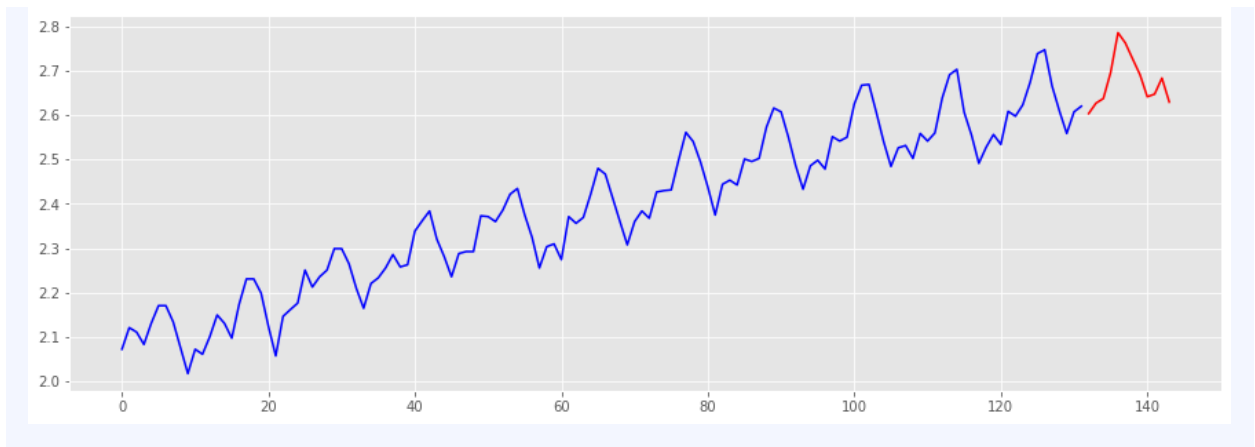


Рис. 4.4. Обсяг пасажирських перевезень у США з прогнозом на наступні 12 місяців: *blue* – початкові дані; *red* – прогноз

Оскільки ми навчили мережу прогнозувати логарифми даних, а не самі дані, необхідно виконати зворотне перетворення – піднесення до степеню. Логарифм був десятковим, тому для отримання реальних результатів (кількості осіб, а не її логарифмів) необхідно піднести 10 до відповідного ступеню. Побудуємо графік (рис. 4.5).

```
# Перетворюємо з логарифмів до реальної величини
non_log_predictions = 10**real_predictions

# Тепер побудуємо фінальний графік.

x_past = numpy.arange(0, len(ser_g))
y_past = ser_g.iloc[:,1]

x_pred = numpy.arange(len(ser_g), len(ser_g) + nb_of_predictions)

plt.plot(x_past, y_past, color='blue')
plt.plot(x_pred, non_log_predictions, color='red')

>>
```

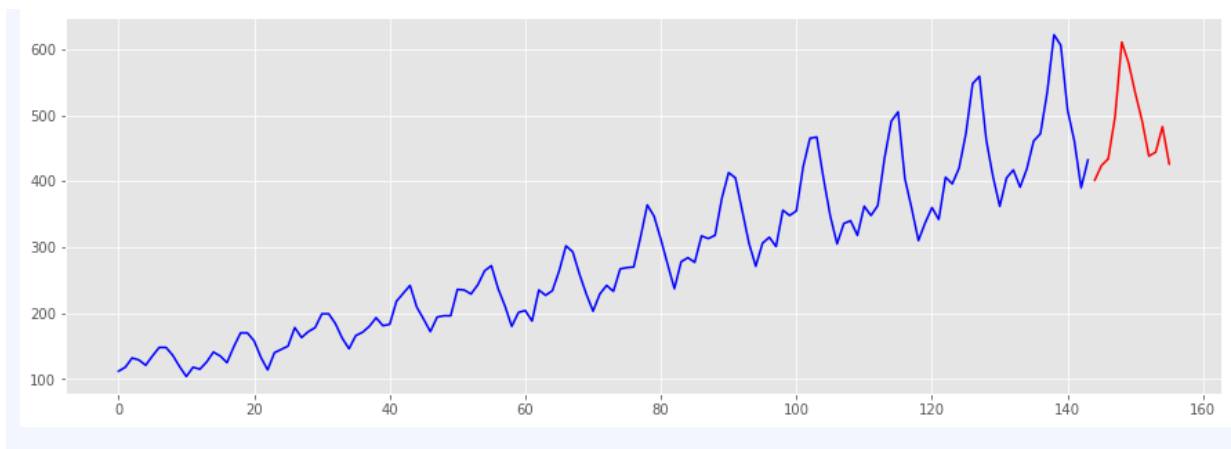


Рис. 4.4. Обсяг пасажирських перевезень у США з прогнозом на наступні 12 місяців: *blue* – початкові дані; *red* – прогноз (без логарифмування)

4.2. Завдання для самостійного виконання

Загальні завдання для всіх варіантів:

1. Завантажте набір даних. Скільки всього записів у ряді?
2. Виведіть заголовок та останні 5 записів таблиці.
3. Побудуйте графік часового ряду.
4. Проаналізуйте ряд, відповівши на 4 питання:
 - a. Чи є тренд?
 - b. Чи є сезонність?
 - c. Чи змінює ряд свій характер?
 - d. Чи є викиди або пропущені дані?
5. Виконайте перетворення даних для формування навчальної таблиці.
Врахуйте сезонність (якщо вона є) для вибору кількості стовпців.
6. Розділіть дані на предиктори X та цілі Y .
7. Сформууйте навчальну та тестову множини.
8. Створіть та навчіть нейронну мережу для прогнозування.
9. Отримайте грубу оцінку роботи мережі на тестових даних.
10. Отримайте реальну оцінку роботи мережі на тестових даних.
11. Побудуйте суміщений графік, на якому має бути відображено:
 - a. відрізок ряду з навчальних (початкових) даних;
 - b. відрізок ряду, отриманий за допомогою опрацювання навчальних даних нейронною мережею;
 - c. відрізок ряду, який відповідає реальним (початковим) даним на тестовій множині;
 - d. відрізок ряду, який відповідає грубому прогнозу мережі на тестовій множині;
 - e. відрізок ряду, який відповідає реальному прогнозу мережі на тестовій множині.

12. Використайте навчену мережу для отримання прогнозу на вказаний термін часу.
13. Побудуйте суміщений графік початкового ряду та прогнозу.

Варіант 1

Набір даних: ДТП в Великобританії (accident_UK_by_month.csv).

Опис даних: В наборі містяться дані про середньомісячну кількість ДТП в Великобританії за 2014-2017 роки. Зробити прогноз кількості ДТП на найближчі місяці.

Термін прогнозу: 8 місяців.

Варіант 2

Набір даних: трафік на дорогах міста (IOT_by_days.csv).

Опис даних: В наборі містяться дані про кількість транспортних засобів на одній із вулиць міста за кожен день протягом одного року (2015-2016). Необхідно зробити прогноз трафіку на наступні місяці.

Термін прогнозу: 1 місяць.

Варіант 3

Набір даних: кількість продажів продукту (Month_Value.csv).

Опис даних: В наборі містяться дані про кількість продажів деякого продукту по місяцях за 2015-2020 роки. Необхідно зробити прогноз продаж на наступні місяці.

Термін прогнозу: 5 місяців.

Варіант 4

Набір даних: клімат в Делі (Month_Value.csv).

Опис даних: В наборі містяться дані про середню температуру в Делі за кожен день 2013-2017 років. Необхідно зробити прогноз температури на наступні місяці.

Термін прогнозу: 2 місяці.

Варіант 5

Набір даних: акції Yahoo (yahoo_stock2.csv).

Опис даних: В наборі містяться дані про вартість акцій компанії Yahoo за 2015-2020 роки. Необхідно зробити прогноз вартості акцій на наступні місяці.

Термін прогнозу: 3 місяці.

Варіант 6

Набір даних: вартість золота (gold_price_data.csv).

Опис даних: В наборі містяться дані про вартість золота за 1970-2020 роки. Необхідно зробити прогноз вартості золота на наступні місяці. Для покращення якості прогнозу рекомендується взяти дані лише за останній рік. Врахувати, що біржа працює лише по робочих днях.

Термін прогнозу: 5 місяців.

Варіант 7

Набір даних: акції Apple (Apple_stock.csv).

Опис даних: В наборі містяться дані про вартість акцій компанії Apple за 2010-2020 роки. Необхідно зробити прогноз вартості акцій на наступні місяці.

Термін прогнозу: 3 місяці.

Варіант 8

Набір даних: продажі супермаркету Walmart (Grocery_Sales.csv).

Опис даних: В наборі містяться дані про щоденний обсяг продажів в одному із супермаркетів мережі Walmart. Необхідно зробити прогноз обсягів продажів.

Термін прогнозу: 2 місяці.

Варіант 9

Набір даних: вартість зерна в США (corn_price_US.csv).

Опис даних: В наборі містяться дані про середню за тиждень вартість зерна в США за 2013-2017 роки. Необхідно зробити прогноз вартості зерна на найближчі тижні.

Термін прогнозу: 10 тижнів.

Варіант 10

Набір даних: міжнародні авіап перевезення в США (US_airlines.csv).

Опис даних: В наборі містяться дані про щоденний обсяг перевезень пасажирів компанією авіакомпанією ASM. Необхідно зробити прогноз обсягів перевезень на наступні місяці.

Термін прогнозу: 5 місяців.

Варіант 11

Набір даних: продажі алкогольних напоїв в США (Alcohol_sales.csv).

Опис даних: В наборі містяться дані про щомісячний обсяг продажів алкоголю в США. Необхідно зробити прогноз обсягів продажів на наступні місяці.

Термін прогнозу: 6 місяців.

Варіант 12

Набір даних: продажі пива в США (Beer.csv).

Опис даних: В наборі містяться дані про щомісячний обсяг продажів пива в США. Необхідно зробити прогноз обсягів продажів на наступні місяці.

Термін прогнозу: 6 місяців.

ПРАКТИКУМ 5. ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ

5.1. Загальні відомості

Згорткові нейронні мережі – це окремий клас глибинних штучних НМ прямого розповсюдження, створений спеціально для аналізу візуальної інформації. Зазвичай, архітектура такої НМ складається з декількох багатовимірних прошарків штучних нейронів, оптимізованих для виявлення закономірностей у візуальних зображеннях. Особливістю моделі згорткової НМ є у доповнення архітектури повнозв'язної мережі прямого розповсюдження окремими згортковими прошарками, в яких кожен нейрон пов'язаний тільки з невеликою групою нейронів попереднього прошарку. Така організація мережі дозволяє виділяти на початковому зображенні лише примітивні діагностичні ознаки, такі як ребра або грані, а на наступних прошарках мережі об'єднувати виділені ознаки для отримання все більш складних елементів. Завдяки цьому з'являється можливість ефективно розпізнавати приховані закономірності та виділяти комплексні образи на зображеннях.

У згорткових НМ використовуються прошарки згортки та підвибірки. Стандартна архітектура виглядає так:

- Згортка у відповідних прошарках застосовується для формування мережею набору діагностичних ознак.
 - За допомогою прошарків підвибірки реалізується вибір найбільш значущих ознак попереднього прошарку і скорочення розмірності наступних прошарків.
 - Далі виконується операція перетворення отриманих карт ознак в одновимірний масив та класифікація термограми за допомогою одного або двох повнозв'язних прошарків.
- Детальніше про згорткові нейронні мережі:

<https://habr.com/ru/post/348000/>

- Згорткові прошарки в Keras:

https://keras.io/api/layers/convolution_layers/

- Прошарки підвибірки (пулінгу) в Keras:

https://keras.io/api/layers/pooling_layers/

Розпочнемо знайомство зі згортковими нейронними мережами на прикладі вирішення задачі бінарної класифікації зображень. Наша мета - автоматично розпізнати, яку тварину показано на зображенні - kota чи собаку.

Для початку імпортуємо всі необхідні модулі та функції.

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D, Flatten, Dense, MaxPooling2D, Dropout
from keras.regularizers import l2
from keras import utils
from keras.preprocessing import image
from tensorflow.keras.preprocessing import image_dataset_from_directory
from tensorflow.keras.layers.experimental.preprocessing import Rescaling

import matplotlib.pyplot as plt
```

Встановимо необхідний розмір батчу та розміри вхідного зображення. Всі зображення, які подаються на вхід згорткової нейронної мережі, повинні приводитись до однакового розміру.

```
BATCH_SIZE = 64
IMAGE_SIZE = (180, 180)
```

Сформуємо множину навчальних зображень, використовуючи функцію `image_dataset_from_directory()`. Ця функція зчитує зображення, які знаходяться у вказаній директорії. Набори зображень, які відповідають кожному класу, повинні знаходитись в окремих папках. За замовчуванням, назви класів будуть відповідати назвам папок, в яких розміщені зображення, що відповідають цим класам. В якості обов'язкового першого аргументу до функції

`image_dataset_from_directory()` передається адреса головної папки, у якій розміщені папки з зображеннями класів.

Параметр `subset` (підмножина) вказує на те, яка підмножина формується - навчальна чи тестова.

- **ВАЖЛИВО!** У разі використання функції `image_dataset_from_directory()`, навчальні зображення всіх класів мають знаходитись в папках, які в свою чергу розміщені в одній головній папці з навчальними даними. Функція автоматично розділить набори зображень на навчальну та валідаційну множину - не потрібно окремо створювати папки *train* та *validation*. Достатньо просто задати відповідне значення параметру `subset`!

Використовуючи такий підхід, обов'язково потрібно встановити параметр `seed` - значення зерна датчика випадкових чисел. Під час формування навчального та валідаційного наборів зображень це значення має бути однаковим для обох підмножин даних.

Після цього необхідно вказати параметр `validation_split` - доля зображень, які будуть віднесені до валідаційної підмножини.

Далі задається розмір батчу `batch_size` та розмір зображення `image_size` - під час формування набору даних всі зображення будуть автоматично масштабовані до вказаного розміру (в пікселях).

