Національний технічний університет України «КПІ ім. Ігоря Сікорського»
Факультет Інформатики та Обчислювальної Техніки

Кафедра інформаційних систем та технологій

Лабораторна робота №7

з дисципліни «Технології Computer Vision»

на тему

«ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ ІДЕНТИФІКАЦІЇ ОБ'ЄКТІВ НА ЦИФРОВИХ ЗОБРАЖЕННЯХ ДЛЯ ЗАДАЧ COMPUTER VISION»

Виконала:
студентка групи ІС-12
Павлова Софія

Перевірив:
Баран Д. Р.

Київ – 2024

1. Постановка задачі

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.

Завдання І рівня:

3	Розробити програмний скрипт, що забезпечує ідентифікацію бінарних зображень 4 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.

2. Виконання

2.1. Штучна нейронна мережа

Вхідні дані

Будемо ідентифікувати 4 спеціальні знаки:

- знак «більше»: $>$;
- знак «ділити»: $\div$;
- знак «усі»: $\forall$;
- знак «належить»: $\exists$.

Подамо їх у програму у вигляді матриць 5×7:

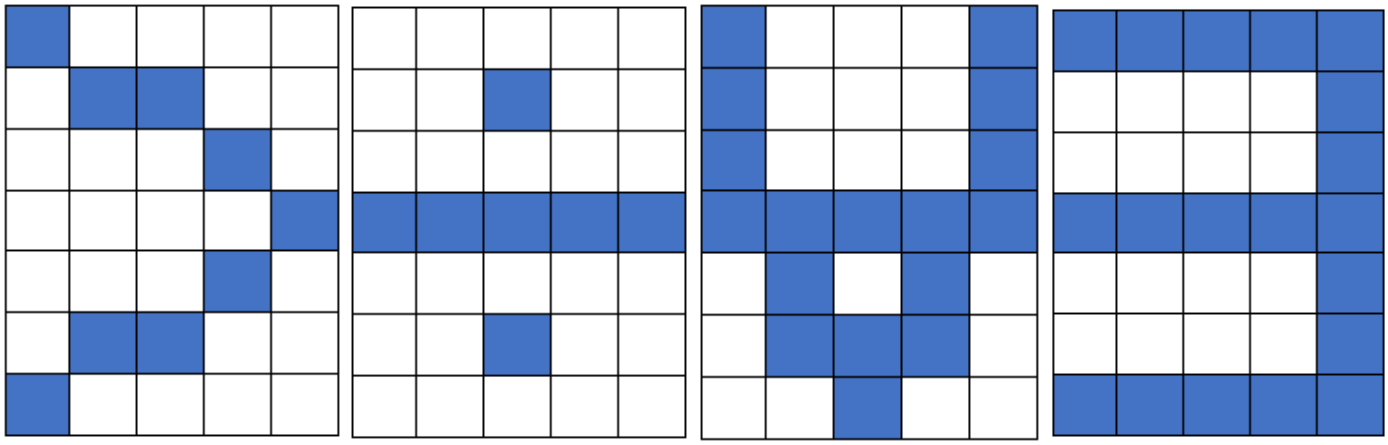


Рисунок 1 – Візуалізація класів у матрицях

Алгоритм програми

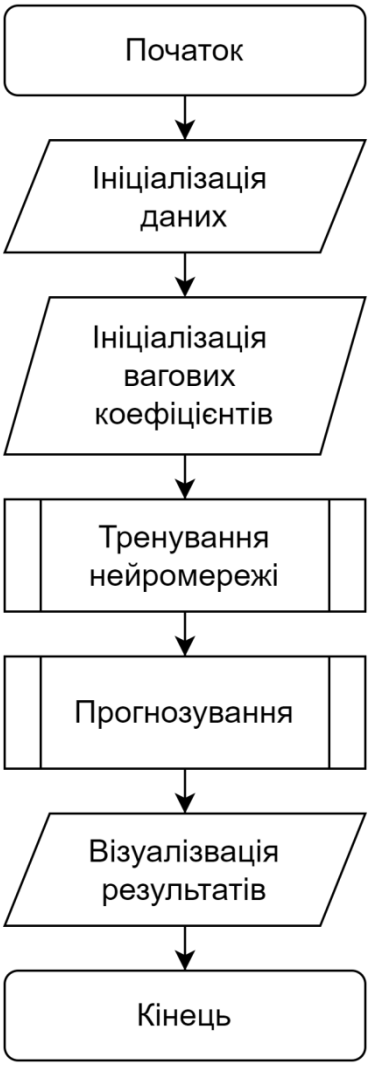


Рисунок 2 – Блок схема програми

Програмна реалізація

Вхідні дані

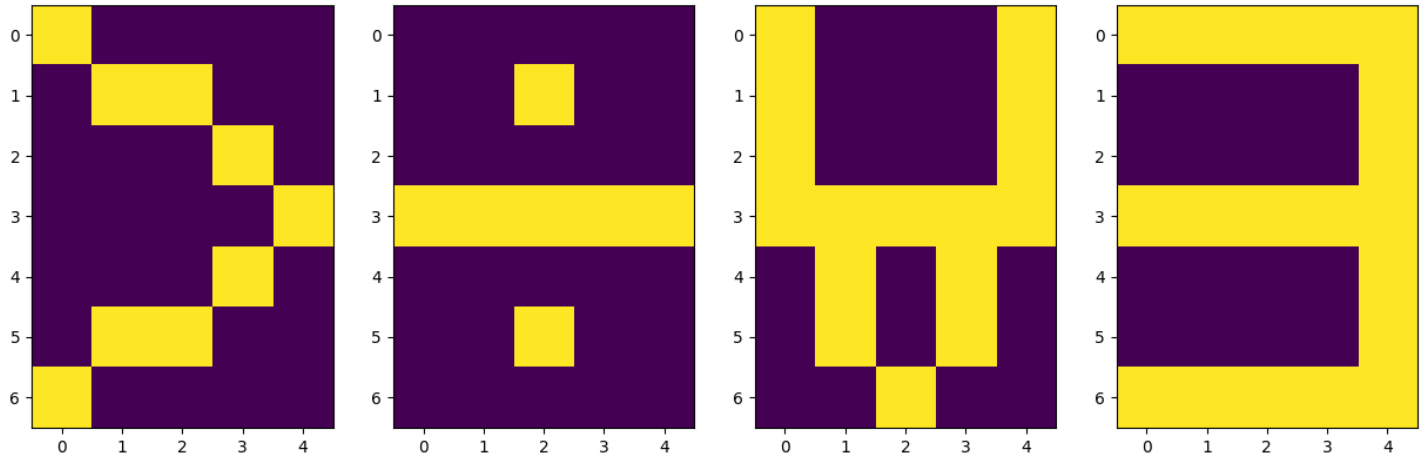
Подамо на вхід нейромережі 4 представлені у вигляді матриць символи та мітки їх класів.

