HTML file

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <link rel="stylesheet" href="main.css">

    <title>AI</title>

</head>

<body>

    <canvas id = "canvas"></canvas>

    <input type = "file" id = "fileInput" accept = "image/bmp" multiple >

    </input>

    <input type="file" name="photo" id="addEtalonBtn" accept = "image/bmp" multiple>

    <label for="addEtalonBtn" class="btn">Додайте еталон</label>

    </input>

    <button class="recognizeBtn btn">Розпізнати</button>

    <p class="output"></p>

</body>

<script type = "module" src = "main.js"></script>

</html>

Js file

const fileInput = document.querySelector("#fileInput");

const addEtalonBtn = document.querySelector("#addEtalonBtn");

const recognizeBtn = document.querySelector(".recognizeBtn");

const recognizeOut = document.querySelector(".output");

let pixelColors;

let n;

const meshSize = 4;

let fileName = "";

let clasters = {};

let vector;

const size=36;

class HopfieldNetwork {

  constructor() {

      this.etalons = [];

      this.numFeatures = 0;

      this.hopfieldMatrix = [];

      this.classNames = [];

  }

  addEtalon(etalon, className) {

      if (this.numFeatures === 0) {

          this.numFeatures = etalon.length;

      } else if (etalon.length !== this.numFeatures) {

          console.error("Incorrect etalon length.");

          return;

      }

      console.log(etalon)

      this.etalons.push(this.binarizeFeatures(etalon));

      this.hopfieldMatrix = this.calculateHopfieldMatrix();

      this.classNames.push(className);

  }

  binarizeFeatures(features) {

      const threshold = 1; // Threshold for binarization

      return features.map(value => (value > 0 ) ? 1 : -1);

  }

  calculateHopfieldMatrix() {

      const hopfieldMatrix = Array.from({ length: this.numFeatures }, () => Array(this.numFeatures).fill(0));

      for (let i = 0; i < this.numFeatures; i++) {

          for (let j = 0; j < this.numFeatures; j++) {

              if (i !== j) {

                  for (let k = 0; k < this.etalons.length; k++) {

                      hopfieldMatrix[i][j] += this.etalons[k][i] \* this.etalons[k][j];

                  }

                  hopfieldMatrix[i][j] /= this.numFeatures;

              }

          }

      }

      return hopfieldMatrix;

  }

  classify(inputVector) {

      const classifiedVector = [...inputVector];

      for (let i = 0; i < this.numFeatures; i++) {

          for (let j = 0; j < this.numFeatures; j++) {

              classifiedVector[i] += this.hopfieldMatrix[i][j] \* inputVector[j];

          }

          classifiedVector[i] = (classifiedVector[i] > 0) ? 1 : -1;

      }

      let maxSimilarity = -Infinity;

      let classifiedClassName = null;

      for (let i = 0; i < this.etalons.length; i++) {

          const similarity = this.calculateVectorSimilarity(this.etalons[i], classifiedVector);

          if (similarity > maxSimilarity) {

              maxSimilarity = similarity;

              classifiedClassName = this.classNames[i];

          }

      }

      return classifiedClassName;

  }

  calculateVectorSimilarity(vector1, vector2) {

      let similarity = 0;

      for (let i = 0; i < vector1.length; i++) {

          similarity += vector1[i] \* vector2[i];

      }

      return similarity;

  }

}

// Приклад використання:

const hopfieldNet = new HopfieldNetwork();

console.log(hopfieldNet)

addEtalonBtn.addEventListener("change", async () => {

  console.log(addEtalonBtn.files)

  for (let i = 0; i < addEtalonBtn.files.length; i++) {

    let imageFile = addEtalonBtn.files[i];

    const img = document.createElement('img');

    img.src = URL.createObjectURL(imageFile);

    const canvas = document.createElement('canvas');

    img.onload = async () => {

      const ctx = canvas.getContext('2d');

      canvas.width = img.width;

      canvas.height = img.height;

      n = img.width;

      ctx.drawImage(img, 0, 0, img.width, img.height);

      const imageData = ctx.getImageData(0, 0, img.width, img.height);

      pixelColors = imageData.data;

      vector = generateVector(pixelColors);

      fileName = imageFile.name.split("\_")[0];

      hopfieldNet.addEtalon(vector, fileName);

      console.log(hopfieldNet)

    };

  }

})

fileInput.addEventListener("change", async () => {

    let imageFile = fileInput.files[0];

    fileName = imageFile.name.split("\_")[0];

    const reader = new FileReader();

    reader.onload = async () => {

      const imageBuffer = reader.result;

      const img = document.createElement('img');

      img.src = URL.createObjectURL(imageFile);

      img.onload = async () => {

        const canvas = document.querySelector('#canvas');

        const ctx = canvas.getContext('2d');

        canvas.width = img.width;

        canvas.height = img.height;

        n = img.width;

        ctx.drawImage(img, 0, 0, img.width, img.height);

        const imageData = ctx.getImageData(0, 0, img.width, img.height);

        pixelColors = imageData.data;

        console.log(pixelColors)

        vector = generateVector(pixelColors);

      };

      // Завантажте зображення

      img.src = URL.createObjectURL(imageFile);

    };

    reader.readAsArrayBuffer(imageFile);;

});

recognizeBtn.addEventListener("click", () => {

  if (!fileInput.files[0]) {

    recognizeOut.textContent = "Виберіть зображення";

    return;

  }

  let prediction = hopfieldNet.classify(vector);;

  recognizeOut.textContent = "Відповідь: " + prediction;

})

function normVector(vector){

  return vector.map((num) => num / Math.max(...vector));

}

function getAvgVector(x1,x2){

  return x1.map((obj, index)=> (obj + x2[index])/2);

}

function generateVector(pixels) {

  let colorBits = [];

  let vector = [];

  for (let i = 0; i < pixels.length; i += meshSize) {

    colorBits.push(pixels[i] === 0 ? -1 : 1);

  }

  console.log("colorBits", colorBits, "Довжина", colorBits.length);

  for (let i = 0; i < n \* n - n; i += n \* 2) {

    for (let j = 0; j < n; j += Math.sqrt(meshSize)) {

      let count = 0;

      count += colorBits[i + j];

      count += colorBits[i + j + 1];

      count += colorBits[i + j + 12];

      count += colorBits[i + j + 13];

      vector.push(count);

      count = 0;

    }

  }

  vector = normVector(vector);

  console.log(vector)

  return vector;

}