```
# Навчальний набір зображень
train_dataset = image_dataset_from_directory('../training_set',
                                              subset='training',
                                              seed=42,
                                              validation_split=0.15,
                                              batch_size=BATCH_SIZE,
                                              image_size=IMAGE_SIZE)

>>

Found 8000 files belonging to 2 classes.
Using 6800 files for training.
```

```
# Валідаційний набір зображень
validation_dataset = image_dataset_from_directory('../training_set',
                                                    subset='validation',
                                                    seed=42,
                                                    validation_split=0.15,
                                                    batch_size=BATCH_SIZE,
                                                    image_size=IMAGE_SIZE)

>>

Found 8000 files belonging to 2 classes.
Using 1200 files for validation.
```

Створимо змінну, в якій збережемо список з імен класів. Цей список міститься в атрибуті *class_names*, який формується автоматично під час роботи функції *image_dataset_from_directory()*.

```
class_names = train_dataset.class_names
class_names

>>

['cats', 'dogs']
```

Переглянемо перші дев'ять зображень з навчального набору даних (рис. 5.1).

```
plt.figure(figsize=(8, 8))

for images, labels in train_dataset.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]])
        plt.axis("off")

>>
```



Рис. 5.1. Перші дев'ять зображень з навчального набору даних

Аналогічним чином створимо тестовий набір даних. Вкажемо лише адресу папки з тестовими зображеннями (як і для навчальних даних, зображення кожного класу повинні розміщуватись в окремій папці), а також розмір батчу та зображення.

```
test_dataset = image_dataset_from_directory('../test_set',
                                             batch_size=BATCH_SIZE,
                                             image_size=IMAGE_SIZE)

>>
Found 2000 files belonging to 2 classes.
```

Перевіримо, чи правильно зчитались імена класів.

```
test_dataset.class_names  
  
>>  
['cats', 'dogs']
```

Все добре. Далі необхідно написати службовий код, який оптимізує процедуру роботи з відеокартою.

```
AUTOTUNE = tf.data.experimental.AUTOTUNE  
  
train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)  
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)  
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

Створюємо модель згорткової нейронної мережі. Оберемо наступну архітектуру:

- Службовий прошарок для стандартизації значень в пікселях зображення (переведення з діапазону 0-255 до діапазону 0-1).
- Вхідний згортковий прошарок (кількість фільтрів - 32, розмір фільтру - 5x5, padding - однаковий з усіх країв, розмірність вхідного зображення - 180x180x3 [палітра RGB має три кольорових канали], активаційна функція - ReLU).
- Прошарок підвибірки (розмір фільтру - 2x2).
- Прошарок дропауту для регуляризації (випадково вимикаємо 20% нейронів попереднього прошарку)
- Другий згортковий прошарок (кількість фільтрів - 64, розмір фільтру - 3x3, padding – однаковий, активаційна функція – ReLU).
- Прошарок підвибірки (розмір фільтру – 2x2).
- Прошарок дропауту.
- Третій згортковий прошарок (кількість фільтрів – 128, розмір фільтру – 3x3, padding – однаковий, активаційна функція – ReLU).
- Прошарок підвибірки (розмір фільтру – 2x2).

- Прошарок дропауту.
- Четвертий згортковий прошарок (кількість фільтрів – 256, розмір фільтру – 3x3, padding – однаковий, активаційна функція – ReLU).
- Прошарок підвибірки (розмір фільтру – 2x2).
- Прошарок дропауту.
- Службовий прошарок для перетворення набору ознак у одновимірний вектор.
- Повнозв'язний прошарок для класифікації (512 нейронів, активаційна функція – ReLU, регуляризація ваг – L2).
- Прошарок дропауту.
- Вихідний прошарок з одним нейроном, значення якого може бути 0 (коти) або 1 (собаки) (активаційна функція – сигмоїдальна).

```
# Створюємо послідовну модель
model = Sequential()

# Додаємо прошарок стандартизації значень пікселів
model.add(Rescaling(scale=1./255))

# Згортковий прошарок
model.add(Conv2D(32, (5, 5), padding='same',
                  input_shape=(180, 180, 3), activation='relu'))
# Прошарок підвибірки
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

# Згортковий прошарок
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
# Прошарок підвибірки
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

# Згортковий прошарок
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
# Прошарок підвибірки
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
```



```

# Згортковий прошарок
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
# Прошарок підвибірки
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

# Повнозв'язна частина нейронної мережі для класифікації
model.add(Flatten())
model.add(Dense(512, activation='relu', kernel_regularizer='l2'))
model.add(Dropout(0.2))

# Вихідний прошарок, 1 нейрон
model.add(Dense(1, activation='sigmoid'))

```

Компілюємо модель. Критерій якості – бінарна крос-ентропія (оскільки в задачі рівно 2 класи), оптимізатор – Adam, метрика – Accuracy.

```

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

```

Переглянемо зведену інформацію щодо архітектури моделі. Як бачимо, задана мережа має 16 250 689 внутрішніх параметрів.

```

model.build((1, 180, 180, 3))
model.summary()

```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
rescaling_3 (Rescaling)	(1, 180, 180, 3)	0
conv2d_12 (Conv2D)	(1, 180, 180, 32)	2432
max_pooling2d_12 (MaxPooling)	(1, 90, 90, 32)	0
dropout_15 (Dropout)	(1, 90, 90, 32)	0
conv2d_13 (Conv2D)	(1, 90, 90, 64)	18496
max_pooling2d_13 (MaxPooling)	(1, 45, 45, 64)	0

dropout_16 (Dropout)	(1, 45, 45, 64)	0
conv2d_14 (Conv2D)	(1, 45, 45, 128)	73856
max_pooling2d_14 (MaxPooling)	(1, 22, 22, 128)	0
dropout_17 (Dropout)	(1, 22, 22, 128)	0
conv2d_15 (Conv2D)	(1, 22, 22, 256)	295168
max_pooling2d_15 (MaxPooling)	(1, 11, 11, 256)	0
dropout_18 (Dropout)	(1, 11, 11, 256)	0
flatten_3 (Flatten)	(1, 30976)	0
dense_6 (Dense)	(1, 512)	15860224
dropout_19 (Dropout)	(1, 512)	0
dense_7 (Dense)	(1, 1)	513
=====		
Total params: 16,250,689		
Trainable params: 16,250,689		
Non-trainable params: 0		

Навчаємо нейронну мережу. Вказуємо навчальний набір даних, набір даних для валідації та кількість епох.

```
history = model.fit(train_dataset,
                    validation_data=validation_dataset,
                    epochs=40)

Epoch 1/40
107/107 [=====] - 18s 170ms/step - loss:
1.7990 - accuracy: 0.4946 - val_loss: 0.7111 - val_accuracy: 0.492
5
Epoch 2/40
107/107 [=====] - 18s 169ms/step - loss:
0.6970 - accuracy: 0.5000 - val_loss: 0.6935 - val_accuracy: 0.492
5
Epoch 3/40
107/107 [=====] - 18s 169ms/step - loss:
0.6933 - accuracy: 0.4985 - val_loss: 0.6933 - val_accuracy: 0.492
5
```

```

...
Epoch 38/40
107/107 [=====] - 18s 169ms/step - loss:
0.4487 - accuracy: 0.8156 - val_loss: 0.4714 - val_accuracy: 0.802
5
Epoch 39/40
107/107 [=====] - 18s 170ms/step - loss:
0.4483 - accuracy: 0.8196 - val_loss: 0.4874 - val_accuracy: 0.805
0
Epoch 40/40
107/107 [=====] - 18s 170ms/step - loss:
0.4362 - accuracy: 0.8204 - val_loss: 0.4878 - val_accuracy: 0.800
0

```

Побудуємо графіки навчання (рис. 5.2).

```

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()

>>

```

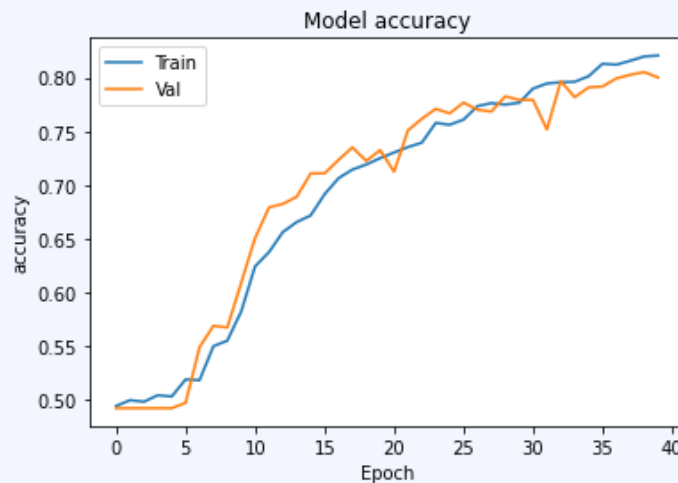


Рис. 5.2. Якість навчання для кожної епохи:

— навчальна множина; — валідаційна множина

Перенавчання немає.

Виконаємо оцінку якості роботи мережі на тестових даних.

```
scores = model.evaluate(test_dataset)

32/32 [=====] - 9s 295ms/step - loss: 0.4963
- accuracy: 0.7945

print("Доля вірних відповідей на тестових даних, у відсотках:",
      round(scores[1] * 100, 4))

>>

Доля вірних відповідей на тестових даних, у відсотках: 79.45
```

Застосуємо створену модель для класифікації довільного зображення. Імпортуємо функції для завантаження зображення в програму.

```
from IPython.display import Image
from tkinter.filedialog import askopenfilename
```

Виконаємо попередню обробку (масштабування та перетворення зображення в масив Numpy) та подамо його на вхід мережі для отримання відповіді. Виведемо результат (рис. 5.3).

```
# Викликаємо віконце для вибору файлу
img_path = askopenfilename()
# Імпортуємо зображення та масштабуємо його (img_path, ширина=180, висота=180)
img = image.load_img(img_path, target_size=(180, 180))

# Перетворюємо зображення в масив
x = image.img_to_array(img)
x = x.reshape(-1, 180, 180, 3)

# Подаємо зображення на вхід мережі для класифікації
prediction = model.predict(x)

# Визначаємо код класу
prediction = int(np.round(prediction))

# Показуємо результат
plt.figure(figsize=(4, 4))
plt.imshow(img)
plt.title(f"Назва класу: {class_names[prediction]}")
plt.axis("off")

(-0.5, 179.5, 179.5, -0.5)
```

Назва класу: dogs



Рис. 5.3. Результат роботи мережі

5.2. Завдання для самостійного виконання

Загальні завдання для всіх варіантів:

1. Завантажте навчальний набір даних. Сформууйте навчальну та тестову підмножини зображень. Самостійно визначте оптимальний розмір зображень та розмір батчу.
2. Виведіть перші 9 зображень із сформованої навчальної підмножини.
3. Перевірте, чи правильно сформовано імена класів.
4. Сформууйте множину тестових даних. Перевірте правильність імен класів на тестовій множині.
5. Створіть та навчіть згорткову нейронну мережу для розпізнавання двох класів об'єктів на зображеннях. Побудуйте графік навчання.
6. Проведіть оцінку якості роботи навченої мережі на тестових даних.
7. Застосуйте навчену мережу для класифікації довільних зображень, які імпортуються в програму з жорсткого диску ПК.

Варіант 1

Набір даних: мавпочки (monkeys.zip).

Опис даних: В наборі містяться зображення двох різних видів мавп. Необхідно навчити нейронну мережу визначати, до якого виду належить мавпа на зображенні.

Варіант 2

Набір даних: чужий/хижак (alien.zip).

Опис даних: В наборі містяться зображення Чужих та Хижаків. Необхідно навчити нейронну мережу визначати, яка істота зображена на рисунку.

Варіант 3

Набір даних: коти і панди (animals.zip).

Опис даних: В наборі містяться зображення котів та панд. Необхідно навчити нейронну мережу визначати, яка істота зображена на фотографії.

Варіант 4

Набір даних: Сімпсони (simpsons.zip).

Опис даних: В наборі містяться зображення персонажів з мультфільму «Сімпсони». Необхідно навчити нейронну мережу визначати, хто зображений на рисунку – Гомер чи Барт Сімпсон.

Варіант 5

Набір даних: види спорту (sports.zip).

Опис даних: В наборі містяться зображення з футбольних та баскетбольних матчів. Необхідно навчити нейронну мережу визначати, який вид спорту показано на фотографії.

Варіант 6

Набір даних: породи собак (dogs.zip).

Опис даних: В наборі містяться зображення двох різних порід собак. Необхідно навчити нейронну мережу визначати, до якої породи належить собака на зображенні.

Варіант 7

Набір даних: сортування сміття (garbage.zip).

Опис даних: В наборі містяться зображення двох різних типів сміття. Необхідно навчити нейронну мережу визначати, до якого типу належить об'єкт на зображенні – скло чи метал.

Варіант 8

Набір даних: погода (weather.zip).

Опис даних: В наборі містяться зображення різних типів погоди. Необхідно навчити нейронну мережу визначати, яка погода показана на зображенні – дощова чи сонячна.

