ПРАКТИКУМ 3. НЕЙРОННІ МЕРЕЖІ В ЗАДАЧАХ ПРОГНОЗУВАННЯ (РЕГРЕСІЇ)

3.1. 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 подібрати 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 навчимось 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 заголовок 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()

>>
```

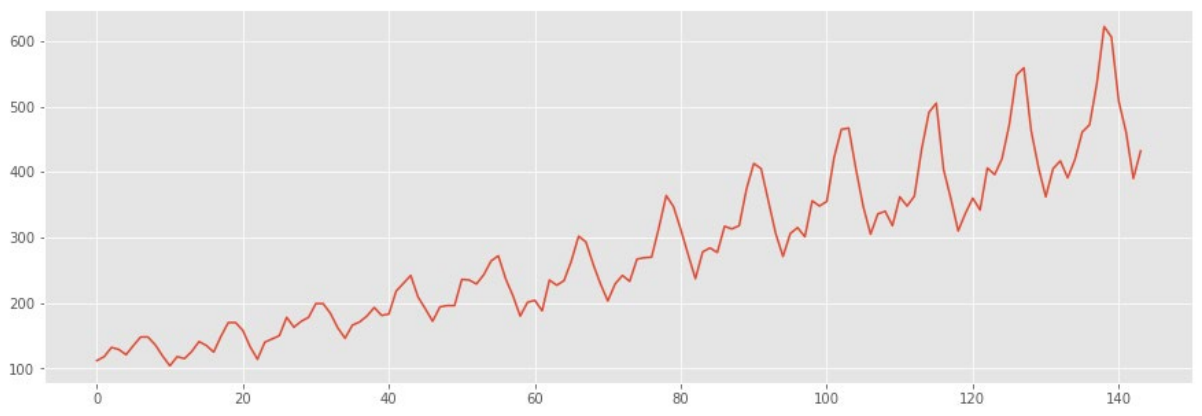


Рис. 3.1. Обсяг пасажирських перевезень у США за 1949 - 1960 роки (за 144 місяці)

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. 3.2). Таким чином, модель буде вчитись робити прогноз для логарифму ряду, а не самого ряду.

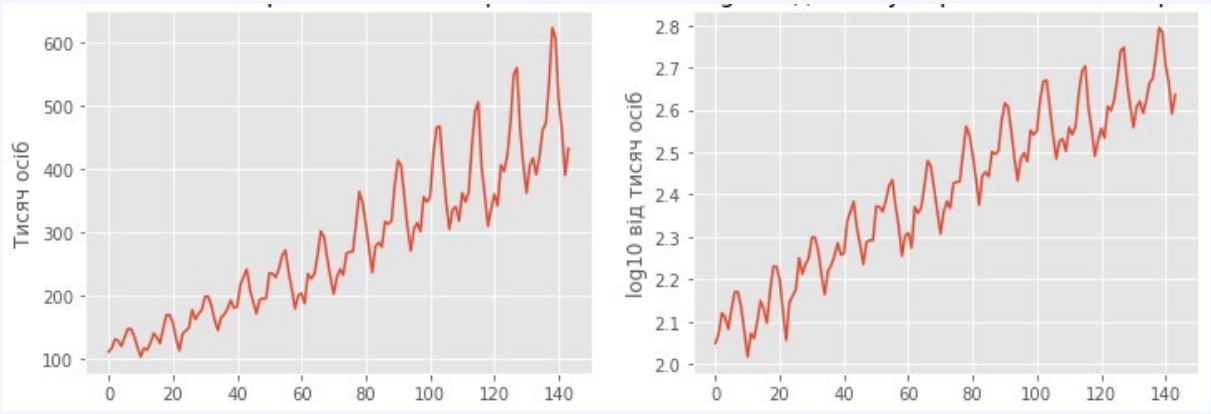
```
#  Потрібно прогнозувати логарифм
ser_g['log_y'] = numpy.log10(ser_g['series_g'])
```

```
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')

>>
```



а б

Рис. 3.2. Обсяг пасажирських перевезень у США за 144 місяці:
а – обсяг перевезень пасажирів; б – log10 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 тесових 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_tes
t, 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')


>>
```

