Colegiul Național “Silvania”

Zalău

Recunoasterea tiparelor cu ajutorul Retelelor Neuronale Convolutionale

LUCRARE DE SPECIALITATE

Filiera: Teoretică

Profil: Real

Specializarea: Matematică – Informatică, Intensiv Informatică

Coordonator:

Prof. Deac Paula – Cristina

Absolvent:

Gurău Bogdan-Vlad

Clasa: **a XII - a D**

Zalău

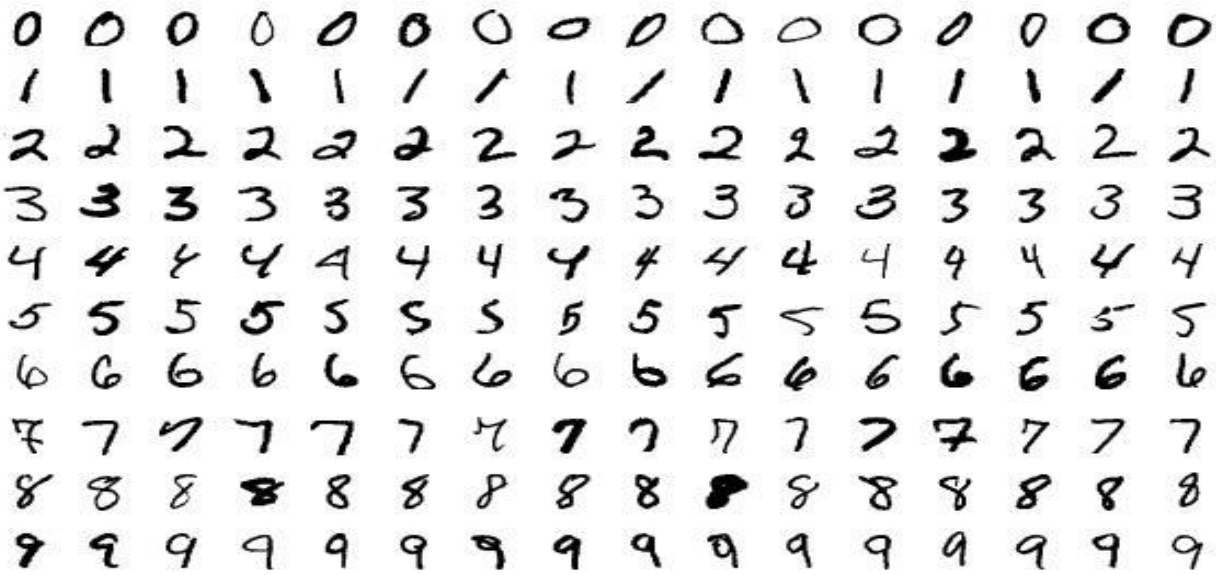
- mai 2025 -

Cuprins

Introducere

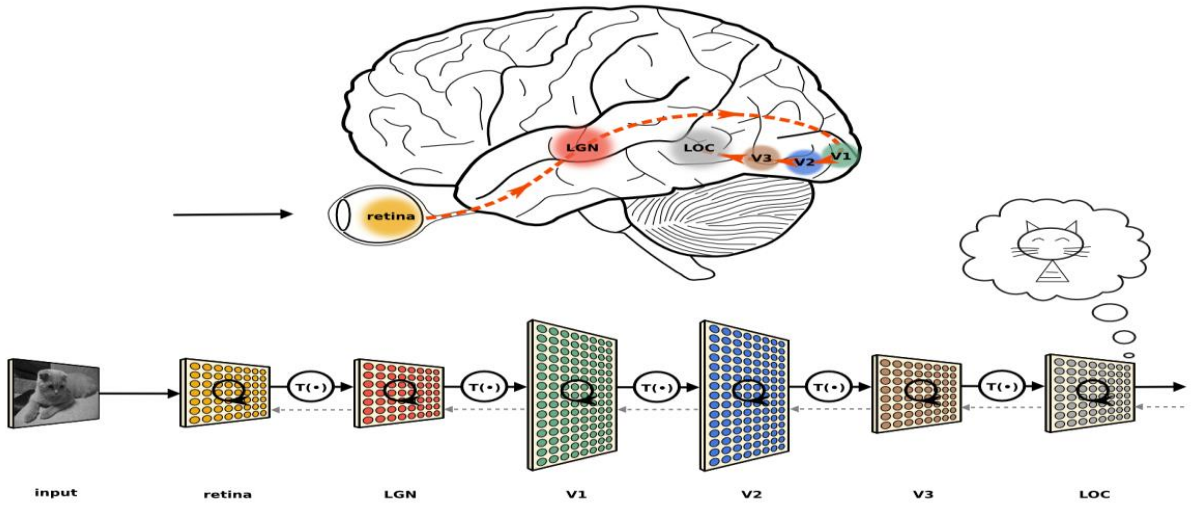
➢ *Problematica:*

Recunoașterea cifrelor scrise de mână este o provocare datorită variațiilor naturale ale scrisului fiecărui individ. Omului îi este ușor să înțeleagă aceste diferențe, dar cum ar fi dacă am crea un program capabil să simuleze modul în care creierul uman procesează informația?



➢ *Abordare:*

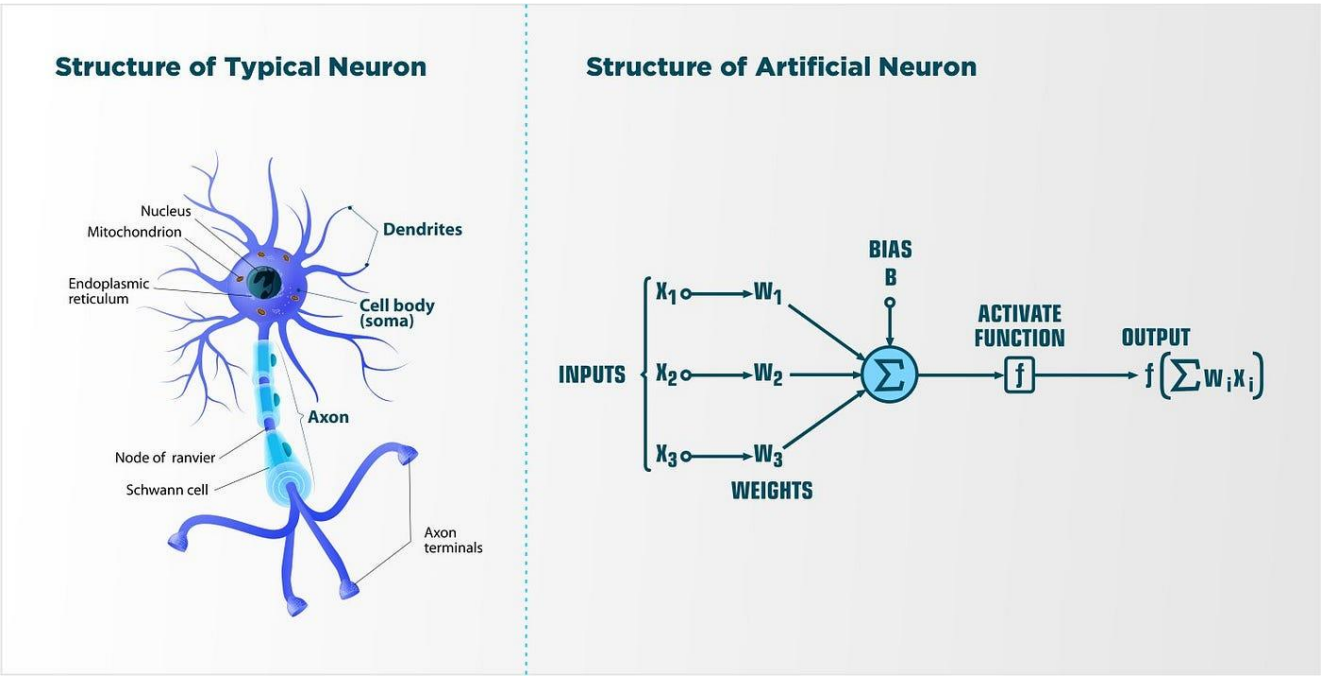
Rețelele neuronale vin ca o soluție, replicând acest proces pentru a oferi o abordare eficientă și precisă, depășind limitările programelor clasice.



Motivare:

Într-o eră marcată de progrese tehnologice rapide, *inteligența artificială* a devenit un domeniu de referință, cu aplicabilitate în aproape toate sectoarele. Alegerea temei acestei lucrări, crearea unei rețele neuronale, se înscrie în această tendință globală și răspunde interesului personal pentru explorarea unor metode inovatoare ce pot rezolva probleme complexe. Motivația provine din dorința de a înțelege mai bine modul în care rețelele neuronale artificiale pot fi utilizate pentru a analiza date, a învăța modele și a lua decizii automatizate. Totodată, această lucrare mi-a oferit o oportunitate de a aplica cunoștințe teoretice acumulate în domeniul matematicii și informaticii, integrându-le într-un proiect practic.