Варіант 9

Набір даних: човни (boats.zip).

Опис даних: В наборі містяться зображення різних типів човнів. Необхідно навчити нейронну мережу визначати, який човен показано на зображенні – вітрильник чи круїзний лайнер.

Варіант 10

Набір даних: рентгенографія COVID-19 (covid.zip).

Опис даних: В наборі містяться рентгенівські знімки легень здорових людей та хворих на COVID-19. Необхідно навчити нейронну мережу визначати, знімок якої людини показано на зображенні – хворої чи здорової.

Варіант 11

Набір даних: дика природа Орегону (wildlife.zip).

Опис даних: В наборі містяться зображення різних тварин, які водяться в штаті Орегон, США. Необхідно навчити нейронну мережу визначати, яку тварину показано на зображенні – ведмедя чи єнота.

Варіант 12

Набір даних: літаки (planes.zip).

Опис даних: В наборі містяться зображення різних типів літаків. Необхідно навчити нейронну мережу визначати, який літак показано на зображенні – пасажирський чи винищувач.

ПРАКТИКУМ 6. АУГМЕНТАЦІЯ ДАНИХ

6.1. Загальні відомості

Для роботи з глибинними згортковими нейронними мережами необхідно мати набори зображень, які містять сотні та тисячі зразків кожного класу. В реальності буває досить складно знайти такий обсяг інформації. Тому для штучного збільшення обсягу навчальної вибірки використовують підхід, який називається аугментацією (доповненням) даних. Доповнення відбувається завдяки виконанню різних операцій над початковими зображеннями - їх масштабують, повертають на деякий кут, віддзеркалюють тощо. Таким чином, на основі одного зображення можна створити декілька його видозмінених копій. Аугментація не дозволить суттєво покращити ефективність мережі, але якість її навчання стане відчутно кращою у порівнянні із навчанням на початковій (не доповненій) вибірці.

Зазвичай аугментація застосовується у випадках, коли у початковому наборі даних не більше 200-300 зображень кожного класу. Але потрібно пам'ятати, що такий метод не завжди можна застосувати. Наприклад, якщо розпізнаються номери автомобілів, то віддзеркалення зображень з метою аугментації навпаки погіршить якість навчання. Оскільки цифри номеру, як і будь-яка інша текстова інформація, втрачають весь свій зміст та характерні ознаки у випадку віддзеркалення.

- В Keras аугментація здійснюється за допомогою генератора зображень ImageDataGenerator:

<https://keras.io/api/preprocessing/image/#imagedatagenerator-class>

Розглянемо приклад роботи з генератором даних в завданні класифікації героїв коміксів Marvel.

Проведемо імпорт необхідних модулів та функцій, включно з генератором зображень ImageDataGenerator.

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import Sequential
```

```

from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Conv2D, Flatten, Dense, MaxPooling2D, Dropout
from tensorflow.keras.regularizers import l2
from tensorflow.keras.optimizers import Adam
from keras import utils
from keras.preprocessing import image

import matplotlib.pyplot as plt

```

Оптимізуємо режим роботи відеокарти.

```

from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession

config = ConfigProto()
config.gpu_options.per_process_gpu_memory_fraction = 0.9
session = InteractiveSession(config=config)

physical_devices = tf.config.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(physical_devices[0], True)

```

Задамо початкові параметри, такі як адреси папок із зображеннями та розміри зображень. Зверніть увагу, що на відміну від використання функції `image_dataset_from_directory()`, у випадку застосування генератора зображень потрібно попередньо розмістити навчальні зразки у трьох різних каталогах: дані для навчання, дані для валідації та дані для тестування. Також на цьому етапі задамо розмір батчу та створимо список із назвами класів.

```

# Каталог з даними для навчання
train_dir = '../marvel/train'
# Каталог з даними для валідації
val_dir = '../marvel/valid'
# Каталог з даними для тестування
test_dir = '../marvel/test'
# Розміри зображення
img_width, img_height = 256, 256
# Розмірність тензору на основі зображення для подання на вхід
нейронної мережі
# backend Tensorflow, channels_last (три канали, тому що зображення
кольорове)
input_shape = (img_width, img_height, 3)

```

```
# Розмір міні-вибірки
batch_size = 32

#Назви класів
classes = ['black widow', 'captain america', 'doctor strange',
           'hulk', 'ironman', 'loki', 'spider-man', 'thanos']
```

Створюємо об'єкт-генератор зображень *ImageDataGenerator()* для навчальної вибірки. Даний об'єкт дозволить одразу під час зчитування зображень з каталогу здійснити над ними деякі перетворення. Наприклад, опція *rescale=1. / 255* виконає стандартизацію значень пікселів зображення. Інші параметри даного генератора використовуються для аугментації даних. Наприклад, задамо наступні налаштування:

- *rotation_range=15* - обертання зображення на максимум 15% від початкового положення;
- *width_shift_range=0.2* - зсув зображення по вертикалі на максимум 20% від початкового положення;
- *height_shift_range=0.2* - зсув зображення по горизонталі на максимум 20% від початкового положення;
- *zoom_range=0.2* - збільшення зображення на максимум 20%;
- *horizontal_flip=True* - віддзеркалення зображення за горизонталлю;
- *fill_mode='nearest'* - режим заповнення пустих пікселів, що утворились внаслідок зсуву або обертання зображення. Значення *nearest* вказує на те, що пусті ділянки будуть заповнені значеннями граничних пікселів початкового зображення.

```
train_datagen = ImageDataGenerator(rescale=1. / 255,
                                   rotation_range=15,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True,
                                   fill_mode='nearest')
```

Аналогічним чином створимо генератор, який буде використовуватись для валідаційної та тестової множини. Однак, під час валідації або тестування аугментація не застосовується. Тому єдиним параметром генератора буде *rescale=* для стандартизації вхідних значень.

```
test_datagen = ImageDataGenerator(rescale=1. / 255)
```

Тепер створимо необхідні набори даних за допомогою методу *flow_from_directory()* генератора. В якості параметрів вказуємо адресу каталога з зображеннями, розмір зображення для його автоматичного масштабування (має відповідати формі входу нейронної мережі), розмір батчу та режим міток класів (в нашому випадку *categorical*, оскільки в завданні більше 2 класів).

```
# Генератор для зображень із папки для навчання
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical')

# Генератори для тестової та валідаційної вибірки
val_generator = test_datagen.flow_from_directory(
    val_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical')

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical')

>>

Found 1660 images belonging to 8 classes.
Found 305 images belonging to 8 classes.
Found 197 images belonging to 8 classes.
```

Тепер перевіримо роботу генератора навчальних зображень. Завантажимо довільне зображення із навчального каталогу (рис. 6.1).

```
image_file_name = train_dir + '/ironman/pic_001.jpg'  
img = image.load_img(image_file_name, target_size=(256, 256))  
plt.imshow(img)
```

```
>>
```

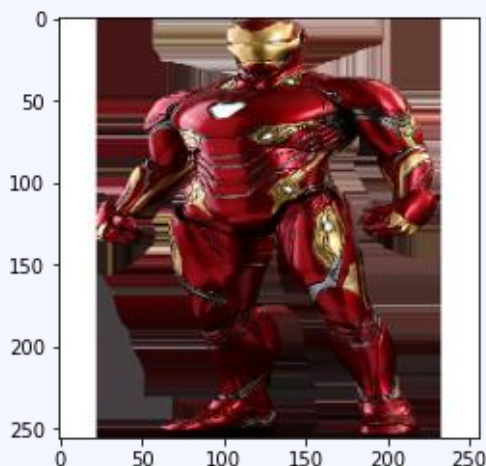


Рис. 6.1. Зображення із навчального каталогу

Тестуємо генератор. Виведемо перші 4 зображення із згенерованого батчу (рис. 6.2).

```
x = image.img_to_array(img)  
x = x.reshape((1,) + x.shape)  
i = 0  
for batch in train_datagen.flow(x, batch_size=1):  
    plt.figure(i)  
    imgplot = plt.imshow(image.array_to_img(batch[0]))  
    i += 1  
    if i % 4 == 0:  
        break  
plt.show()  
  
>>
```

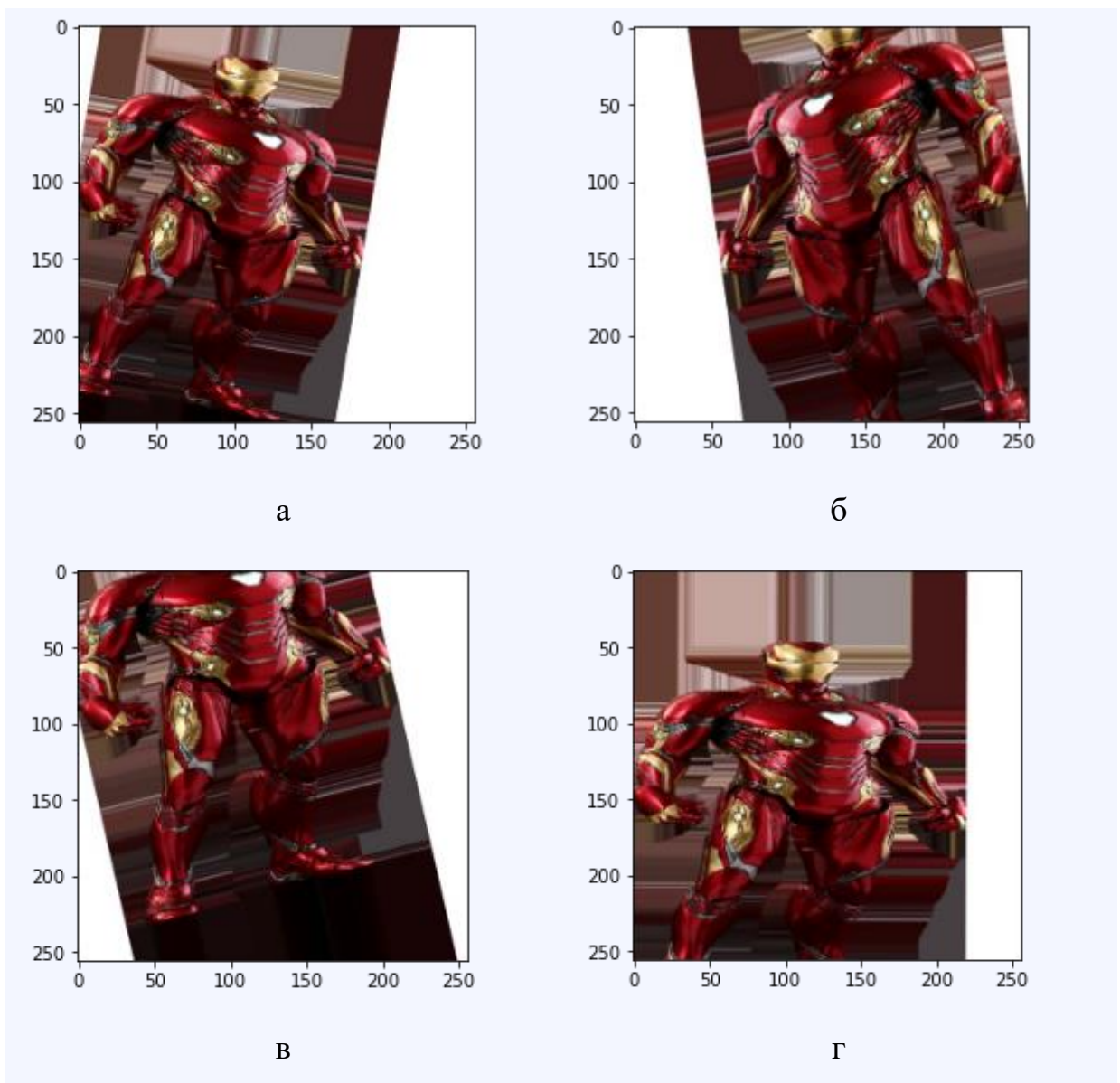


Рис. 6.2. Результат аугментації вхідних даних:

а – поворот зображення; б – поворот + віддзеркалювання;

в,г – поворот + зсув+zoom

Після того, як навчальні дані підготовлені, можна створити модель нейронної мережі.

```
# Створюємо послідовну модель
model = Sequential()

# Згортковий прошарок
model.add(Conv2D(32, (5, 5), padding='same',
```

```

        input_shape=input_shape, activation='relu'))
# Прошарок підвибірки
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

# Згортковий прошарок
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
# Згортковий прошарок
# Прошарок підвибірки
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

# Згортковий прошарок
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
# Згортковий прошарок
# Прошарок підвибірки
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

# Згортковий прошарок
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
# Прошарок підвибірки
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

# Повнозв'язна частина нейронної мережі для класифікації
model.add(Flatten())
model.add(Dense(512, activation='relu', kernel_regularizer='l2'))
model.add(Dropout(0.5))

# Вихідний прошарок, 8 нейронів (за кількістю класів)
model.add(Dense(8, activation="softmax"))

# Компілюємо модель. В якості оптимізатора вказуємо Adam з
модифікованим значенням швидкості навчання
model.compile(loss="categorical_crossentropy", optimizer=Adam(learning_rate=1e-5), metrics=["accuracy"])

```

Навчаємо описану нейронну мережу. У методі *fit()* вказуємо наступні атрибути:

- *train_generator* - посилання на об'єкт-генератор навчальних даних;

- *steps_per_epoch* = 40 - кількість батчів за одну епоху. Повинна приблизно дорівнювати (кількість картинок в навчальному каталозі / розмір батчу);
- *epochs=150* - кількість епох, визначається емпірично;
- *validation_data=val_generator* - посилання на об'єкт-генератор даних для валідації;
- *validation_steps= 10* - кількість батчів за одну епоху. Повинна приблизно дорівнювати (кількість картинок в каталозі для валідації / розмір батчу).

