Чернівецький національний університет імені Юрія Федьковича

Факультет математики та інформатики

Кафедра математичного моделювання

# Лабораторна робота №1

# з навчальної дисципліни: “Основи штучного інтелекту”

на тему:

“Створення середовища для попередньої обробки образів та формування ознакових векторів в системах розпізнавання”

Виконала: студентка 307 групи спеціальності “Системний аналіз”

Раренко А.С.

# Чернівці – 2024

**Завдання.** У візуальному середовищі програмування розробити програму для побудови абсолютних та нормованих векторів ознак розпізнавання стандартизованих графічних зображень (цифр, букв, логотипів). Програма повинна виконувати наступні функції:

* Створити вікно, в якому має відображатися завантажене зображення, вектор ознак що відповідає зображенню, нормований вектор ознак та інші потрібні вам елементи.
* Дозволяти введення графічних зображень з графічних файлів. Розміри зображень – змінні, але образ має займати весь прямокутник зображення і не потребувати додаткової локалізації у графічному компоненті. Зображення чорно-білі, у форматі .bmp, не потребують додаткової соляризації та фільтрації.
* Будувати вектор ознак для кожного зображення. Для побудови числового вектора ознак, що відповідає зображенню, слід розділити зображення на прямокутні області (напр., сіткою 5 на 5, 4 на 5, 6 на 6, чи ін. (мал.1)). В кожній області порахувати кількість пікселів того кольору, яким ви малювали зображеня. Координати вектора ознак – це кількість цих пікселів в кожній області. Наприклад, на мал.1 вектор ознак для даного зображення цифри 2 матиме 4\*5=20 координат:

Х=(150; 400; 450; 30; 100; 15; 160; 156; 0; 80; 260; 26; 110; 245; 10; 0; 300; 320; 330; 120).

* Відображати вектор ознак у відповідному компоненті.
* Нормувати вектор ознак і зберігати його.

***Варіант сегментації:***

Задана кількість комірок прямокутної табличної схеми;

***Варіант нормування:***

За модулем (діленням всіх елементів абсолютного вектору ознак на максимальний елемент вектора).

let pixels = [];

const EMPTY = 0;

const FILLED = 1;

const BORDER = -1;

let array = [];

let gridX = 12; *// кількість стовпців у сітці*

let gridY = 12; *// кількість рядків у сітці*

let pixelSize = 50;

let squareSize = 4; *// розмір кожного квадрата для обчислення вектора ознак*

let drawing = false;

let allNormalizedVectors = [];  *// Масив для зберігання всіх нормованих векторів ознак*

function setup() {

  createCanvas(600, 600).mouseMoved(mouseMovedOnCanvas);

*// Initialize the pixel grid*

  initializePixels();

*// Button to calculate feature vector*

  let calcButton = createButton("Розрахувати вектор");

  calcButton.mouseClicked(() => {

    let absoluteVector = calculateFeatureVector();

    alert("Абсолютний вектор ознак:\n" + absoluteVector.join(", "));

  });

*// Button to clear the canvas*

  let clearButton = createButton("Очистити малюнок");

  clearButton.mouseClicked(() => {

    clearCanvas();

  });

*// Button to show all saved vectors*

  let showButton = createButton("Показати всі вектори");

  showButton.mouseClicked(() => {

    alert("Збережені нормовані вектори ознак:\n" + allNormalizedVectors.map(*v* => *v*.join(", ")).join("\n\n"));

  });

*// Button to save vectors to a file*

  let saveButton = createButton("Зберегти вектори в файл");

  saveButton.mouseClicked(() => {

    saveVectorsToFile();

  });

}

function draw() {

  background(255);

  drawGrid();

  drawOverlayGrid(); *// Додаємо нову функцію для малювання сітки розбиття*

}

function drawGrid() {

  stroke(0);

*// Draw grid for drawing*

*for* (let y = 0; y < height; y += pixelSize) line(0, y, width, y);

*for* (let x = 0; x < width; x += pixelSize) line(x, 0, x, height);

*// Draw pixels*

*for* (let x = 0; x < gridX; x++) {

*for* (let y = 0; y < gridY; y++) {

*if* (pixels[x][y] === FILLED || pixels[x][y] === BORDER) {

        fill(0);

        square(x \* pixelSize, y \* pixelSize, pixelSize);

      }

    }

  }

}

*// Нова функція для малювання сітки розбиття*

function drawOverlayGrid() {

  stroke(255, 0, 0); *// Червона сітка*

  strokeWeight(2);   *// Товщина ліній сітки*

*// Малюємо сітку розбиття*

*for* (let y = 0; y < height; y += pixelSize \* squareSize) {

    line(0, y, width, y);

  }

*for* (let x = 0; x < width; x += pixelSize \* squareSize) {

    line(x, 0, x, height);

  }

  strokeWeight(1);  *// Повертаємо товщину ліній до звичайного значення*

}

function calculateFeatureVector() {

  array = [];

*for* (let j = 0; j < gridY; j += squareSize) {

*for* (let i = 0; i < gridX; i += squareSize) {

      let count = 0;

*for* (let x = i; x < i + squareSize; x++) {

*for* (let y = j; y < j + squareSize; y++) {

*if* (x < gridX && y < gridY && pixels[x][y] === BORDER) {

            count++;

          }

        }

      }

      array.push(count);

    }

  }

  let absoluteVector = array.slice();  *// Зберігаємо абсолютний вектор до нормалізації*

  let maxCount = Math.max(...array);

*for* (let i = 0; i < array.length; i++) {

    array[i] = (array[i] / maxCount).toFixed(2);

  }

  saveFeatureVector(array);  *// Збереження нормованого вектора ознак*

*return* absoluteVector;  *// Повертаємо абсолютний вектор для відображення*

}

function saveFeatureVector(*vector*) {

  allNormalizedVectors.push(*vector*.slice());  *// Додаємо копію нормованого вектора до масиву збережених векторів*

}

function saveVectorsToFile() {

*// Перетворюємо масив векторів у текстовий формат*

  let content = allNormalizedVectors.map(*v* => *v*.join(", ")).join("\n");

*// Створюємо Blob для збереження даних*

  let blob = new Blob([content], { type: 'text/plain' });

*// Створюємо посилання для завантаження файлу*

  let a = document.createElement('a');

  a.href = URL.createObjectURL(blob);

  a.download = 'vectors.txt';

*// Додаємо посилання на сторінку і натискаємо його*

  document.body.appendChild(a);

  a.click();

*// Видаляємо посилання після завантаження*

  document.body.removeChild(a);

}

function mousePressed() {

  drawing = true;

}

function mouseReleased() {

  drawing = false;

}

function mouseMovedOnCanvas(*event*) {

*if* (drawing) {

    let x = ~~(mouseX / pixelSize);

    let y = ~~(mouseY / pixelSize);

*if* (x >= 0 && x < gridX && y >= 0 && y < gridY) {

      pixels[x][y] = BORDER;

    }

  }

}

*// Initialize the pixel grid*

function initializePixels() {

  pixels = [];

*for* (let i = 0; i < gridX; i++) {

    pixels.push(new Array(gridY).fill(EMPTY));

  }

}

*// Clear the canvas and reset the grid*

function clearCanvas() {

  initializePixels();

  redraw();

}