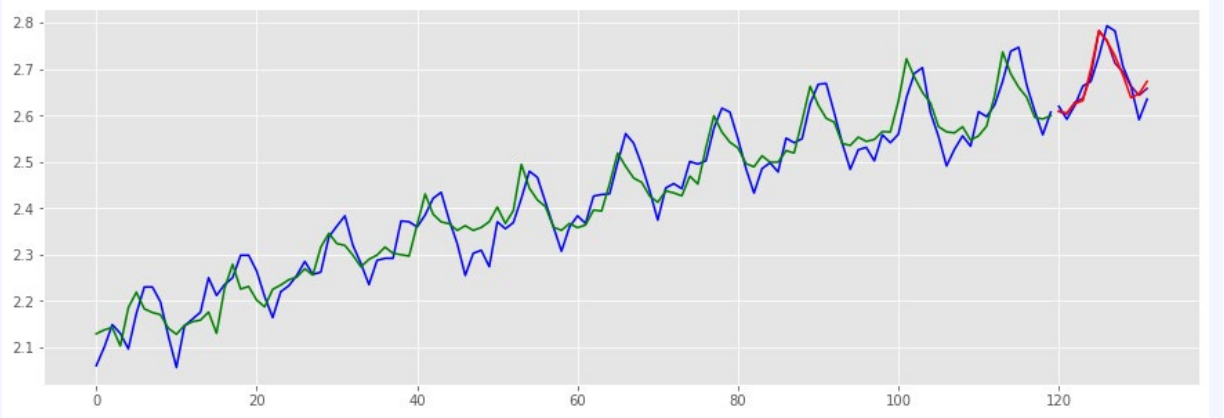


Рис. 3.3. Обсяг пасажирських перевезень у США за 144 місяці:
blue – початкові дані; *green* – апроксимація моделлю; *purple* – грубий прогноз; *red* – правильний прогноз

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. 3.4), 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')

>>
```

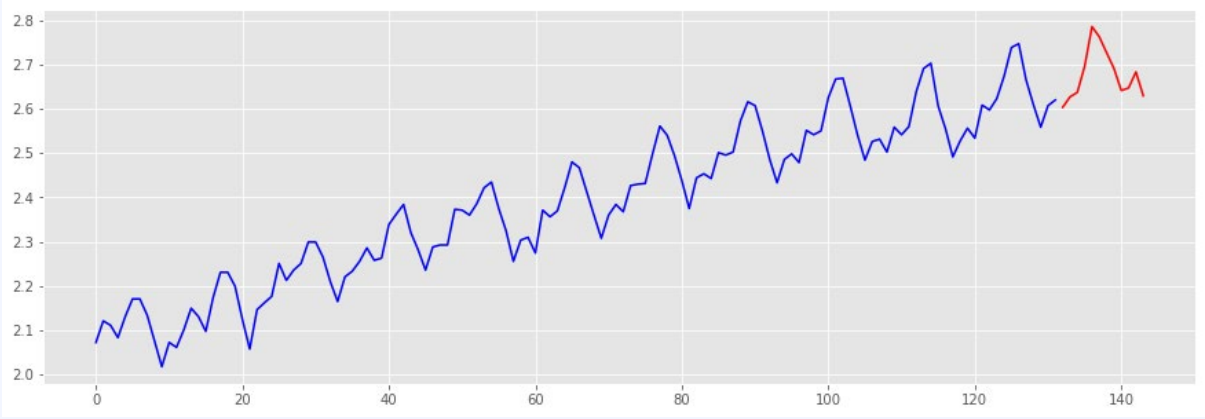


Рис. 3.4. Обсяг пасажирських перевезень у США з прогнозом на наступні 12 місяців: *blue* – початкові дані; *red* – прогноз

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. 3.5).