```
history = model.fit(
    train_generator,
    steps_per_epoch = 40,
    epochs=150,
    validation_data=val_generator,
    validation_steps= 10)

>>

Epoch 1/150
40/40 [=====] - 34s 857ms/step - loss: 12.2887
- accuracy: 0.1719 - val_loss: 11.3672 - val_accuracy: 0.1705
40/40 [=====] - 53s 1s/step - loss: 12.0474 -
accuracy: 0.1450 - val_loss: 11.8009 - val_accuracy: 0.1672
Epoch 2/150
40/40 [=====] - 34s 857ms/step - loss: 11.6173
- accuracy: 0.1442 - val_loss: 11.3672 - val_accuracy: 0.1705
Epoch 3/150
40/40 [=====] - 30s 750ms/step - loss: 11.1889
- accuracy: 0.1293 - val_loss: 10.9583 - val_accuracy: 0.1344
...
Epoch 148/150
40/40 [=====] - 29s 730ms/step - loss: 1.8596
- accuracy: 0.4538 - val_loss: 1.8551 - val_accuracy: 0.4820
Epoch 149/150
40/40 [=====] - 30s 756ms/step - loss: 1.8448
- accuracy: 0.4397 - val_loss: 1.8562 - val_accuracy: 0.4820
Epoch 150/150
40/40 [=====] - 29s 735ms/step - loss: 1.8542
- accuracy: 0.4569 - val_loss: 1.8553 - val_accuracy: 0.4623
```


Побудуємо графік навчання (рис. 6.3).

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()
```

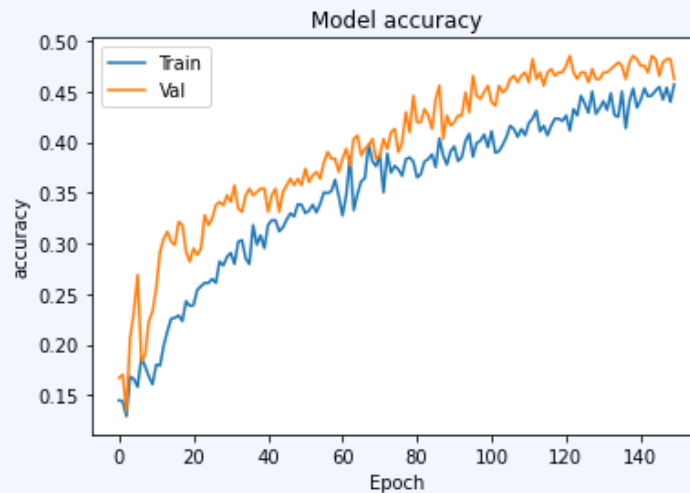


Рис. 6.3. Якість навчання для кожної епохи:

— навчальна множина; — валідаційна множина

Перенавчання немає. Проведемо оцінку якості роботи мережі на тестовій множині, вказавши у методі `evaluate()` посилання на генератор даних для тестування.

```
scores = model.evaluate(test_generator)
print(f"Доля правильних відповідей на тестових даних:
      {(scores[1]*100):.2f}")
>>

7/7 [=====] - 3s 430ms/step - loss: 1.9899
- accuracy: 0.3909
Доля правильних відповідей на тестових даних: 39.09
```

Як бачимо, мережа демонструє не найкращі показники на тестовій множині. Для покращення результатів необхідно використовувати більш складну архітектуру мережі.

Застосуємо навчену мережу для розпізнавання довільного зображення. Виведемо результат (рис. 6.4).

```
from IPython.display import Image
from tkinter.filedialog import askopenfilename

# Викликаємо віконце для вибору файлу
img_path = askopenfilename()

# Імпортуємо зображення та масштабуємо його (img_path, ширина=256, в
исота=256)
img = image.load_img(img_path, target_size=(256, 256))

# Перетворюємо зображення в масив
x = image.img_to_array(img)
x = x.reshape(-1, 256, 256, 3)

# Подаємо зображення на вхід мережі для класифікації
prediction = model.predict(x)

# Визначаємо код класу
prediction = np.argmax(prediction)

# Показуємо результат
plt.figure(figsize=(4, 4))
plt.imshow(img)
plt.title(f"Назва класу: {classes[prediction]}")
plt.axis("off")

>>
```

Назва класу: hulk



Рис. 6.3. Результат роботи мережі

6.2. Завдання для самостійного виконання

Загальні завдання для всіх варіантів:

1. Завантажте навчальний набір даних за допомогою генератора. Сформууйте навчальну і тестову (та валідаційну – за бажанням) множини. Самостійно визначте оптимальний розмір зображень та розмір батчу. Під час формування навчальної множини використовуйте одну або декілька доступних опцій генератора для аугментації даних.
2. Перевірте роботу генератора на довільному одному зображенні.
3. Створіть та навчіть згорткову нейронну мережу для розпізнавання класів об'єктів на зображеннях. Побудуйте графік навчання.
4. Проведіть оцінку якості роботи навченої мережі на тестових даних.
5. Застосуйте навчену мережу для класифікації довільних зображень, які імпортуються в програму з жорсткого диску ПК.

Варіант 1

Набір даних: літаки (planes.zip).

Опис даних: В наборі містяться зображення різних типів літаків. Необхідно навчити нейронну мережу визначати, який літак показано на зображенні.

Варіант 2

Набір даних: дика природа Орегону (wildlife.zip).

Опис даних: В наборі містяться зображення різних тварин, які водяться в штаті Орегон, США. Необхідно навчити нейронну мережу визначати, яку тварину показано на зображенні.

Варіант 3

Набір даних: рентгенографія COVID-19 (covid.zip).

Опис даних: В наборі містяться рентгенівські знімки легень здорових людей та хворих на COVID-19 або іншу легеневу хворобу. Необхідно навчити нейронну мережу визначати діагноз людини, знімок легень якої показано на зображенні.

Варіант 4

Набір даних: човни (boats.zip).

Опис даних: В наборі містяться зображення різних типів човнів. Необхідно навчити нейронну мережу визначати, який човен показано на зображенні.

Варіант 5

Набір даних: погода (weather.zip).

Опис даних: В наборі містяться зображення різних типів погоди. Необхідно навчити нейронну мережу визначати, яка погода показана на зображенні.

Варіант 6

Набір даних: сортування сміття (garbage.zip).

Опис даних: В наборі містяться зображення різних типів сміття. Необхідно навчити нейронну мережу визначати, до якого типу належить об'єкт на зображенні.

Варіант 7

Набір даних: індійська їжа (food.zip).

Опис даних: В наборі містяться зображення різних страв індійської кухні. Необхідно навчити нейронну мережу визначати, яка страва зображена на рисунку.

Варіант 8

Набір даних: породи собак (dogs.zip).

Опис даних: В наборі містяться зображення різних порід собак. Необхідно навчити нейронну мережу визначати, до якої породи належить собака на зображенні.

Варіант 9

Набір даних: види спорту (sports.zip).

Опис даних: В наборі містяться зображення різних спортивних матчів. Необхідно навчити нейронну мережу визначати, який вид спорту показано на фотографії.

Варіант 10

Набір даних: мавпочки (monkeys.zip).

Опис даних: В наборі містяться зображення різних видів мавп. Необхідно навчити нейронну мережу визначати, до якого виду належить мавпа на зображенні.

Варіант 11

Набір даних: тварини (animals.zip).

Опис даних: В наборі містяться зображення котів, собак та панд. Необхідно навчити нейронну мережу визначати, яка тварина зображена на фотографії.

Варіант 12

Набір даних: Сімпсони (simpsons.zip).

Опис даних: В наборі містяться зображення персонажів з мультфільму «Сімпсони». Необхідно навчити нейронну мережу визначати, хто саме зображений на рисунку.

ПРАКТИКУМ 7. ПРИЙОМИ ГЛИБИННОГО НАВЧАННЯ

7.1. Підготовка власного набору даних

Для роботи зі згортковими нейронними мережами доводиться використовувати навчальні набори даних, які складаються з декількох тисяч зображень. Підготовка навчальних даних в такому випадку є відповідальним та затратним етапом. З метою пришвидшення цього процесу, розбиття початкового набору даних на навчальну, валідаційну та тестову множини можна автоматизувати. Розглянемо приклад, як це зробити за допомогою засобів Python.

Завдання полягає у наступному: необхідно навчити нейронну мережу розпізнавати персонажів фільмів "Зоряні війни". Початковий набір даних представлений у вигляді базового каталогу, що містить папки із зображеннями персонажів, які відносяться до різних класів. Необхідно на базі цього початкового набору сформувати навчальну, валідаційну та тестову підмножини даних. Кожна підмножина повинна складатись з основного каталогу, в якому у різних папках будуть розміщені зображення відповідних класів.

Для початку роботи, імпортуємо модулі для роботи з файловою системою.

```
import shutil
import os
```

Задамо значення необхідних змінних, долю валідаційної та тестової підмножини відносно початкового набору даних, а також імпортуємо імена класів. Пам'ятаємо, що папки в кожному головному каталозі повинні мати назву, яка відповідає конкретному класу, зображення якого розміщені всередині папки.

```
# Каталог з початковим набором даних
data_dir = '../star_wars/images'
# Каталог з даними для навчання
train_dir = '../star_wars/train'
# Каталог з даними для валідації
val_dir = '../star_wars/valid'
# Каталог з даними для тестування
test_dir = '../star_wars/test'
```

```
# Доля набору даних для валідації
val_data_portion = 0.15
# Доля набору даних для тестування
test_data_portion = 0.15

# Імпортуємо назви класів (як назви папок з початкового каталогу)
classes = os.listdir(data_dir)
```

Оголосимо функцію для створення каталогів. Дана функція створить головний каталог підмножини, а також пусті папки для кожного із класів.

```
def create_directory(dir_name):
    if os.path.exists(dir_name):
        shutil.rmtree(dir_name)
    os.makedirs(dir_name)
    for name in classes:
        os.makedirs(os.path.join(dir_name, name))
```

Використаємо написану функцію для створення каталогів для навчальної, валідаційної та тестової підмножин.

```
create_directory(train_dir)
create_directory(val_dir)
create_directory(test_dir)
```

Напишемо функцію для копіювання зображень із початкового набору даних до відповідного каталогу. Функція прийматиме наступні аргументи:

- *start_indexes* - список, який містить номери зображень, починаючи з яких потрібно виконувати копіювання (для кожного класу).
- *end_indexes* - список, який містить номери зображень, якими треба завершити копіювання (для кожного класу);
- *source_dir* - адреса початкового каталогу (з якого копіювати);
- *dest_dir* - адреса каталогу призначення (до якого копіювати).

Наприклад, у деякому початковому каталозі *source* є дві папки (два класи), які містять по 100 зображень. Потрібно сформувати валідаційну підмножину, яка буде

розміщена в каталозі *destination*. В якості вибірки валідації обмеремо 15 останніх зображень із початкового набору (для кожного класу). Тоді аргументами створеної функції *copy_images()* будуть:

```
copy_images(start_indexes = [74, 74], [99,99], source, destination)
```

Варто зазначити, що списки із початковими та кінцевими індексами будуть формуватись автоматично за допомогою функції, яку ми напишемо на наступному етапі.

```
def copy_images(start_indexes, end_indexes, source_dir, dest_dir):  
    k = 0  
    for name in classes:  
        curr_dir = source_dir + '/' + name  
        files = os.listdir(curr_dir)  
  
        for i, file in enumerate(files):  
            if start_indexes[k] <= i < end_indexes[k]:  
                shutil.copy2(os.path.join(curr_dir, file),  
                             os.path.join(dest_dir, name))  
            k += 1
```

Тепер створимо функцію для формування підмножин даних. Така функція прийматиме наступні аргументи:

- *source_dir* - адреса початкового каталогу;
- *dest_dir* - адреса каталогу призначення
- *subset* - підмножина (навчальна, валідаційна або тестова / 'train', 'val', 'test').

Всередині функції *create_dataset()* автоматично визначаються індекси початкових та кінцевих зображень (кожного класу) для кожної підмножини, а потім викликається раніше створена функція *copy_images()* для копіювання зображень із початкового набору до відповідного каталогу вказаної підмножини.

```
def create_dataset(source_dir, dest_dir, subset):  
    start_val_data_idx = []  
    start_test_data_idx = []  
    end_data_idx = []
```

```

if subset == 'train':
    for name in classes:
        nb_of_images = int(len(os.listdir(source_dir + '/' +
                                           name)))
        start_val_data_idxes.append(nb_of_images * (1 -
                                                    val_data_portion - test_data_portion))

    copy_images([0]*len(classes), start_val_data_idxes,
                source_dir, dest_dir)

elif subset == 'val':
    for name in classes:
        nb_of_images = int(len(os.listdir(source_dir + '/' +
                                           name)))
        start_val_data_idxes.append(int(nb_of_images * (1 -
                                                    val_data_portion - test_data_portion)))
        start_test_data_idxes.append(int(nb_of_images * (1 -
                                                         test_data_portion)))

    copy_images(start_val_data_idxes, start_test_data_idxes,
                source_dir, dest_dir)

elif subset == 'test':
    for name in classes:
        nb_of_images = int(len(os.listdir(source_dir + '/' +
                                           name)))
        start_test_data_idxes.append(int(nb_of_images * (1 -
                                                         test_data_portion)))
        end_data_idxes.append(nb_of_images)

    copy_images(start_test_data_idxes, end_data_idxes,
                source_dir, dest_dir)

else:
    print('subset має бути "train", "val" або "test"!')