Лістинг коду:

```
import numpy as np
import matplotlib.pyplot as plt

# Задання відних даних
def data_x():
    # Знак "більше"
    more = [
        1, 0, 0, 0, 0,
        0, 1, 1, 0, 0,
        0, 0, 0, 1, 0,
        0, 0, 0, 0, 1,
        0, 0, 0, 1, 0,
        0, 1, 1, 0, 0,
        1, 0, 0, 0, 0
    ]

    # Знак "ділити"
    divide = [
        0, 0, 0, 0, 0,
        0, 0, 1, 0, 0,
        0, 0, 0, 0, 0,
        1, 1, 1, 1, 1,
        0, 0, 0, 0, 0,
        0, 0, 1, 0, 0,
        0, 0, 0, 0, 0
    ]

    # Знак "усі"
    all = [
        1, 0, 0, 0, 1,
        1, 0, 0, 0, 1,
        1, 0, 0, 0, 1,
        0, 1, 1, 1, 0,
        0, 1, 0, 1, 0,
        0, 1, 0, 1, 0,
        0, 0, 1, 0, 0
    ]

    # Знак "належить"
    belong = [
        1, 1, 1, 1, 1,
        0, 0, 0, 0, 1,
        0, 0, 0, 0, 1,
        1, 1, 1, 1, 1,
        0, 0, 0, 0, 1,
        0, 0, 0, 0, 1,
        1, 1, 1, 1, 1
    ]
```

```
    x = [
        np.array(more).reshape(1, 35),
        np.array(divide).reshape(1, 35),
        np.array(all).reshape(1, 35),
        np.array(belong).reshape(1, 35),
    ]

    plt.subplot(1, 5, 1)
    plt.imshow(np.array(more).reshape(7, 5))
    plt.subplot(1, 5, 2)
    plt.imshow(np.array(divide).reshape(7, 5))
    plt.subplot(1, 5, 3)
    plt.imshow(np.array(all).reshape(7, 5))
    plt.subplot(1, 5, 4)
    plt.imshow(np.array(belong).reshape(7, 5))
    plt.show()

    return x

# Мітки
def data_y():
    out = [[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]]
    y = np.array(out)

    return y

# Головні виклики
if __name__ == '__main__':
    x = data_x()
    y = data_y()
    print('Датасет:')
    print(f'x =\n{x[0]}\n{x[1]}\n{x[2]}\n{x[3]}\n')
    print(f'y =\n{y}')
```

Результат:



```
Датасет:
x =
[[1 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0]]
[[0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]]
[[1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 0 1 1 1 0 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0]]
[[1 1 1 1 1 0 0 0 0 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0 1 0 0 0 0 1 1 1 1 1 1]]

y =
[[1 0 0 0]
 [0 1 0 0]
 [0 0 1 0]
 [0 0 0 1]]
```

Рисунок 3 – Вхідні дані нейромережі

Нейромережа

Створимо нейромережу, яка буде навчатися за допомогою алгоритму зворотного поширення помилки, де спочатку ініціалізуються рандомні вагові коефіцієнти, а з кожною наступною епохою корегуються для досягнення найкращого результату.

Архітектура нейромережі

Архітектура нейромережі складатиметься з трьох шарів: вхідного шару, прихованого шару та вихідного шару.

```
Архітектура нейромережі:
Вхідний шар (35 нейронів)
Прихований шар (4 нейрони)
Вихідний шар (4 нейрони)
```

Рисунок 4 – Архітектура нейромережі

Ініціалізація вагових коефіцієнтів

Отже для початку згенеруємо рандомні вагові коефіцієнти.

Лістинг коду:

```python
# Генерація вагових коефіцієнтів
def generate(x, y):
    l = []
    for _ in range(x * y):
        l.append(np.random.randn())
```

```
    return np.array(l).reshape(x, y)

# Головні виклики
if __name__ == '__main__':
    [...]

    w1 = generate(35, 4)
    w2 = generate(4, 4)
```

