**Міністерство освіти і науки України**

**Національний технічний університет України «КПІ» імені Ігоря Сікорського**

**Кафедра обчислювальної техніки ФІОТ**

**ЗВІТ**

**з лабораторної роботи №7**

**з навчальної дисципліни «Computer Vision»**

**Тема:**

**ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ ІДЕНТИФІКАЦІЇ ОБ’ЄКТІВ НА**

**ЦИФРОВИХ ЗОБРАЖЕННЯХ ДЛЯ ЗАДАЧ COMPUTER VISION**

**Виконав:**

Студент 3 курсу кафедри ІПІ ФІОТ,

Навчальної групи ІП-11

Головня О.Р.

**Перевірив:**

Професор кафедри ОТ ФІОТ

Писарчук О.О.

**Київ 2024**

**І. Мета роботи:**

Дослідити принципи та особливості підготовки даних, синтезу, навчання та

застосування штучних нейронних мереж (Artificial Neural Networks) для практичних задач ідентифікації в технологіях Computer Vision

**ІІ. Завдання**

Розробити програмний скрипт мовою Python що реалізує обчислювальний алгоритм

ідентифікації об’єктів на цифрових зображеннях за технологіями штучних нейронних

мереж (Artificial Neural Networks): підготовка даних; конструювання нейромережі;

навчання штучної нейронної мережі; застосування нейромережі:

|  |  |
| --- | --- |
| Варіант (місяць народження) | Технічні умови завдання |
| 12 | Розробити програмний скрипт, що забезпечує ідентифікацію бінарних зображень 7 спеціальних знаків, заданих матрицею растра. Для ідентифікації синтезувати, навчити та застосувати штучну нейронну мережу в «сирому» вигляді реалізації матричних операцій.  Обґрунтувати вибір архітектури та алгоритму навчання нейромережі. Довести працездатність та ефективність синтезованої нейронної мережі. |
| 12 | Із впровадженням технологій штучних нейронних мереж розробити програмний скрипт що забезпечує ідентифікацію користувача цифрового пристрою за відбитком пальця. |

**ІІІ. Результати виконання лабораторної роботи.**

**3.1. Синтезована математична модель відповідно до індивідуального завдання.**

Відповідно до завдання, синтезовано математичну модель, яка описується наступним чином:

**Вхідні дані:** Нехай *x* - вхідний вектор розміром 48, який представляє собою зображення символу 8x6 пікселів.

**Прихований шар:** Нейронна мережа має один прихований шар з 16 нейронами. Для кожного нейрона в прихованому шарі виконується наступні операції:

hidden\_inputs=*x*⋅weights1

hidden\_outputs=*σ*(hidden\_inputs), де *σ* - сигмоїдальна активаційна функція.

**Вихідний шар:** Вихідний шар має 7 нейронів, кожен з яких представляє один із символів (символи "+", "#", "&", "0", "1", "C", "3"). Для кожного нейрона в вихідному шарі виконується наступна операція:

final\_inputs=hidden\_outputs⋅weights2

final\_outputs=*σ*(final\_inputs)

**Функція втрат:** Для навчання мережі використовується квадратична функція втрат, яка вимірює різницю між прогнозованими вихідними значеннями та фактичними значеннями.

**Навчання:** Навчання мережі виконується за допомогою алгоритму зворотнього поширення помилки (backpropagation) з використанням стохастичного градієнтного спуску.

Ця модель може бути застосована для ідентифікації символів на цифрових зображеннях та може бути розширена для роботи з більшими наборами даних та складнішими мережами.

**3.2. Блок схема алгоритму та її опис.**

Блок-схема розв’язку матиме вигляд на рис. 1.

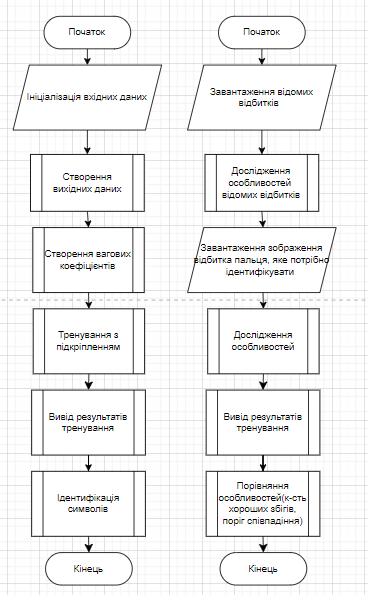


Рис. 1. Блок-схеми логіки програм для завдань 1 та 2 відповідно

**3.3. Опис структури проекту програми**

|  |
| --- |
|  |
| Рис.2. Структура проекту. |

Fingerpint \* .png – зображення відбитків для другого завдання

Holovnia\_lr\_7.docx – звіт

Holovnia\_lr\_7\_1- файл програми першого завдання

Holovnia\_lr\_7\_2- файл програми другого завдання

**3.4. Результати роботи програми відповідно до завдання.**

Для виконання роботи було обрано сім сиволів для ідентифікації:

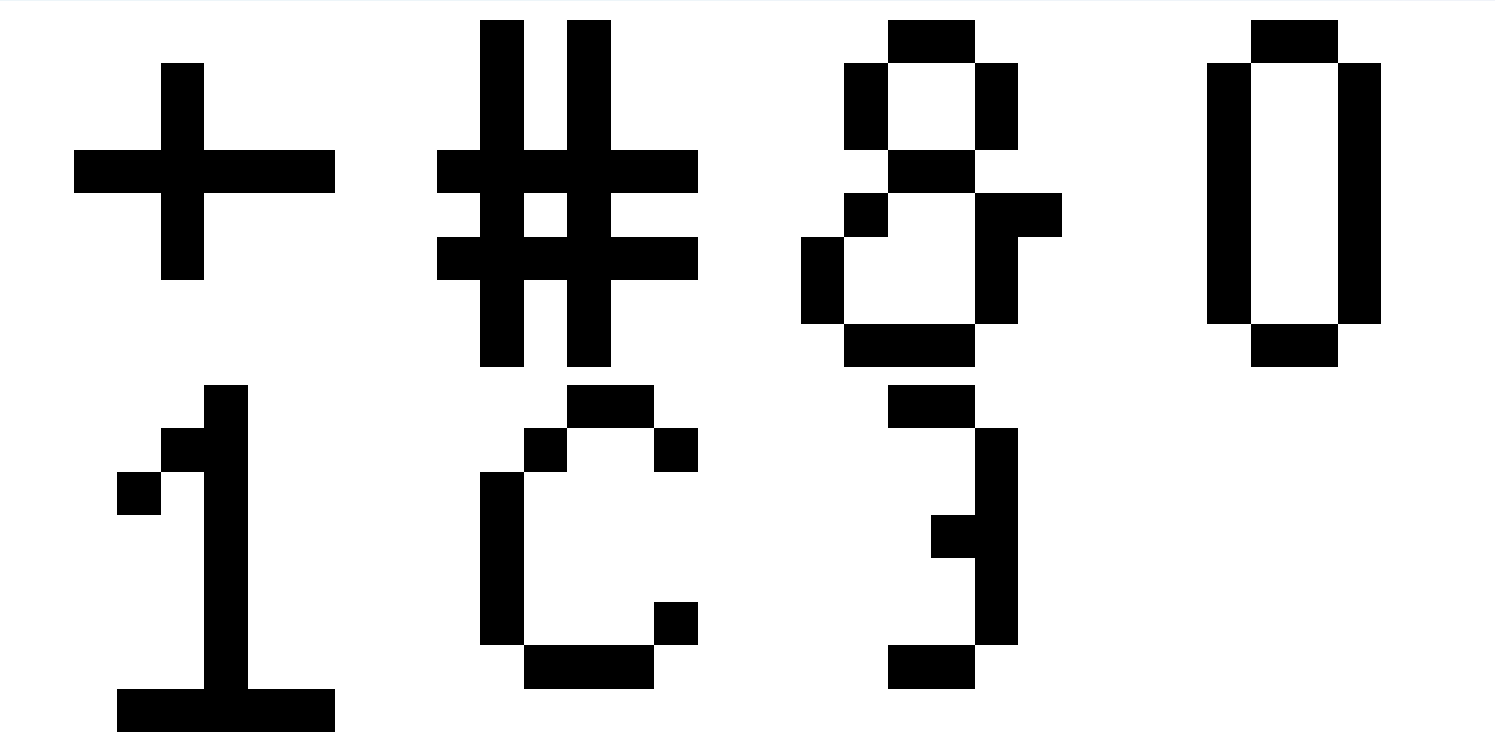


Рис. 3. Зображення обраних символів

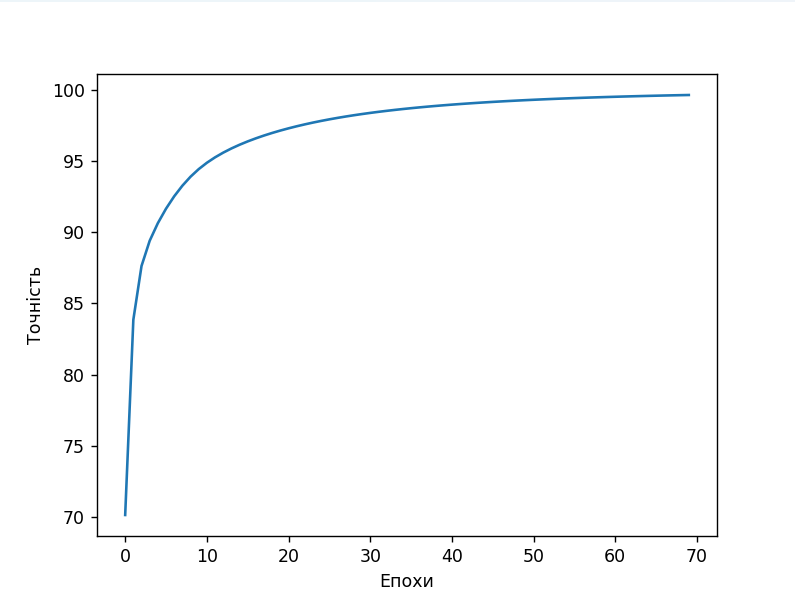


Рис. 4. Графік залежності точності від епох навчання

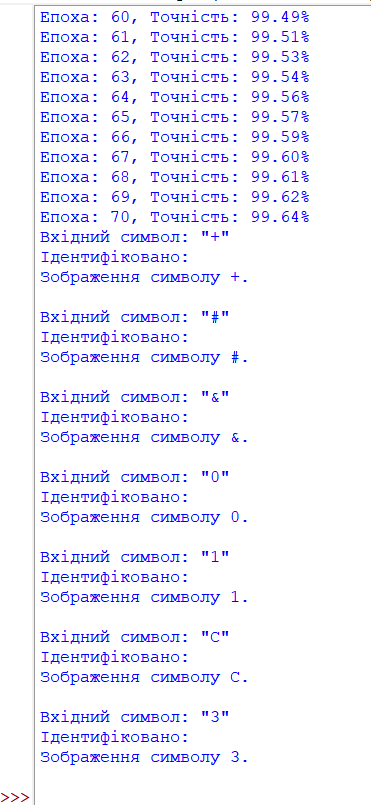


Рис. 5. Зображення результату виконання програми для завдання 1

**Завдання 7.2:**

Для виконання було обрано декілька зображень відбитків:



Рис. 6. Зображення відбитків які були занесені для ідентифікації



Рис. 7. Зображення відбитків, які ідентифікуються (Перші два – це модифікація відбитка на рис. 6)

Таким чином, змінюючи зображення(Або/чи коефіціент співпадіння) можна дійти до різного результату:

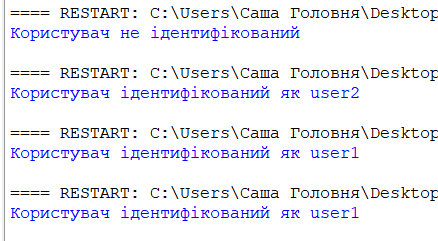


Рис. 8. Зображення результату виконання програми для завдання 2

**3.5. Програмний код.**

**Holovnia\_lr\_7\_2.**

import cv2

import numpy as np

def extract\_features(image\_path):

# Завантаження зображення

img = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

sift = cv2.SIFT\_create()

keypoints, descriptors = sift.detectAndCompute(img, None)

return descriptors

# Функція для порівняння особливостей

def compare\_features(features1, features2):

# Порівняння особливостей

bf = cv2.BFMatcher()

matches = bf.knnMatch(features1, features2, k=2)

# Визначення "хороших" збігів

good\_matches = []

for m, n in matches:

if m.distance < 0.75 \* n.distance:

good\_matches.append([m])

# Розрахунок відношення кількості "хороших" збігів до загальної кількості збігів

similarity\_ratio = len(good\_matches) / len(matches)

return similarity\_ratio

# Зразок відомих відбитків пальців (зображення та їх особливості)

known\_fingerprints = {

'user1': extract\_features('fingerprint1.png'),

'user2': extract\_features('fingerprint2.png'),

#'user22': extract\_features('fingerprint3.png'),

}

# Зображення відбитка пальця, яке потрібно ідентифікувати

unknown\_fingerprint = extract\_features('fingerprint3.png')

# Порівняння з відомими відбитками пальців

for user, features in known\_fingerprints.items():

similarity\_ratio = compare\_features(features, unknown\_fingerprint)

if similarity\_ratio > 0.1: # Задайте поріг для визначення співпадіння

print(f"Користувач ідентифікований як {user}")

break

else:

print("Користувач не ідентифікований")