```

Застосуємо функцію *create_dataset()* для створення всіх необхідних підмножин даних (навчальна, валідаційна, тестова). В результаті, будуть створені відповідні каталоги, які матимуть структуру, необхідну для подальшої роботи з нейронною мережею.

```
create_dataset(data_dir, train_dir, 'train')
create_dataset(data_dir, val_dir, 'val')
create_dataset(data_dir, test_dir, 'test')
```

7.2. Перенесення навчання

Навчання глибоких нейронних мереж може займати декілька днів і навіть тижнів. В такому випадку, у разі незадовільного результату, ціна помилки буде дуже великою. Тому що на перенавчання буде витрачено занадто багато часу. У зв'язку з цим, популярним є підхід, який називається *transfer learning* - перенесення навчання. Він полягає в тому, що вже колись навчена нейронна мережа з відомою архітектурою використовується для вирішення нових задач, тобто не тих, для яких вона навчалась. Це досягається завдяки наступним крокам:

- Імпортується навчена мережа з відомою архітектурою.
- З мережі видаляються останні прошарки, які відповідають за класифікацію (так звана "вершина" (top, head) мережі).
- Замість видалених прошарків створюються нові. Архітектура цих нових прошарків має відповідати новій задачі, яку повинна буде вирішувати нейронна мережа.

Чому це працює? Справа в тому, що навчена нейронна мережа вже вміє якісно виділяти значущі ознаки із зображення. Тобто немає сенсу знову навчати її знаходити аналоги ознак Хаара - мережа і так вміє це робити, оскільки всі значення синаптичних ваг згорткових прошарків були встановлені під час попереднього навчання. Єдине, що залишається, це навчити мережу правильно класифікувати об'єкти за тим набором ознак, які вона виділить із зображення. Таким чином, у разі використання *transfer learning* навчається лише та частина мережі, яка відповідає за класифікацію. Це дозволяє значно пришвидшити навчання, а також покращити його якість (оскільки показники ефективності роботи мережі відомі заздалегідь).

Застосуємо підхід *transfer learning* для навчання нейронної мережі для класифікації персонажів фільмів "Зоряні війни". В якості попередньо навченої

мережі імпортуємо мережу VGG16 з набором синаптичних ваг, отриманих в результаті навчання на наборі даних *ImageNet*. Цей набір містить 1000 різних класів об'єктів, і мережа VGG16 вже вміє проводити їх класифікацію з високою достовірністю. Але в даній задачі ми будемо застосовувати мережу VGG16 для класифікації всього лише 8 типів об'єктів. Більше того, дані об'єкти не зустрічались мережі під час попереднього навчання. Тому прошарки мережі, які відповідають за класифікацію, потрібно буде відкинути, і замість них створити та навчити свої власні.

Для початку, імпортуємо необхідні модулі (серед яких і сама мережа VGG16).

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dropout, Flatten, Dense
from tensorflow.keras.optimizers import Adam
from keras.preprocessing import image

from tensorflow.keras.applications import VGG16

import matplotlib.pyplot as plt
```

Задамо оптимальні параметри роботи відеокарти.

```
from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession

config = ConfigProto()
config.gpu_options.per_process_gpu_memory_fraction = 0.6
session = InteractiveSession(config=config)

physical_devices = tf.config.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(physical_devices[0], True)
```

Вкажемо початкові налаштування.

```

# Каталог з даними для навчання
train_dir = '../star_wars/train'
# Каталог з даними для валідації
val_dir = '../star_wars/valid'
# Каталог з даними для тестування
test_dir = '../star_wars/test'

# Назви класів
classes = os.listdir(train_dir)

# Розміри зображення
img_width, img_height = 150, 150
# Розмірність тензора вхідного зображення для входу нейронної мережі
input_shape = (img_width, img_height, 3)
# Розмір батчу
batch_size = 32

```

Створимо об'єкт мережі VGG16 з наступними параметрами:

- *weights='imagenet'* - набір попередньо навчених ваг: ImageNet
- *include_top=False* - не завантажувати частину мережі, призначену для класифікації
- *input_shape=input_shape* - форма вхідного зображення (в нашому випадку - (150, 150, 3)).

```

vgg16_net = VGG16(weights='imagenet', include_top=False,
                  input_shape=input_shape)

```

Заборонимо навчання всіх завантажених прошарків мережі VGG16, встановивши значення атрибуту *vgg16_net.trainable = False*. Після цього переглянемо зведену інформацію щодо мережі і переконаємось, що кількість внутрішніх параметрів, доступних для навчання, дорівнює нулю (*Trainable params: 0*).

```

vgg16_net.trainable = False
vgg16_net.summary()

>>
Model: "vgg16"

```

Layer (type)	Output Shape	Param #
=====	=====	=====
input_1 (InputLayer)	[(None, 150, 150, 3)]	0
block1_conv1 (Conv2D)	(None, 150, 150, 64)	1792
block1_conv2 (Conv2D)	(None, 150, 150, 64)	36928
block1_pool (MaxPooling2D)	(None, 75, 75, 64)	0
block2_conv1 (Conv2D)	(None, 75, 75, 128)	73856
block2_conv2 (Conv2D)	(None, 75, 75, 128)	147584
block2_pool (MaxPooling2D)	(None, 37, 37, 128)	0
block3_conv1 (Conv2D)	(None, 37, 37, 256)	295168
block3_conv2 (Conv2D)	(None, 37, 37, 256)	590080
block3_conv3 (Conv2D)	(None, 37, 37, 256)	590080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0
block4_conv1 (Conv2D)	(None, 18, 18, 512)	1180160
block4_conv2 (Conv2D)	(None, 18, 18, 512)	2359808
block4_conv3 (Conv2D)	(None, 18, 18, 512)	2359808
block4_pool (MaxPooling2D)	(None, 9, 9, 512)	0
block5_conv1 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv2 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv3 (Conv2D)	(None, 9, 9, 512)	2359808
block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0
=====	=====	=====
Total params: 14,714,688		
Trainable params: 0		
Non-trainable params: 14,714,688		

Створюємо власну нейронну мережу, яка буде складатись з завантаженої частини навченої VGG16 та власного класифікатора.

```
model = Sequential()  
# Додаємо до моделі мережу VGG16 замість прошарку  
model.add(vgg16_net)  
  
# Додаємо до моделі власний класифікатор  
model.add(Flatten())  
model.add(Dense(256, activation='relu'))  
model.add(Dropout(0.2))  
model.add(Dense(len(classes), activation='softmax'))
```

Переглянемо інформацію про створену мережу. Як можна побачити, під час навчання доведеться знайти значення лише 2,098,693 параметрів, тоді як загалом наша модель містить 16,813,381 внутрішніх параметрів. Але з них 14,714,688 параметрів вже були попередньо навчені, їх не потрібно модифіковувати (що ми і заборонили робити на попередніх етапах). Це значно пришвидшить процес навчання.

```
model.summary()  
  
>>  
Model: "sequential"  
  
Layer (type)                Output Shape                Param #  
=====
```

vgg16 (Functional)	(None, 4, 4, 512)	14714688
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 256)	2097408
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 5)	1285

```
=====
```

Total params: 16,813,381		
Trainable params: 2,098,693		
Non-trainable params: 14,714,688		

Компілюємо модель. Оскільки додана частина мережі VGG16 вже була попередньо навчена і вміє гарно виявляти ознаки, необхідно знизити швидкість навчання *learning_rate*. Це зроблено для того, щоб оптимізатор не пропустив мінімум функції втрат.

```
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(learning_rate=1e-5),
              metrics=['accuracy'])
```

Створюємо об'єкт-генератор зображень *ImageDataGenerator()* для автоматичної стандартизації значень пікселів під час імпорту картинки з каталогу.

```
datagen = ImageDataGenerator(rescale=1. / 255)
```

Створюємо та налаштовуємо генератори для навчальної, валідаційної та тестової підмножин.

```
train_generator = datagen.flow_from_directory(
    train_dir, target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical')
```

```
>>
```

```
Found 892 images belonging to 5 classes.
```

```
val_generator = datagen.flow_from_directory(
    val_dir, target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical')
```

```
>>
```

```
Found 189 images belonging to 5 classes.
```

```
test_generator = datagen.flow_from_directory(
    test_dir, target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical')
```

```
>>
```

```
Found 192 images belonging to 5 classes.
```


Запам'ятовуємо кількість зображень у навчальній та валідаційній мережі (знадобиться для коректного визначення кількості крок за епоху навчання).

```
nb_train_samples = 892
nb_validation_samples = 189
```

Навчаємо модель.

```
history = model.fit(
    train_generator,
    steps_per_epoch=nb_train_samples // batch_size,
    epochs=15, validation_data=val_generator,
    validation_steps=nb_validation_samples // batch_size)

>>

Epoch 1/15
27/27 [=====] - 14s 516ms/step - loss: 1.6738
- accuracy: 0.2616 - val_loss: 1.4439 - val_accuracy: 0.4187
Epoch 2/15
27/27 [=====] - 4s 136ms/step - loss: 1.4253
- accuracy: 0.4151 - val_loss: 1.2821 - val_accuracy: 0.5938
Epoch 3/15
27/27 [=====] - 4s 136ms/step - loss: 1.2727
- accuracy: 0.5093 - val_loss: 1.1702 - val_accuracy: 0.6625
...

Epoch 13/15
27/27 [=====] - 4s 138ms/step - loss: 0.5298
- accuracy: 0.8640 - val_loss: 0.6196 - val_accuracy: 0.8000
Epoch 14/15
27/27 [=====] - 4s 139ms/step - loss: 0.4950
- accuracy: 0.8767 - val_loss: 0.6192 - val_accuracy: 0.8250
Epoch 15/15
27/27 [=====] - 4s 136ms/step - loss: 0.4675
- accuracy: 0.8907 - val_loss: 0.5932 - val_accuracy: 0.8188
```

Перевіряємо модель на тестових даних.

```
scores = model.evaluate(test_generator)
print(f"Accuracy на тестових даних: {(scores[1]*100):.2f}")

>>

6/6 [=====] - 2s 368ms/step - loss: 0.6725
- accuracy: 0.7812
Accuracy на тестових даних: 78.12
```

Отже, завдяки застосуванню підходу *transfer learning*, нам вдалось навчити нейронну мережу вирішувати нову задачу з високою ефективністю всього лише за 10-15 епох навчання. Але навіть цей результат не є найкращим. Для його покращення можна застосувати додатковий підхід, який має назву *fine tuning*.

7.3. Тонкі налаштування мережі

Під час використання *transfer learning* виокристовується попередньо навчена нейронна мережа, в якій замінено прошарки, що відповідають за класифікацію. Прошарки для виділення ознак на зображеннях залишаються і "заморожуються" під час навчання. Тобто синаптичні ваги цих прошарків не змінюються, оскільки вони не доступні для навчання. Однак, якщо нове завдання значно відрізняється від того, на якому була попередньо навчена нейронна мережа, деякі з останніх прошарків для виділення ознак можна активувати для навчання. Чим сильніша різниця між попередньою та новою задачею, тим більше останніх прошарків для виділення ознак рекомендується зробити доступними для навчання. Такий підхід називається тонким налаштуванням мережі, або *fine tuning*.

Застосуємо *fine tuning* для покращення якості мережі, отриманої у попередньому прикладі. Для цього дозволимо навчання останнього згорткового прошарку мережі VGG16, який називається *'block5_conv3'*. Навчання всіх інших прошарків залишаємо забороненим.

```
# Активуємо для навчання всю мережу
vgg16_net.trainable = True
# Перебираємо в циклі прошарки та вимикаємо навчання для всіх прошарків, окрім потрібного
trainable = False
for layer in vgg16_net.layers:
    if layer.name == 'block5_conv3':
        trainable = True
    layer.trainable = trainable
```

Перевіряємо кількість параметрів, доступних для навчання. Бачимо, що у порівнянні з попереднім варіантом, їх кількість збільшилась, оскільки прошарок *'block5_conv3'* тепер доступний для навчання.

```
model.summary()

>>
Model: "sequential"

```

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 4, 4, 512)	14714688
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 256)	2097408
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 5)	1285

```

Total params: 16,813,381
Trainable params: 4,458,501
Non-trainable params: 12,354,880

```

Заново компілюємо модифіковану модель.

```
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(lr=1e-5),
              metrics=['accuracy'])
```

Навчаємо модифіковану модель.

```
history = model.fit(
    train_generator,
    steps_per_epoch=nb_train_samples // batch_size,
    epochs=15,
    validation_data=val_generator,
    validation_steps=nb_validation_samples // batch_size)

Epoch 1/15
27/27 [=====] - 4s 151ms/step - loss: 0.4377
- accuracy: 0.8837 - val_loss: 0.5352 - val_accuracy: 0.8375
Epoch 2/15
```

```

27/27 [=====] - 4s 139ms/step - loss: 0.3370
- accuracy: 0.9244 - val_loss: 0.4848 - val_accuracy: 0.8562
Epoch 3/15
27/27 [=====] - 4s 141ms/step - loss: 0.2874
- accuracy: 0.9384 - val_loss: 0.4906 - val_accuracy: 0.8250
...