Результат:

```
Ініціалізуємо вагові коефіцієнти:
w1 =
[[ 1.93053618e-01 -2.18257456e-01  6.11554446e-01  1.14393828e+00]
 [-4.96412574e-01  4.21941087e-01 -3.22356825e-01 -1.82683717e+00]
 [-1.04391304e-01 -2.97920615e-01  1.32777316e+00 -1.34576875e+00]
 [ 1.61162701e+00  4.37525774e-02  8.31409026e-02 -1.48410705e+00]
 [ 2.34359379e-01 -2.20979280e+00 -1.53760035e+00 -6.67482393e-01]
 [-2.77943883e-01 -1.91691178e-02  1.00619109e+00  2.64111690e-01]
 [-2.09479845e+00  2.15894943e-01 -3.32915849e+00 -2.81132664e-01]
 [ 8.06200045e-01 -1.39279298e+00 -5.75369956e-01 -1.19554765e+00]
 [-5.75251201e-01 -6.86169166e-01  1.45933651e+00  8.02761545e-01]
 [-6.89086439e-01 -7.16737831e-01 -1.43329612e-01 -2.79873970e+00]
 [-1.75111801e+00 -6.98614025e-01  4.19277815e-02 -1.57236051e-01]
 [ 1.06449685e-01 -5.59545963e-01  1.52439844e+00  4.64725705e-01]
 [-1.93415893e-01  1.03272209e+00  7.71290708e-01 -8.01998295e-02]
 [-8.22397953e-01 -1.04316100e+00  7.39421747e-01  6.08646868e-02]
 [-2.02011005e-01 -9.45954629e-01  7.75053663e-01 -6.43591303e-01]
 [-1.87300898e-02 -3.04966192e-01 -7.51577964e-01 -9.06634687e-01]
 [-2.02747038e+00 -3.99449843e-01  7.63050817e-01 -9.85495522e-01]
 [-1.80225697e+00  2.55444050e-01  1.21752284e+00 -1.53041554e+00]
 [ 7.42637028e-01 -1.45191984e+00 -9.37597973e-01 -7.44725806e-02]
 [-7.56745534e-01  3.32578914e-01 -2.91807921e-01  4.57683367e-01]
 [-9.28620608e-01  1.33072042e+00  3.27884213e-01 -4.95670725e-01]
 [ 2.53369281e-01 -3.38887437e-01  5.47547077e-01 -3.16754191e-01]
 [-1.04604610e-01 -9.11929665e-01  2.89015645e-03  2.21934401e+00]
 [ 1.34781468e+00  3.68821189e-01  1.77324111e-01  1.02057545e+00]
 [ 1.65943566e+00 -2.46201823e-01 -8.45698698e-01  9.00451223e-01]
 [-1.90276619e-02 -1.67122698e+00  2.73019201e-01 -5.59424495e-01]
 [ 2.45195464e+00  6.91679427e-01  1.41234237e-01  4.70919972e-01]
 [-3.53315988e-01 -9.31598461e-01  9.99391089e-01  1.50934065e+00]
 [-2.65756078e-01  2.38769304e+00 -6.16634159e-02 -4.77224279e-01]
 [ 2.07351503e+00  9.72439833e-02  5.41392640e-01  6.74272834e-01]
 [-6.90625884e-01  7.25049042e-01 -7.48716403e-01  7.52288301e-01]
 [-7.52582022e-01 -2.90582172e-01  1.34675619e+00  2.55457708e-01]
 [ 7.85158629e-01 -3.45482889e-01  1.07956027e+00 -7.02166906e-01]
 [-1.71902052e-01  1.69134739e+00  7.96365331e-01 -2.04520774e-01]
 [ 1.32909147e+00  5.39816491e-01 -1.69655318e-01 -1.64499066e+00]]
```

```
w2 =
[[-0.38663035 -0.83958884  1.90082423  1.489897  ]
 [-0.02285489  0.07498443  0.26783986 -0.17612492]
 [-1.50674502  0.33242493  0.90292748 -0.69009532]
 [ 0.51356471  0.20563884  0.43146074 -0.06805719]]
```

Рисунок 5 – Ініціалізовані вагові коефіцієнти

Тренування нейромережі

Виконаємо тренування нейромережі на **20 епохах**.