```
# 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')

>>
```

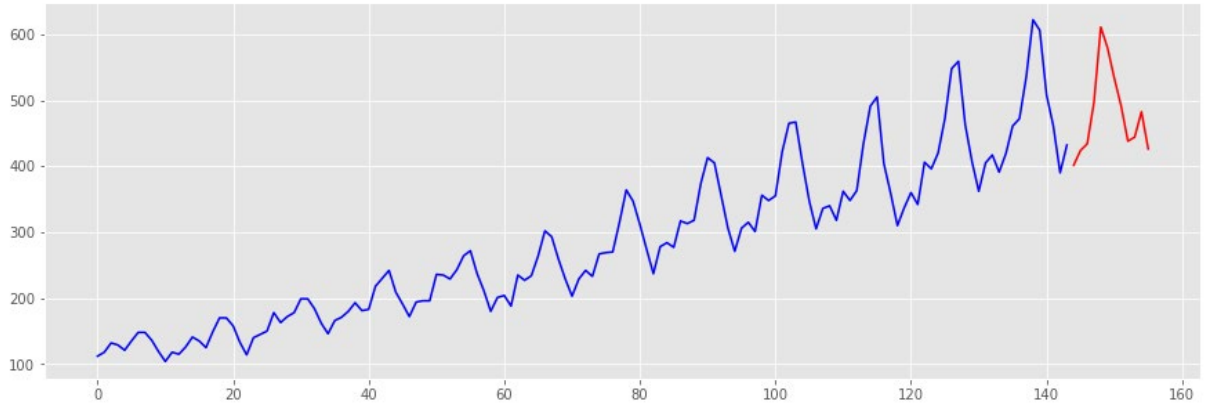


Рис. 3.5. Обсяг пасажирських перевезень у США з прогнозом на наступні 12 місяців: *blue* – початкові дані; *red* – прогноз (без логарифмування)

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. Сформуйте 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 Великобрит анії (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: 5 місяців.

Варіант 2

Набір даних: т рафік на дорогах міст а (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 тижні.

Термін прогнозу: 3 тижні.

Варіант 3

Набір даних: кількіст ь продаж ів продукт у (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 (DailyDelhiClimate.csv).

Опис даних: В наборі містяться дані про середню температуру в Делі за кожен день 2013-2017 років. Необхідно зробити прогноз температури на наступні тижні.

Термін прогнозу: 3 тижні.

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: варт іст ь золот а (gold_price_data.csv).

Опис даних: В наборі містяться дані про вартість золота за 1970-2020 роки. Необхідно зробити прогноз вартості золота на наступні тижні. Для покращення якості прогнозу рекомендується взяти дані лише за останній рік. Врахувати, що біржа працює лише по робочих днях.

Термін прогнозу: 2 тижні.

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 наступний місяць.

Термін прогнозу: 1 місяць.

Варіант 8

Набір даних: продаж і супермаркет у 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: варт іст ь 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: між народні авіаперевезення 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: продаж і 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: продаж і 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 4. ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ

4.1. 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 зорбраженні - кота 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 зображеннь, 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 необхадно вказати параметр *validation_split* - доля зображень, які будуть віднесені до валідаційної підмножини.

Далі задається розмур 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. 4.1).

```
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. 4.1. 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. 4.2).

```
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()

>>
```

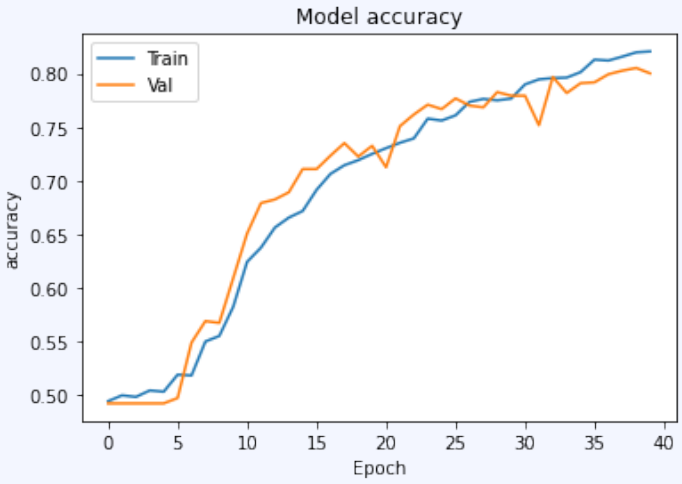


Рис. 4.2. Якість навчання для кожної епохи:

— 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. 4.3).

```
# 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)
```

Рис. 4.3. Результат роботи мережі

4.2. Завдання для самостійного виконання

Загальні завдання для всіх варіантів:

1. Завантажте навчальний набір даних. Сформуйте 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. Сформуйте 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: чуж ий/хиж ак (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: кот и 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 спорт у (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: сорт ування сміт т я (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.