Epoch 13/15
27/27 [=====] - 4s 142ms/step - loss: 0.0627
- accuracy: 0.9977 - val_loss: 0.3981 - val_accuracy: 0.8438
Epoch 14/15
27/27 [=====] - 4s 141ms/step - loss: 0.0570
- accuracy: 0.9988 - val_loss: 0.3639 - val_accuracy: 0.8500
Epoch 15/15
27/27 [=====] - 4s 145ms/step - loss: 0.0461
- accuracy: 0.9988 - val_loss: 0.3927 - val_accuracy: 0.8687

```

Перевіряємо якість мережі на тестовій підмножині.

```

scores = model.evaluate(test_generator)
print(f"Accuracy на тестових даних: {(scores[1]*100):.2f}")

>>

6/6 [=====] - 1s 100ms/step - loss: 0.4672
- accuracy: 0.8594
Accuracy на тестових даних: 85.94

```

Як видно з результатів, використання *fine tuning* дозволило покращити якість роботи мережі на тестових даних майже на 9% у порівнянні з попереднім варіантом, коли навчався лише класифікатор. Аналогічним чином можна розблокувати для навчання і інші частини (прошарки) початкової мережі, до якої застосовується *transfer learning*. Кількість доступних для навчання прошарків залежить від конкретних задач.

Використаємо навчену нейронну мережу для розпізнавання зображення, яке імпортується з жорсткого диску ПК. Виведемо результат (рис. 7.1).

```

from IPython.display import Image
from tkinter.filedialog import askopenfilename

img_path = askopenfilename()

img = image.load_img(img_path, target_size=(img_width, img_height))

```

```
x = image.img_to_array(img)
x = x.reshape(-1, img_width, img_height, 3)

prediction = model.predict(x)
prediction = np.argmax(prediction)

plt.figure(figsize=(4, 4))
plt.imshow(img)
plt.title(f"Назва класу: {classes[prediction]}")
plt.axis("off")

>>
```



Рис. 7.1. Результат роботи мережі

7.4. Завдання для самостійного виконання

Загальні завдання для всіх варіантів:

1. Створіть власний набір навчальних зображень для згорткової нейронної мережі. Набір має містити зображення як мінімум трьох різних класів об'єктів. Всі зображення в наборі мають бути розподілені на три підмножини – навчальну, валідаційну та тестову. Підмножини мають розміщуватись у окремих папках.
2. Завантажте створений набір даних за допомогою генератора. Сформууйте навчальну, валідаційну і тестову множини. Самостійно визначте оптимальний розмір зображень та розмір батчу. У разі необхідності, під час формування навчальної множини використовуйте аугментацію даних.
3. Перевірте роботу генератора на довільному одному зображенні.
4. Використовуючи підхід transfer learning, створіть та навчіть згорткову нейронну мережу на базі архітектури VGG-16 для розпізнавання класів об'єктів на зображеннях. Побудуйте графік навчання.
5. Проведіть оцінку якості роботи навченої мережі на тестових даних.
6. Застосуйте навчену мережу для класифікації довільних зображень, які імпортуються в програму з жорсткого диску ПК.
7. Спробуйте максимально покращити якість створеної нейронної мережі, використавши тонкі налаштування (fine tuning).

ПРАКТИКУМ 8. РЕКУРЕНТНІ НЕЙРОННІ МЕРЕЖІ

8.1. Загальні відомості

Нейронні мережі прямого розповсюдження є досить ефективними моделями в завданнях аналізу часових послідовностей. Однак, суттєвим недоліком є те, що в такому випадку кожен елемент послідовності аналізується мережею окремо. Іншими словами, нейронна мережа приймає рішення на основі аналізу сукупності вхідних елементів, які ніяк між собою не пов'язані. Це призводить до втрати взаємозв'язків між елементами послідовностей, що є неприпустимими у ряді випадків. Наприклад, найпростішою часовою послідовністю є текст. У тексті всі слова пов'язані між собою певним семантичним змістом. Кожне речення є не просто невпорядкованим набором слів, а складає певну цілісну змістовну конструкцію, яка дозволяє нам зрозуміти думку, що передає цей текст. Тому виникає необхідність у застосуванні таких архітектур нейронних мереж, які будуть враховувати порядок слідування елементів у послідовності. Це дозволяє зберегти характер взаємозв'язків між вхідними даними і значно покращує результат аналізу часових послідовностей.

Найбільш ефективною на сьогоднішній день архітектурою нейронних мереж для аналізу послідовностей є рекурентні нейронні мережі (Recurrent Neural Networks, RNN). Це мережі, які містять зворотні зв'язки та мають внутрішню пам'ять, що дозволяє зберігати інформацію про попередні вхідні зразки. Однією з привабливих ідей RNN є те, що вони потенційно вміють пов'язувати попередню інформацію з поточним завданням. Так, наприклад, знання про попередній кадр відео можуть допомогти в розумінні поточного кадру. За останні кілька років RNN з неймовірним успіхом застосовували до цілого ряду завдань: розпізнавання мови, мовне моделювання, переклад, розпізнавання зображень тощо.

Однак, класичні RNN мають ряд недоліків. Основним із них є те, що такі мережі здатні пам'ятати інформацію лише про 5-10 попередніх елементів. Але, наприклад, текст зазвичай складається із більшої кількості слів. Для вирішення цієї

проблеми застововуються більш складні рекурентні архітектури. Чимала роль в досягненнях RNN належить LSTM - незвичайній модифікації рекуррентної нейронної мережі, яка на багатьох задачах значно перевершує стандартну версію. Майже всі вражаючі результати RNN досягнуті саме за допомогою LSTM.

Довга короткострокова пам'ять (Long short-term memory, LSTM) - особливий різновид архітектури рекурентних нейронних мереж, здатний до вивчення довготривалих залежностей. Вони прекрасно вирішують цілий ряд різноманітних завдань і в даний час широко використовуються. Однак, мережі LSTM є досить складними, тому під час їх навчання виникають деякі проблеми. Існує більш спрощений варіант даної архітектури під назвою Gated recurrent unit (GRU) - керований рекурентний нейрон.

- [Дуже хороша стаття по рекурентним нейронним мережам та їх модифікаціям](#)
- [Рекурентні прошарки в Keras](#)

Підготовка набору даних

Набір даних AG - це колекція з понад 1 мільйона текстів новин. Новини були зібрані з більш ніж 2000 джерел компанією ComeToMyHead за більш ніж 1 рік діяльності. ComeToMyHead - це академічна пошукова система новин, яка працює з липня 2004 року. Отримати додаткову інформацію можна за [посиланням](#).

Набір для класифікації новин за темами складається з заголовків новин, які відносяться до 4 класів (Світ/Спорт/Бізнес/Технології). Кожен клас містить 31900 навчальних зразків. Загальна кількість навчальних зразків - 127 600.

- [Набір даних AG news у форматі .csv](#)

Для початку роботи, імпортуємо необхідні модулі.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, LSTM, GRU
from tensorflow.keras import utils
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
```



```

from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras import utils
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

```

Задамо деякі початкові налаштування.

```

# Максимальна кількість слів
num_words = 10000
# Максимальна довжина новини
max_news_len = 30
# Кількість класів новин
nb_classes = 4
# Зерно датчика випадкових чисел для відтворюваності результатів
np.random.seed(123)

```

Імпортуємо набір даних.

```

df = pd.read_csv("../AG_news.csv", sep=",")
df.head()

>>

```

	class	title \	text
0	3	Wall St. Bears Claw Back Into the Black (Reuters)	Reuters - Short-sellers, Wall Street's dwindli...
1	3	Carlyle Looks Toward Commercial Aerospace (Reu...	Reuters - Private investment firm Carlyle Grou...
2	3	Oil and Economy Cloud Stocks' Outlook (Reuters)	Reuters - Soaring crude prices plus worries\ab...
3	3	Iraq Halts Oil Exports from Main Southern Pipe...	Reuters - Authorities have halted oil export\f...
4	3	Oil prices soar to all-time record, posing new...	AFP - Tearaway world oil prices, toppling reco...

Виділяємо стовпець із текстом новин для формування векторів предикторів.

```

news = df['text']
news.head()

>>

```

```

0 Reuters - Short-sellers, Wall Street's dwindle...
1 Reuters - Private investment firm Carlyle Grou...
2 Reuters - Soaring crude prices plus worries\ab...
3 Reuters - Authorities have halted oil export\f...
4 AFP - Tearaway world oil prices, toppling reco...

```

Виділяємо стовпець із номерами класів для формування векторів цілей.

```

classes = df['class']
classes.head()

>>

0    3
1    3
2    3
3    3
4    3

```

Перетворюємо номери класів до формату one hot encoding.

```

classes = pd.get_dummies(classes, prefix='Class')
classes.head()

>>

   Class_1  Class_2  Class_3  Class_4
0         0         0         1         0
1         0         0         1         0
2         0         0         1         0
3         0         0         1         0
4         0         0         1         0

```

Переводимо створені набори даних до формату масивів numpy.

```

news = news.values
classes = classes.values

```

Розділяємо отримані набори на навчальну та тестову вибірки.

```

news_train, news_test, y_train, y_test = train_test_split(news,
                                                            classes,
                                                            test_size=0.1)

```

Створюємо токенизатор Keras для подання тексту у форматі частотного кодування. Для цього використовується спеціальний клас *Tokenizer*, який дозволяє автоматично створити словник частотного подання слів. В якості параметру під час створення токенизатора вказується максимальна кількість слів *num_words* - до словника буде входити лише вказана кількість слів. Слова, які зустрічаються дуже рідко не увійдуть до словника і будуть видалені із тексту під час формування його векторного подання.

➤ [*Детальніше про токенизатор в Keras*](#)

```
tokenizer = Tokenizer(num_words=num_words)
```

Навчаємо токенизатор на текстах новин.

```
tokenizer.fit_on_texts(news_train)
```

Переглядаємо сформований словник токенизатора.

```
tokenizer.word_index  
{  
  'the': 1,  
  'a': 2,  
  'to': 3,  
  'of': 4,  
  'in': 5,  
  ...  
  'wants': 996,  
  'does': 997,  
  'average': 998,  
  'chance': 999,  
  'experts': 1000,  
  ...  
}
```

Як бачимо, найчастіше в текстах новин зустрічаються артиклі ("the", "a", "to" тощо). Цим словам буде привласнено найменші відповідні числові значення. Слова, які зустрічаються в тексті більш рідко, будуть кодуватись більшими числами. Наприклад, слову "monday" у векторі частотного подання буде відповідати код 46.

Перетворимо новини до числового подання.

```
sequences = tokenizer.texts_to_sequences(news_train)
```

Для перевірки результату можна переглянути текст деякої новини та відповідний їй вектор у частотному кодуванні.

```
# Індекс новини для перегляду
index = 3
# Текст новини із початкового набору даних
print(news_train[index])
# Текст тієї ж новини у вигляді вектору частотного подання
print(sequences[index])
```

```
>>
```

```
As the Olympic Games headed into the home straight, Argentina won th
eir first Olympic gold medal in more than half a century when their
soccer team beat Paraguay Saturday.
```

```
[13, 1, 311, 234, 1835, 68, 1, 172, 561, 2681, 224, 31, 38, 311, 351,
670, 5, 45, 56, 287, 2, 1510, 78, 31, 1086, 120, 415, 99]
```

Переглянемо код із словника токенизатора для слова "Games" із розглянутої новини.

```
tokenizer.word_index['games']
```

```
>>
```

```
234
```

Всі вхідні вектори для нейронної мережі мають бути однакової довжини. Однак, тексти кожної з новин мають різну кількість слів. Відповідно, вектори їх частотного подання також містять різну кількість елементів. Щоб виправити цю ситуацію, скористаємось функцією *pad_sequences()*. Ця функція буде доповнювати нулями вектори, які мають довжину меншу, ніж вказану у параметрі *maxLen*. А вектори, які є довшими за *maxLen*, будуть обрізані.

```
x_train = pad_sequences(sequences, maxlen=max_news_len)
```

Переглянемо перші 5 записів із навчального набору даних.

```
x_train[:5]
array([[ 698, 1055, 159, 2, 7407, 64, 27, 550, 794, 5, 1,
        2441, 149, 4, 7779, 259, 82, 16, 1, 302, 4330, 5,
         2, 212, 4, 1075, 475, 5, 1808, 3059],
       [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 39, 1, 129, 2588, 588,
         7, 99, 3, 1622, 700, 6363, 235, 2324],
       [ 69, 4, 1364, 1408, 9, 2038, 1, 55, 59, 6, 73,
        1572, 2, 1239, 4, 18, 6701, 486, 9757, 9, 229, 1112,
         12, 2, 147, 136, 204, 3138, 7, 51],
       [ 0, 0, 13, 1, 311, 234, 1835, 68, 1, 172, 561,
        2681, 224, 31, 38, 311, 351, 670, 5, 45, 56, 287,
         2, 1510, 78, 31, 1086, 120, 415, 99],
       [8043, 7639, 7640, 5420, 8, 322, 339, 18, 2, 1388, 3499,
        68, 1, 102, 300, 4, 1, 311, 1368, 939, 89, 12,
         2, 561, 2325, 142, 35, 2617, 4, 3324]])
```

Тепер дані готові до їх аналізу нейронними мережами.