Лістинг коду:

```python
# Прохід уперед
def forward(x, w1, w2):
    # Арихований шар
    z1 = x.dot(w1)
    a1 = sigmoid(z1)

    # Вихідний шар
    z2 = a1.dot(w2)
    a2 = sigmoid(z2)

    return a2

# Прохід назад
def back_prop(x, y, w1, w2, alpha):
    # Прихований шар
    z1 = x.dot(w1)
    a1 = sigmoid(z1)

    # Вихідний шар
    z2 = a1.dot(w2)
    a2 = sigmoid(z2)
    d2 = (a2 - y)
    d1 = np.multiply((w2.dot((d2.transpose()))).transpose(),
    np.multiply(a1, 1 - a1))
    w1_adj = x.transpose().dot(d1)
    w2_adj = a1.transpose().dot(d2)
    w1 = w1 - (alpha * w1_adj)
    w2 = w2 - (alpha * w2_adj)

    return w1, w2

# Тренування нейромережі
def train(x, Y, w1, w2, alpha, epoch):
    acc = []
    loss_val = []
    for j in range(epoch):
        l = []
        for i in range(len(x)):
            out = forward(x[i], w1, w2)
            l.append((loss(out, Y[i])))
            w1, w2 = back_prop(x[i], y[i], w1, w2, alpha)
        if (j + 1) % 10 == 0:
            print("Епоха: ", j + 1, " Точність: ", (1 - (sum(l) / len(x))) * 100)
```

```
        acc.append((1 - (sum(l) / len(x))) * 100)
        loss_val.append(sum(l) / len(x))

    return acc, loss_val, w1, w2

# Функція втрат
def loss(out, Y):
    s = (np.square(out - Y))
    s = np.sum(s) / len(y)

    return s

# Головні виклики
if __name__ == '__main__':
    [...]

    print('\nТренування нейромережі:')
    acc, loss_value, w1, w2 = train(x, y, w1, w2, 0.1, 20)
```

Результат:

```
Тренування нейромережі:
Епоха:  10  Точність:  80.86723772247227
Епоха:  20  Точність:  83.84273039978117
```

Рисунок 6 – Тренування нейромережі

Бачимо, що нейромережа показує точність **83,8%**.

Виведемо графіки точності втрат.

Лістинг коду:

```
# Головні виклики
if __name__ == '__main__':
    [...]
    plt.plot(acc)
    plt.ylabel('Точність')
    plt.xlabel('Епохи')
    plt.show()

    plt.plot(loss_value)
    plt.ylabel('Втрати')
    plt.xlabel('Епохи')
    plt.show()
```

Результат:

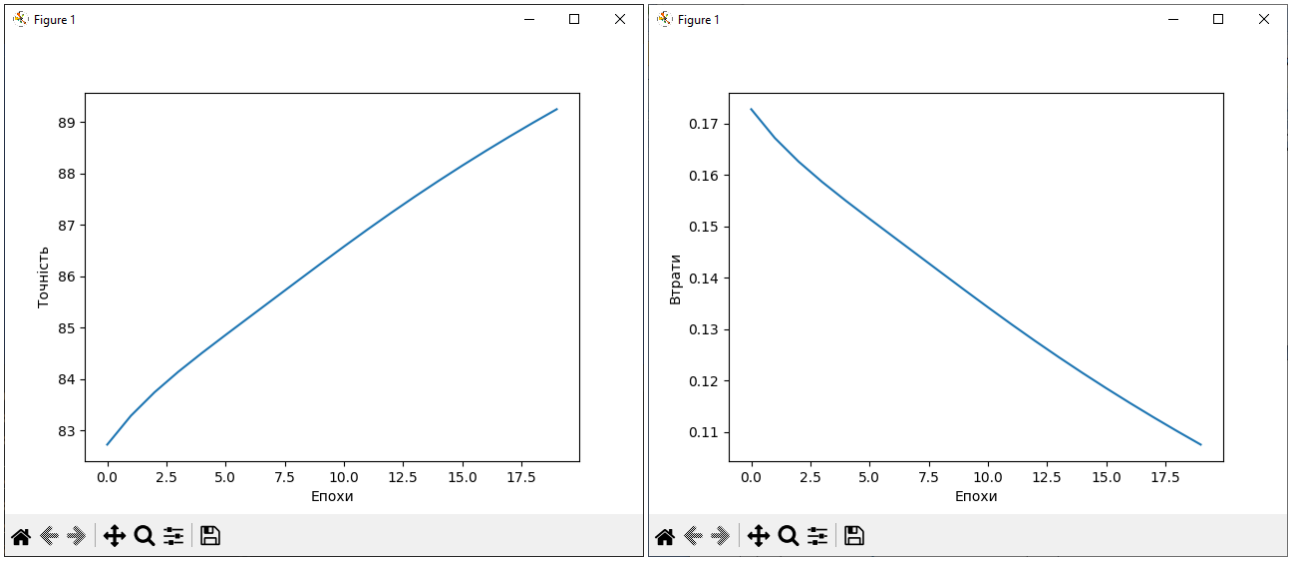


Рисунок 7 – Графіки точності та втрат 20 епох

З графіків добре видно, що нейромережа ще навчається, тому для досягнення кращого результату **збільшимо кількість епох до 100**.

Результат:

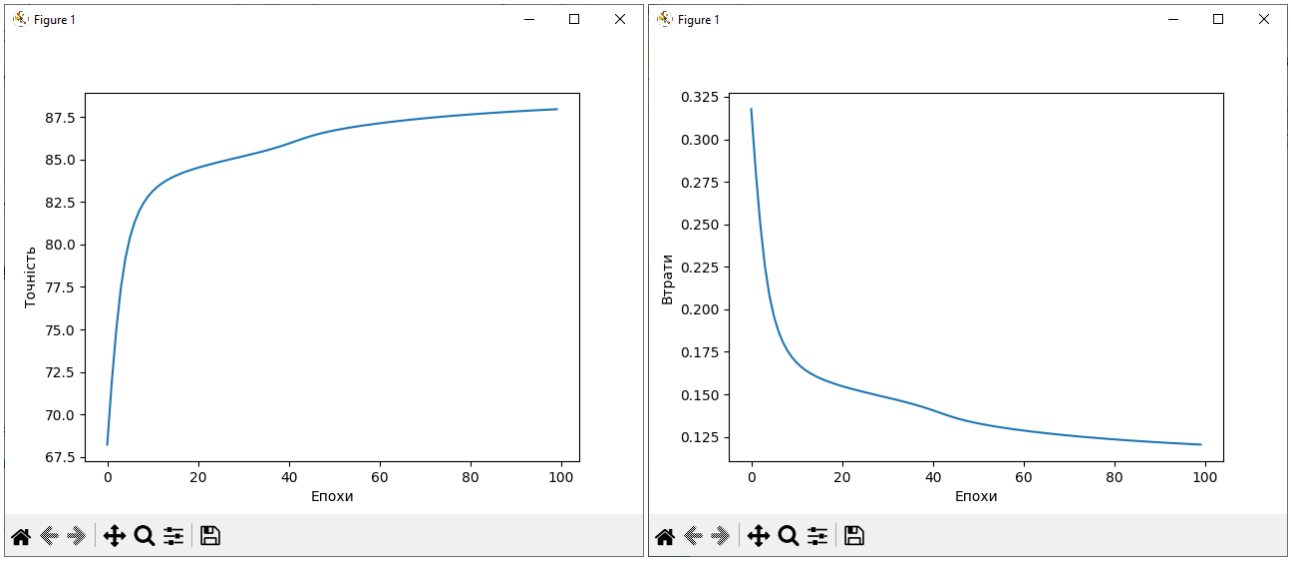


Рисунок 8 – Графіки точності та втрат 100 епох

Тепер крива показує, що **нейромережа достатньо навчилась**, але ще не почала перенавчатись. Приріст точності та зменшення втрат на останніх епохах незначне, порівняно з першими епохами.

Отже **оптимальна кількість епох** для даної задачі – **100**.

Прогнозування

Виконаємо прогнозування для наших вхідних даних.