Conceptul pornește de la fundamentele neuronului, unitatea de bază a sistemului nervos uman. Prin capacitatea sa de a primi, procesa și transmite informații, neuronul biologic a devenit sursa de inspirație pentru dezvoltarea unuia dintre cele mai inovatoare concepte din informatică – neuronul artificial. Trecerea de la înțelegerea funcționării creierului la construirea unor modele informatice a fost posibilă datorită matematicii, statisticii și progreselor în tehnologie. Astfel, neuronul "artificial" reprezintă o abstracție, un model matematic simplificat, dar extrem de puternic, capabil să analizeze date, să învețe și să facă predicții.



Descrierea proiectului:

Proiectul meu constă în **realizarea arhitecturii unei rețele neuronale (R.N) de la zero**, cu scopul de a recunoaște cifre scrise de mână. Deși recunoașterea cifrelor este adesea asociată cu rețele neuronale convoluționale (R.N.C), în cazul de față am optat pentru o arhitectură *feed-forward* (fără straturi convoluționale), pentru a demonstra principiile de bază ale antrenării unei rețele neuronale și a le pune în practică într-o manieră cât mai transparentă. Rețeaua a fost ulterior integrată într-o **interfață web intuitivă**, care permite utilizatorilor să testeze funcționalitatea modelului direct online. Astfel, arhitectura generală a proiectului poate fi împărțită în două componente principale: **partea de inteligență artificială** (modelul R.N) și **partea de interfață** (aplicația web).

A. Componenta de inteligentă artificială

I. Elementele compoziționale și structura:

1. **”*Neuronul*”,** în esență, este o funcție matematică simplă. *Are proprii parametrii*: ponderile (notate cu ”w”) și ”factorul de ajustare” (notat cu ”b”), iar apoi *primește* niște *intrări* (un sir de valori numerice notat cu ”A”) pe care *le combina* folosind ponderile si *calculează* astfel *o sumă ponderată* (notată cu ”z”). La final *aplică* o funcție de *activare* iar *rezultatul* obținut este *ieșirea neuronului*.
2. ***Stratul*** este un grup de neuroni care funcționează în paralel. Fiecare neuron din stratul respectiv va avea parametrii proprii cu valori aleatorii, dar toți primesc același set de intrări. Eu am ales sa transpun principiile neuronului la un strat întreg pentru a face calculul mai compact, alcătuind astfel matricea pentru ponderi și matricele coloane pentru restul datelor. Asta mi-a facilitat calculul matematic cu operațiile matriceale după formula: $Z^{[L]}_{(n,1)} = W^{[L]}_{(n,m)} * A^{[L-1]}_{(m,1)} + B^{[L]}_{(n,1)}$, unde:
   - n reprezintă dimensiunea stratului curent, dimensiunea adică ”numărul de neuroni”.
   - m reprezintă dimensiunea stratului anterior.
   - L reprezintă indicele stratului.

   Am ales funcția matematica de activare **ReLU** (Rectified Linear Unit), pentru a introduce non-linearitate. Ea este definită astfel: ReLU(x) = max(0, x), pentru x număr real.

3. ***Structura rețelei neuronale*:**
   - **Stratul de intrare** nu e un strat cu neuroni în sens propriu pentru că nu prelucrează informația, nu conține parametrii și servește doar drept punct de intrare pentru date, anume stochează 784 activări (în cazul nostru, imaginea 28×28).
   - **Straturile ascunse** sunt destinate să proceseze și să transforme informațiile provenite de la stratul de intrare pana la stratul de ieșire. Eu am ales o configurație de 2 astfel de straturi, fiecare cu 16 neuroni, pentru o mai bună vizualizare grafică.
   - **Stratul de ieșire** primește informațiile procesate de straturile anterioare și generează rezultatul final al rețelei. Pentru problema noastră acest strat conține 10 neuroni (câte unul pentru fiecare clasă (cifra) de la 0 la 9). La acest strat am ales funcția matematica ***softmax*** pentru a obține probabilități pentru fiecare clasă, în loc de actăvări.