Класифікація текстів мережею LSTM

Прошарки мережі LSTM складаються зі специфічних складних рекурентних нейронів, які називаються комірками. Кожен прошарок LSTM містить декілька таких нейронів, їх кількість є зовнішнім параметром та задається аналітиком. На вхід прошарку LSTM обов'язково має бути подана послідовність. Зі скалярними величинами LSTM не працює. У зв'язку з цим, якщо мережа LSTM планується глибинною, то усі LSTM-прошарки, окрім останнього, повинні мати параметр *return_sequences=True*. Також можна задати різні активаційні функції, активувати дропаут для регуляризації, визначити метод ініціалізації ваг тощо.

- З усіма доступними параметрами прошарків LSTM в Keras можна ознайомитись в [документації](#).

Створимо мережу LSTM для класифікації новин за темами. Мережа буде мати наступну архітектуру:

- Прошарок *Embedding* для перетворення вхідних даних до формату щільних векторів. В параметрах прошарку вказуємо кількість слів у словнику

щільного подання, розмірність щільного вектора та довжину вхідного вектора.

- Прошарок *LSTM*, який містить 16 комірок.
- Повнозв'язний вихідний прошарок для класифікації. Кількість нейронів має відповідати кількості класів у задачі категоріальної класифікації, або дорівнювати 1 у задачі бінарної класифікації.

```
model_lstm = Sequential()

model_lstm.add(Embedding(num_words, 32, input_length=max_news_len))
model_lstm.add(LSTM(16))
model_lstm.add(Dense(4, activation='softmax'))
```

Компілюємо модель. Вказуємо оптимізатор *adam*, критерій якості - *categorical_crossentropy* (для задачі категоріальної класифікації), метрику - *accuracy*.

```
model_lstm.compile(optimizer='adam',
                    loss='categorical_crossentropy',
                    metrics=['accuracy'])
```

Переглянемо інформацію про створену модель.

```
model_lstm.summary()

>>
Model: "sequential"

```

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 30, 32)	320000
lstm (LSTM)	(None, 16)	3136
dense (Dense)	(None, 4)	68

```

Total params: 323,204
Trainable params: 323,204
Non-trainable params: 0
```

Створюємо callback у вигляді класу `ModelCheckpoint` для збереження нейронної мережі на кожній епосі, якщо якість роботи на валідаційному наборі даних покращилась. Найкраща мережа автоматично зберігається до файлу *best_model_lstm.h5*. В якості параметру для моніторингу вказуємо долю правильних відповідей на валідаційній множині *val_accuracy*. Будемо зберігати лише найкращу модель (*save_best_only=True*).

- Детальніше про роботу даної функції зворотного виклику можна прочитати в [документації](#).

```
model_lstm_save_path = 'best_model_lstm.h5'
checkpoint_callback_lstm = ModelCheckpoint(model_lstm_save_path,
                                           monitor='val_accuracy',
                                           save_best_only=True,
                                           verbose=1)
```

Навчаємо мережу. В якості функції зворотного виклику вказуємо щойно створений об'єкт *checkpoint_callback_lstm*.

```
history_lstm = model_lstm.fit(x_train,
                              y_train,
                              epochs=5,
                              batch_size=128,
                              validation_split=0.1,
                              callbacks=[checkpoint_callback_lstm])

>>
Epoch 1/5
802/808 [=====>.] - ETA: 0s - loss: 0.5397 - accuracy: 0.7979
Epoch 00001: val_accuracy improved from -inf to 0.89307,
saving model to best_model_lstm.h5
808/808 [=====] - 6s 7ms/step - loss: 0.5384
- accuracy: 0.7985 - val_loss: 0.3294 - val_accuracy: 0.8931
Epoch 2/5
799/808 [=====>.] - ETA: 0s - loss: 0.2844 - accuracy: 0.9094
Epoch 00002: val_accuracy improved from 0.89307 to 0.89507,
saving model to best_model_lstm.h5
```

```

808/808 [=====] - 5s 6ms/step - loss: 0.2843
- accuracy: 0.9094 - val_loss: 0.3132 - val_accuracy: 0.8951
Epoch 3/5
801/808 [=====>.] - ETA: 0s - loss: 0.2443 - accuracy:
0.9224
Epoch 00003: val_accuracy did not improve from 0.89507
808/808 [=====] - 5s 6ms/step - loss: 0.2441
- accuracy: 0.9224 - val_loss: 0.3144 - val_accuracy: 0.8917
Epoch 4/5
808/808 [=====] - ETA: 0s - loss: 0.2190 - accuracy:
0.9298
Epoch 00004: val_accuracy did not improve from 0.89507
808/808 [=====] - 5s 6ms/step - loss: 0.2190
- accuracy: 0.9298 - val_loss: 0.3273 - val_accuracy: 0.8892
Epoch 5/5
804/808 [=====>.] - ETA: 0s - loss: 0.1981 - accuracy:
0.9363
Epoch 00005: val_accuracy did not improve from 0.89507
808/808 [=====] - 5s 6ms/step - loss: 0.1983
- accuracy: 0.9362 - val_loss: 0.3363 - val_accuracy: 0.8886

```

Як видно з історії, найкращий результат на валідаційній множині мережа показувала на другій епосі. Саме ця модель була збережена до файлу *best_model_lstm.h5*.

Переглянемо графік навчання (рис. 8.1).

```

plt.plot(history_lstm.history['accuracy'])
plt.plot(history_lstm.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()

>>

```

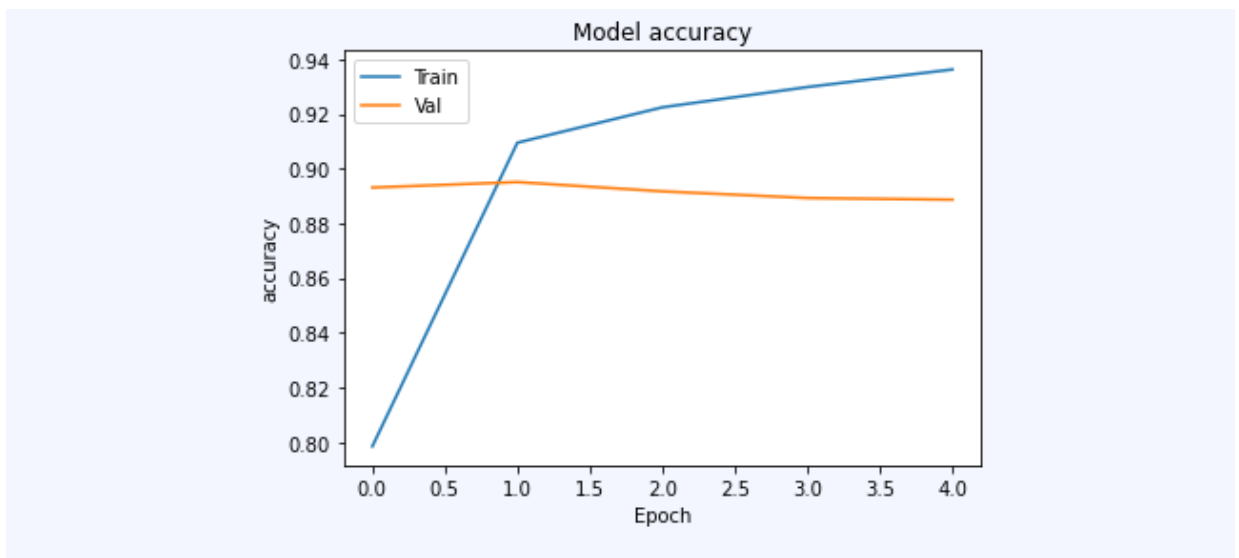



Рис. 8.1. Якість навчання для кожної епохи:

— навчальна множина; — валідаційна множина

Дійсно, з графіку видно, що після другої епохи виникає перенавчання. Callback спрацював коректно.

Класифікація текстів мережею GRU

Робота з мережами GRU є повністю аналогічною роботі з LSTM, оскільки GRU по суті є спрощеною версією LSTM. Рекурентні нейрони в прошарках GRU називаються юнітами. Практично всі параметри прошарків GRU є ідентичними до параметрів LSTM.

➤ [*Повний опис прошарків GRU в Keras*](#)

Створимо мережу GRU для класифікації новин за темами. Архітектура цієї мережі буде повністю аналогічною до попередньо розглянутої LSTM. Лише замінімо прошарок LSTM на прошарок GRU.

```
model_gru = Sequential()

model_gru.add(Embedding(num_words, 32, input_length=max_news_len))
model_gru.add(GRU(16))
model_gru.add(Dense(4, activation='softmax'))
```

Компілюємо модель.

```
model_gru.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

```
model_gru.summary()
```

```
>>
```

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
=====		
embedding_1 (Embedding)	(None, 30, 32)	320000
=====		
gru (GRU)	(None, 16)	2400
=====		
dense_1 (Dense)	(None, 4)	68
=====		
Total params: 322,468		
Trainable params: 322,468		
Non-trainable params: 0		
=====		

Створюємо callback для збереження нейронної мережі на кожній епосі, якщо якість роботи на валідаційному наборі даних покращилась. Найкраща мережа зберігається до файлу *best_model_gru.h5*.

```
model_gru_save_path = 'best_model_gru.h5'
checkpoint_callback_gru = ModelCheckpoint(model_gru_save_path,
                                          monitor='val_accuracy',
                                          save_best_only=True,
                                          verbose=1)
```

Навчаємо модель.

```
history_gru = model_gru.fit(x_train,
                             y_train,
                             epochs=5,
                             batch_size=128,
                             validation_split=0.1,
                             callbacks=[checkpoint_callback_gru])

>>
Epoch 1/5
800/808 [=====>.] - ETA: 0s - loss: 0.5557 - accuracy:
0.7994
```

```

Epoch 00001: val_accuracy improved from -inf to 0.88750,
saving model to best_model_gru.h5
808/808 [=====] - 5s 7ms/step - loss: 0.5535
- accuracy: 0.8002 - val_loss: 0.3372 - val_accuracy: 0.8875
Epoch 2/5
808/808 [=====] - ETA: 0s - loss: 0.2903 - accuracy:
0.9059
Epoch 00002: val_accuracy improved from 0.88750 to 0.89281,
saving model to best_model_gru.h5
808/808 [=====] - 5s 6ms/step - loss: 0.2903
- accuracy: 0.9059 - val_loss: 0.3204 - val_accuracy: 0.8928
Epoch 3/5
801/808 [=====>.] - ETA: 0s - loss: 0.2477 - accuracy:
0.9201
Epoch 00003: val_accuracy improved from 0.89281 to 0.89394,
saving model to best_model_gru.h5
808/808 [=====] - 5s 6ms/step - loss: 0.2473
- accuracy: 0.9201 - val_loss: 0.3163 - val_accuracy: 0.8939
Epoch 4/5
801/808 [=====>.] - ETA: 0s - loss: 0.2217 - accuracy:
0.9278 ETA: 0s - loss: 0.2200 -
Epoch 00004: val_accuracy did not improve from 0.89394
808/808 [=====] - 5s 6ms/step - loss: 0.2218
- accuracy: 0.9278 - val_loss: 0.3304 - val_accuracy: 0.8898
Epoch 5/5
806/808 [=====>.] - ETA: 0s - loss: 0.2041 - accuracy:
0.9337
Epoch 00005: val_accuracy did not improve from 0.89394
808/808 [=====] - 5s 6ms/step - loss: 0.2040
- accuracy: 0.9337 - val_loss: 0.3390 - val_accuracy: 0.8878

```

Найкраща модель була отримана на третій епосі.

Переглянемо графік навчання (8.2).

```

plt.plot(history_gru.history['accuracy'])
plt.plot(history_gru.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()

>>

```

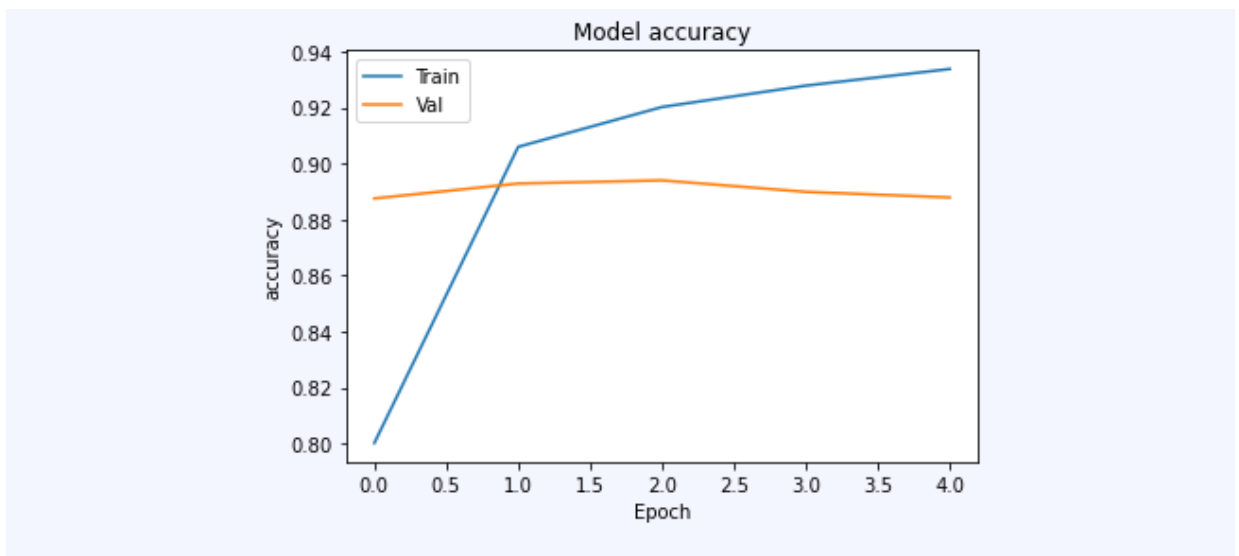


Рис. 8.2. Якість навчання для кожної епохи:

— навчальна множина; — валідаційна множина

З графіку видно, що після третьої епохи виникає перенавчання.

Оцінка результативності мереж LSTM та GRU на тестових даних

Проведемо оцінку якості роботи обох навчених мереж на тестовій множині.

Для цього спочатку перетворимо тестові дані у формат частотного подання слів.