Лістинг коду:

```
# Класифікація
def predict(x, w1, w2):
    res = forward(x, w1, w2)
    max_w = 0
    k = 0
    for i in range(len(res[0])):
        if max_w < res[0][i]:
            max_w = res[0][i]
            k = i

    return k

# Візуалізація результату
def print_results(x, w1, w2):
    symbols = ['?', '!', '↑', '↓']
    print('Очікувано\tРозпізнано\tСпівпало')
    for i in range(4):
        expected_letter = symbols[i]
        recognized_letter = symbols[predict(x[i], w1, w2)]
        print(f'{expected_letter}\t\t\t{recognized_letter}\t\t\t{True if expected_letter
== recognized_letter else False}')


# Головні виклики
if __name__ == '__main__':
    [...]

    print_results(x, w1, w2)
```

Результат:

```
Очікувано   Розпізнано  Співпало
>           >           True
÷           ÷           True
∀           ∀           True
∃           ∃           True
```

Рисунок 9 – Отримані результати

Бачимо, що **нейромережа успішно розпізнала** зашифровані у вигляді матриць символи.

2.2. Аналіз отриманих результатів

Закодовано 4 бінарних зображення спеціальних символів – підтверджено рисунком та кодом:

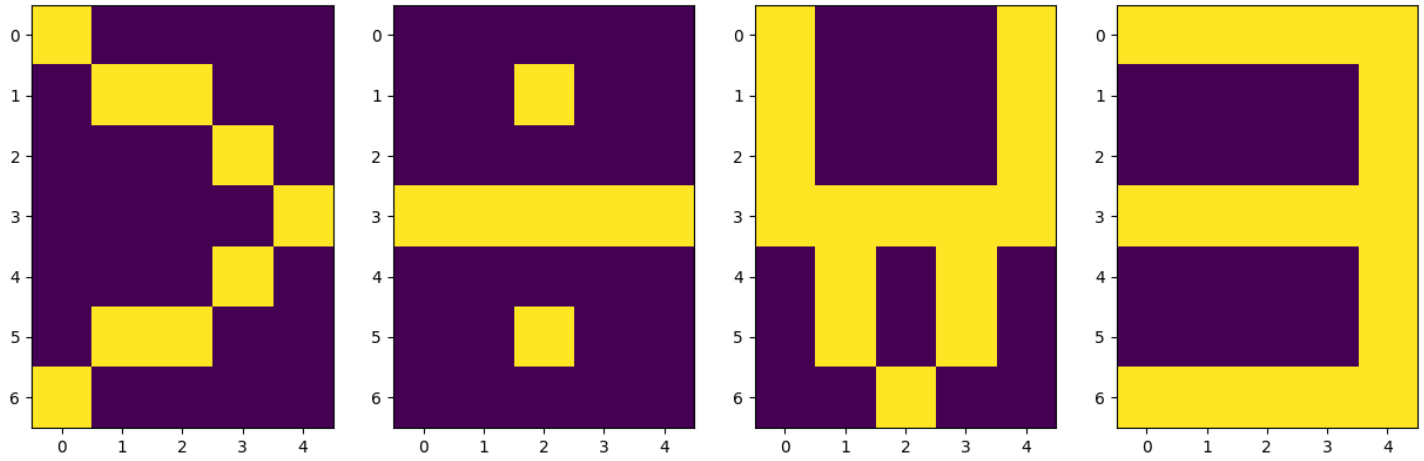


Рисунок 10 – Підтвердження виконання вимог 1

Синтезована нейромережа – підтверджено кодом.

Нейромережа успішно класифікує зображення – підтверджено рисунком.

```
Очікувано   Розпізнано  Співпало
>           >           True
÷           ÷           True
∀           ∀           True
∃           ∃           True
```

Рисунок 11 – Підтвердження виконання вимог 2

Висновок:

У результаті виконання лабораторної роботи отримано практичні та теоретичні навички підготовки датасету для задач класифікації бінарних зображень, синтезу та роботи зі штучними нейромережами зі зворотнім поширенням.

Реалізовано штучну нейромережу зворотнього поширення помилки з одним прихованим шаром. Виконано генерацію початкових коефіцієнтів та тренування нейромережі. Результати прогнозування та швидкість навчання нейромережі демонструють успішний вибір типу та архітектури нейромережі для даного завдання.