**Holovnia\_lr\_7\_1.**

import numpy as np

import matplotlib.pyplot as plt

def create\_input\_data():

characters = [

np.array([

0, 0, 0, 0, 0, 0,

0, 0, 1, 0, 0, 0,

0, 0, 1, 0, 0, 0,

1, 1, 1, 1, 1, 1,

0, 0, 1, 0, 0, 0,

0, 0, 1, 0, 0, 0,

0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0,

]).reshape(8, 6),

np.array([

0, 1, 0, 1, 0, 0,

0, 1, 0, 1, 0, 0,

0, 1, 0, 1, 0, 0,

1, 1, 1, 1, 1, 1,

0, 1, 0, 1, 0, 0,

1, 1, 1, 1, 1, 1,

0, 1, 0, 1, 0, 0,

0, 1, 0, 1, 0, 0,

]).reshape(8, 6),

np.array([

0, 0, 1, 1, 0, 0,

0, 1, 0, 0, 1, 0,

0, 1, 0, 0, 1, 0,

0, 0, 1, 1, 0, 0,

0, 1, 0, 0, 1, 1,

1, 0, 0, 0, 1, 0,

1, 0, 0, 0, 1, 0,

0, 1, 1, 1, 0, 0,

]).reshape(8, 6),

np.array([

0, 0, 1, 1, 0, 0,

0, 1, 0, 0, 1, 0,

0, 1, 0, 0, 1, 0,

0, 1, 0, 0, 1, 0,

0, 1, 0, 0, 1, 0,

0, 1, 0, 0, 1, 0,

0, 1, 0, 0, 1, 0,

0, 0, 1, 1, 0, 0,

]).reshape(8, 6),

np.array([

0, 0, 0, 1, 0, 0,

0, 0, 1, 1, 0, 0,

0, 1, 0, 1, 0, 0,

0, 0, 0, 1, 0, 0,

0, 0, 0, 1, 0, 0,

0, 0, 0, 1, 0, 0,

0, 0, 0, 1, 0, 0,

0, 1, 1, 1, 1, 1,

]).reshape(8, 6),

np.array([

0, 0, 0, 1, 1, 0,

0, 0, 1, 0, 0, 1,

0, 1, 0, 0, 0, 0,

0, 1, 0, 0, 0, 0,

0, 1, 0, 0, 0, 0,

0, 1, 0, 0, 0, 1,

0, 0, 1, 1, 1, 0,

0, 0, 0, 0, 0, 0,

]).reshape(8, 6),

np.array([

0, 0, 1, 1, 0, 0,

0, 0, 0, 0, 1, 0,

0, 0, 0, 0, 1, 0,

0, 0, 0, 1, 1, 0,

0, 0, 0, 0, 1, 0,

0, 0, 0, 0, 1, 0,

0, 0, 1, 1, 0, 0,

0, 0, 0, 0, 0, 0,

]).reshape(8, 6)

]

# Visualization

fig, axs = plt.subplots(2, 4, figsize=(12, 6))

for i, ax in enumerate(axs.flatten()):

if i < len(characters): # Ensure the index is within the range of characters

ax.imshow(characters[i], cmap='binary')

ax.axis('off')

else:

ax.axis('off') # Turn off the axis for empty subplots

plt.tight\_layout()

plt.show()

# Input part of the training DataSet array

input\_data = [char.reshape(1, 48) for char in characters]

return input\_data

def create\_output\_data():

output\_data = [

[1, 0, 0, 0, 0, 0, 0],

[0, 1, 0, 0, 0, 0, 0],

[0, 0, 1, 0, 0, 0, 0],

[0, 0, 0, 1, 0, 0, 0],

[0, 0, 0, 0, 1, 0, 0],

[0, 0, 0, 0, 0, 1, 0],

[0, 0, 0, 0, 0, 0, 1]

]

return np.array(output\_data)

def sigmoid(x):

return 1 / (1 + np.exp(-x))

# Побудова нейронної мережі

def forward\_propagation(x, weights1, weights2):

# Прихований шар

hidden\_inputs = x.dot(weights1)

hidden\_outputs = sigmoid(hidden\_inputs)

# Вихідний шар

final\_inputs = hidden\_outputs.dot(weights2)

final\_outputs = sigmoid(final\_inputs)

return final\_outputs

# Ініціалізація початкових значень вагових коефіцієнтів

def initialize\_weights(rows, cols):

weights = []

for \_ in range(rows \* cols):

weights.append(np.random.randn())

return np.array(weights).reshape(rows, cols)

def calculate\_loss(output, target):

squared\_error = np.square(output - target)

loss = np.sum(squared\_error) / len(target)

return loss

def backpropagation(x, y, weights1, weights2, learning\_rate):