```
test_sequences = tokenizer.texts_to_sequences(news_test)
```

Вирівнюємо довжини векторів у тестовому наборі даних.

```
x_test = pad_sequences(test_sequences, maxlen=max_news_len)
```

Переглянемо перші 5 записів із тестового набору даних.

```
x_test[:5]
array([[ 14,  913,  12,  14,   4, 3020, 3913,   5,  21,  61,   6,
        14, 5958,  15,   1,  885,   4, 1318, 1365, 512, 164, 451,
        1755,  19,   3,  30,  154,  215, 2847, 105],
       [  1, 8048,  19,  27, 178,   3,   5, 9245,   6,  90, 549,
         2, 5703, 145, 1092,  12, 2509,   4, 3799, 105, 304,  17,
         2,  12,   1, 176, 515, 517,   5,  799],
       [ 57,  10, 7398, 2339,   7,   49,   6,  389, 9851, 3924, 283,
        815, 267,  28, 129,  17,  14, 1143,  74, 3708, 811, 674,
         4,   1, 2712, 915, 149, 326,   4, 269],
       [ 13,  31,  21, 478,  58, 103,   1, 2157, 596,  17,  36,
        4538, 1437, 1327,  11, 602, 4691,  20,  70, 2572, 478,  12,
        2961, 2149,  36, 4538,  17,   5,   2, 1100],
```

```
[ 0, 24, 15, 157, 77, 7, 51, 17, 19, 27, 7093,
 1, 417, 7, 14, 117, 1326, 160, 18, 405, 145, 404,
146, 7159, 1, 417, 23, 438, 141, 7465]])
```

Проведемо оцінку якості роботи мережі LSTM на тестовій множині. Для цього завантажимо ваги найкращої моделі.

```
model_lstm.load_weights(model_lstm_save_path)
```

Тестуємо.

```
model_lstm.evaluate(x_test, y_test, verbose=1)
>>
399/399 [=====] - 1s 2ms/step - loss: 0.3140 -
accuracy: 0.8978
[0.3140435516834259, 0.8978056311607361]
```

Виконаємо аналогічну перевірку для мережі GRU.

```
model_gru.load_weights(model_gru_save_path)
model_gru.evaluate(x_test, y_test, verbose=1)
>>
399/399 [=====] - 1s 2ms/step - loss: 0.3135 -
accuracy: 0.8976
[0.3135139048099518, 0.8976489305496216]
```

На основі отриманих результатів робимо висновок, що якість обох моделей є практично однаковою. Обидві мережі показали долю правильних відповідей на тестовій множині на рівні близько 89,8%.

Перевірка роботи мереж на власних даних

Застосуємо навчені мережі для класифікації довільного тексту. Для прикладу візьмемо перші два речення з [новини про чіпування котів](#).

```
my_news = ["Microchipping pet cats will become compulsory under a wide-  
-ranging new animal welfare plan.\nEnvironment Secretary George Eustice told the BBC the policy would be  
monitored by vets \nand enforced in the same way as it is for dogs, which he said has led  
to over 90% compliance."]
```

```
my_news
```

```
>>
```

```
['Microchipping pet cats will become compulsory under a wide-ranging n  
ew animal welfare plan.Environment Secretary George Eustice told the B  
BC the policy would be monitored by vets and enforced in the same way  
as it is for dogs, which he said has led to over 90% compliance.']
```

Перетворимо текст новини до частотного подання. Зверніть увагу, що має використовуватись той самий токенизатор, що і для навчальних даних.

```
my_sequence = tokenizer.texts_to_sequences(my_news)
```

```
my_sequence
```

```
>>
```

```
[[9964, 7748, 26, 549, 204, 2, 1206, 3755, 21, 4699,  
8453, 292, 2511, 562, 734, 491, 1, 2649, 1, 890, 81, 30,  
18, 6, 5, 1, 600, 245, 13, 19, 16, 8, 5597, 84, 41, 17,  
20, 395, 3, 35, 1995, 6534]]
```

Вирівняємо довжину отриманого вектора.

```
x_my = pad_sequences(my_sequence, maxlen=max_news_len)
```

Запишемо назви класів.

```
class_names = ['Світ', 'Спорт', 'Бізнес', 'Технології']
```

Використаємо мережу LSTM для класифікації даного тексту.

```
prediction_lstm = model_lstm.predict(x_my)  
score_lstm = np.max(prediction_lstm)  
prediction_lstm = np.argmax(prediction_lstm)  
  
print(f'Категорія новини: {class_names[prediction_lstm]} \n  
Достовірність {score_lstm*100:.2f} %')
```

```
>>
```

```
Категорія новини: Технології  
Достовірність 77.81 %
```

Аналогічно застосуємо мережу GRU.

```
prediction_gru = model_gru.predict(x_my)  
score_gru = np.max(prediction_gru)  
prediction_gru = np.argmax(prediction_gru)  
  
print(f'Категорія новини: {class_names[prediction_gru]} \n  
      Достовірність {score_gru*100:.2f} %')  
  
>>
```

```
Категорія новини: Світ  
Достовірність 73.93 %
```

Як бачимо, на практиці мережа GRU гірше впоралась із класифікацією, адже насправді новина належить до категорії "Технології". Водночас, текст новини є мультикласовим, тобто її можна віднести до декількох категорій. І категорія "Світ" також є прийнятною для даної новини.

8.2. Завдання для самостійного виконання

Загальні завдання для всіх варіантів:

8. Завантажте навчальний набір даних. Проведіть оглядовий аналіз даних – визначте, чи є пропущені значення, скільки зразків у наборі даних, скільки класів, які імена класів, наскільки збалансовані класи.
9. Здійсніть необхідні попередні перетворення над даними – створіть набори предикторів та цілей.
10. Сформууйте навчальну і тестову множини.
11. Використовуючи токенізатор в Keras, перетворіть текст у формат частотного числового подання.
12. Зрівняйте довжини навчальних векторів для нейронної мережі.
13. Створіть та навчіть рекурентну нейронну мережу вказаного у завданні типу для класифікації текстів або визначення їх емоційного забарвлення (в залежності від варіанту). Побудуйте та проаналізуйте графік навчання.
14. Проведіть оцінку якості роботи навченої нейронної мережі на тестових даних.
15. Застосуйте навчену нейронну мережу для аналізу будь-якого власного тексту.

Варіант 1

Набір даних: твіти про коронавірус (corona.csv).

Тип нейронної мережі: LSTM.

Опис даних: В наборі даних зібрані твіти за тематикою пандемії коронавірусної хвороби COVID-19. Необхідно класифікувати їх за типом емоційного забарвлення. Тексти для аналізу містяться у стовпці «OriginalTweet», мітки класів – у стовпці «Sentiment».

Варіант 2

Набір даних: спам в смс (spam.csv).

Тип нейронної мережі: GRU.

Опис даних: В наборі даних зібрані текстові смс-повідомлення. Необхідно визначити, які повідомлення є спамом, а які – звичайними. Тексти для аналізу містяться у стовпці «Message», мітки класів – у стовпці «Category».

Варіант 3

Набір даних: твіти про фільм «Соціальна дилема» (theSocialDilemma.csv).

Тип нейронної мережі: LSTM.

Опис даних: В наборі даних зібрані твіти з відгуками про американський документальний фільм «The Social Dilemma» (2020). Необхідно класифікувати їх за типом емоційного забарвлення. Тексти для аналізу містяться у стовпці «text», мітки класів – у стовпці «Sentiment».

Варіант 4

Набір даних: відгуки на Amazon (amazon.csv).

Тип нейронної мережі: GRU.

Опис даних: В наборі даних зібрані відгуки до різних товарів на сайті Amazon. Необхідно визначити, які відгуки є негативними (__label__1), а які – позитивними (__label__2). Тексти для аналізу містяться у стовпці «text», мітки класів – у стовпці «class».

Варіант 5

Набір даних: твіти про катастрофи (disaster_tweets.csv).

Тип нейронної мережі: LSTM.

Опис даних: В наборі даних зібрані твіти зі словами, які описують катастрофи. Необхідно визначити, чи твіт дійсно відноситься до реального лиха, чи автор просто

використовує негативно забарвлені слова для опису інших подій (наприклад, «У мене катастрофічна ситуація з сесією в університеті»). Тексти для аналізу містяться у стовпці «text», мітки класів – у стовпці «target» (1 – катастрофа реальна, 0 – твіт не про реальну катастрофу).

Варіант 6

Набір даних: відгуки про товари (products.csv).

Тип нейронної мережі: GRU.

Опис даних: В наборі даних зібрані відгуки про товари із різних категорій. Необхідно класифікувати їх за типом емоційного забарвлення (0 – нейтральний відгук, 1 – негативний, 2 – позитивний, 3 – без оцінки). Тексти для аналізу містяться у стовпці «Product_Description», мітки класів – у стовпці «Sentiment».

Варіант 7

Набір даних: коментарі на Reddit (reddit.csv).

Тип нейронної мережі: LSTM.

Опис даних: В наборі даних зібрані коментарі до різних дописів на сайті Reddit. Деякі із відгуків були видалені модераторами (Removed = True). На основі цієї інформації потрібно створити нейронну мережу для розпізнавання коментарів, які потенційно порушують правила. Тексти для аналізу містяться у стовпці «body», мітки класів – у стовпці «REMOVED».

Варіант 8

Набір даних: відгуки про готелі в Ченнаї (hotels.csv).

Тип нейронної мережі: GRU.

Опис даних: В наборі даних зібрані коментарі з відгуками про більш ніж 500 готелів у індійському місті Ченнай. Необхідно класифікувати їх за типом емоційного

забарвлення (1: Negative 2:Average 3:Positive). Тексти для аналізу містяться у стовпці «Review_Text», мітки класів – у стовпці «Sentiment».

Варіант 9

Набір даних: емоційність твітів (emotions_tweets.csv).

Тип нейронної мережі: LSTM.

Опис даних: В наборі даних зібрані твіти з різним емоційним забарвленням. Необхідно класифікувати їх за типом емоцій, що відображає твіт. Тексти для аналізу містяться у стовпці «content», мітки класів – у стовпці «sentiment».

Варіант 10

Набір даних: відгуки у Steam (steam.csv).

Тип нейронної мережі: GRU.

Опис даних: В наборі даних зібрані рецензії на різні ігри на платформі Steam. Необхідно визначити, які рецензії є позитивними (1), а які – негативними (0). Тексти для аналізу містяться у стовпці «user_review», мітки класів – у стовпці «user_suggestion».

Варіант 11

Набір даних: категорії сайтів (websites.csv).

Тип нейронної мережі: LSTM.

Опис даних: В наборі даних містяться тексти, отримані з сайтів різних категорій. Необхідно за характером тексту визначити, до якої саме категорії належить цей сайт. Тексти для аналізу містяться у стовпці «cleaned_website_text», мітки класів – у стовпці «Category».

Варіант 12

Набір даних: настрої Ілона Маска (elonMusk.csv).

Тип нейронної мережі: GRU.

Опис даних: В наборі даних зібрані твіти з акаунту Ілона Маска. Необхідно визначити, які твіти Ілон писав з позитивним настроєм, а які – з негативним. Тексти для аналізу містяться у стовпці «tweets», мітки класів – у стовпці «sentiment».

Список рекомендованой литературы

1. Хайкин С. Нейронные сети. Полный курс (2-е издание) / С. Хайкин.– М:Диалектика, 2020.– 1104 с.
2. Шолле Ф. Глубокое обучение на Python / Ф. Шолле. – СПб: Питер Пресс, 2018. – 400 с.
3. Гудфеллоу Я. Глубокое обучение / Я. Гудфеллоу, И. Бенджио, А. Курвилль. – М: ДМК Пресс, 2017. – 652 с.
4. Орельен Ж. Прикладное машинное обучение с помощью Scikit-Learn и TensorFlow: концепции, инструменты и техники для создания интеллектуальных систем / Ж. Орельен., 2018. – 688 с.
5. Рашка С. Python и машинное обучение. Машинное и глубокое обучение с использованием Python, scikit-learn и TensorFlow 2. 3-е издание / С. Рашка, В. Мирджалили., 2020. – 841 с..
6. Rosebrock A. Deep Learning for Computer Vision with Python. Practitioner Bundle / A. Rosebrock., 2017. – 210 p.
7. Саттон Р. С. Обучение с подкреплением / Р. С. Саттон, Э. Д. Барто. – М: ДМК Пресс, 2020.– 552 с.
8. Trask A. Grokking Deep Learning / A. Trask..– Manning Publications, 2019. – 335 p.
9. Burkov A. The Hundred-Page Machine Learning Book / A. Burkov., 2019. – 160 p.
- 10.Лапань М. Глубокое обучение с подкреплением / М. Лапань. – СПб: ИД "Питер", 2020. – 496 с.
- 11.Николенко С. Глубокое обучение. Погружение в мир нейронных сетей / С. Николенко, Е. Архангельская, А. Кадури. – СПб: ИД "Питер", 2018. – 480 с.