# Прихований шар

hidden\_inputs = x.dot(weights1)

hidden\_outputs = sigmoid(hidden\_inputs)

# Вихідний шар

final\_inputs = hidden\_outputs.dot(weights2)

final\_outputs = sigmoid(final\_inputs)

output\_errors = final\_outputs - y

hidden\_errors = np.multiply((weights2.dot((output\_errors.transpose()))).transpose(),

(np.multiply(hidden\_outputs, 1 - hidden\_outputs)))

weights1\_gradients = x.transpose().dot(hidden\_errors)

weights2\_gradients = hidden\_outputs.transpose().dot(output\_errors)

# Оновлення параметрів з контролем помилки learning\_rate

weights1 -= learning\_rate \* weights1\_gradients

weights2 -= learning\_rate \* weights2\_gradients

return weights1, weights2

def train\_network(x, y, weights1, weights2, learning\_rate=0.01, num\_epochs=10):

def update\_weights(inputs, targets, w1, w2, lr):

output = forward\_propagation(inputs, w1, w2)

loss = calculate\_loss(output, targets)

updated\_w1, updated\_w2 = backpropagation(inputs, targets, w1, w2, lr)

return loss, updated\_w1, updated\_w2

def train\_epoch(epoch, data, labels, w1, w2, lr):

epoch\_loss, updated\_w1, updated\_w2 = zip(\*[update\_weights(x, y, w1, w2, lr) for x, y in zip(data, labels)])

avg\_loss = sum(epoch\_loss) / len(data)

accuracy = (1 - avg\_loss) \* 100

print(f"Епоха: {epoch + 1}, Точність: {accuracy:.2f}%")

return accuracy, avg\_loss, updated\_w1[-1], updated\_w2[-1]

accuracies, losses, trained\_weights1, trained\_weights2 = zip(\*[train\_epoch(epoch, x, y, weights1, weights2, learning\_rate) for epoch in range(num\_epochs)])

return accuracies, losses, trained\_weights1[-1], trained\_weights2[-1]

def predict\_symbol(x, weights1, weights2):

def get\_predicted\_class(output):

return max(range(len(output[0])), key=lambda i: output[0][i])

def get\_symbol(predicted\_class):

symbol\_mapping = {

0: "+",

1: "#",

2: "&",

3: "0",

4: "1",

5: "C",

6: "3"

}

return symbol\_mapping[predicted\_class]

output = forward\_propagation(x, weights1, weights2)

predicted\_class = get\_predicted\_class(output)

symbol = get\_symbol(predicted\_class)

print(f"Зображення символу {symbol}.\n")

plt.imshow(x.reshape(8, 6), cmap='binary')

plt.show()

return

if \_\_name\_\_ == '\_\_main\_\_':

# Вхідні дані

input\_data = create\_input\_data()

output\_data = create\_output\_data()

print('Масив DataSet: навчальна пара для навчання з учителем')

print('Вхідні дані:', input\_data, '\n')

print('Вихідні дані:', output\_data, '\n')

layer\_sizes = [(48, 16), (16, 7)]

weights = [initialize\_weights(\*size) for size in layer\_sizes]

print('Навчання мережі')

accuracies, losses, \*trained\_weights = train\_network(input\_data, output\_data, \*weights, 0.1, 70)

training\_metrics = [

('Точність', accuracies),

('Втрати', losses)

]

for metric, data in training\_metrics:

plt.figure()

plt.plot(data)

plt.ylabel(metric)

plt.xlabel("Епохи")

plt.show()

symbols = ["+", "#", "&", "0", "1", "C", "3"]

for i, symbol in enumerate(symbols):

print(f'Вхідний символ: "{symbol}"')

print('Ідентифіковано:')

predict\_symbol(input\_data[i], \*trained\_weights)

**3.6. Аналіз результатів відлагодження та верифікації результатів роботи програми.**

Результати відладки та тестування підтвердили працездатність розробленого коду. Порівняння отриманих результатів з технічними умовами завдання на лабораторну роботу та верифікація функціоналу програмного коду показали, що всі вимоги були виконані у повному обсязі.

**IV. Висновки.**

Під час виконання лабораторної роботи було досліджено принципи та особливості підготовки даних, синтезу, навчання та застосування штучних нейронних мереж (Artificial Neural Networks) для практичних задач ідентифікації в технологіях Computer Vision. Було розроблено програмний скрипт, що забезпечує ідентифікацію бінарних зображень 7 спеціальних знаків, заданих матрицею растра, також програмний скрипт що забезпечує ідентифікацію користувача цифрового пристрою за відбитком пальця.