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Concretizarea noțiunilor de mai sus se găsesc în aceste secvențe de cod:

```
class SmpLayer {
    constructor(numNeurons, connectionsToPrevL, activFunction = null) {
        this.activFunction = activFunction
        this.weights; // Li x Li - 1
        this.biases; // Li x 1

        this.basicInit(numNeurons, connectionsToPrevL);
    }

    basicInit(numNeurons, connectionsToPrevL) {// Initialize weights and biases with random values
        this.weights = math.random([numNeurons, connectionsToPrevL], -1, 1);
        this.biases = math.random([numNeurons, 1], 0, 1);
    }
}

class SmpNetwork {

    constructor(size) {
        this.size = size; // hid + output (== index of Output Layer)
        this.network = new Array(size + 1); // 0 is input layer
    }

    initializeLayer(index, size, connections, activFunction) {
        this.network[index] = new SmpLayer(size, connections, activFunction);
    }
}

function setup() {
    simpleNN = new SmpNetwork(simpleNNsize);

    simpleNN.initializeLayer(1, 16, inputDim, ReLU); // HidL2
    simpleNN.initializeLayer(2, 16, 16, ReLU); // HidL2
    simpleNN.initializeLayer(3, 10, 16, softmax); // OutL
}
```

Acesta a fost punctul de pornire al rețelei neuronale, codul fiind la prima sa versiune, însă pe măsura parcurgerii documentului vor veni în completare modificările ce formează rețeaua în punctul final.

II. Mecanismul de antrenare

1. *Propagarea* datei de intrare prin rețea pentru a obține predicția:

- Se atribuie stratului de intrare data de intrare (”imaginea”).
- Se calculează pentru fiecare strat matricea sumei ponderate utilizând formula definită și se aplica funcția de activare corespunzătoare stratului în vederea obținerii matricei de activări.

```
feedForward(input) {
    this.network[0].setActivationsArr(input);

    let W_matrix, prevA_matrix, B_matrix, Z_matrix;
    for (let layerIndex = 1; layerIndex <= this.numHiddenLayers + 1; ++layerIndex) {
        W_matrix = this.network[layerIndex].getWeightsMatrix();
        prevA_matrix = this.network[layerIndex - 1].getActivationMatrix();
        B_matrix = this.network[layerIndex].getBiasesMatrix();

        Z_matrix = math.add(math.multiply(W_matrix, prevA_matrix), B_matrix);
        this.network[layerIndex].computeActivationFromMatrix(Z_matrix);
```

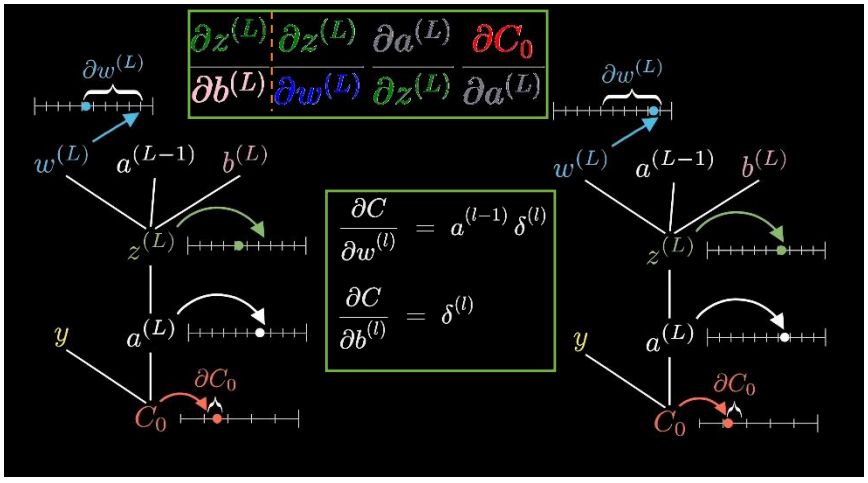
- Se stochează pentru fiecare strat activarea în vederea eliminării calculelor redundante.

2. *Calculul erorii* pentru a stabili cât a greșit rețeaua prin exprimarea unei valori**:**

- Se compară predicția cu eticheta (cifra corespunzătoare imaginii din setul de date) folosind funcția de cost. Eu am ales să implementez o funcție de cost de tipul clasificare multi-clasă și am utilizat **Cross-Entropy Loss**: se ia probabilitatea prezisă pentru clasa corectă și se aplică funcția logaritm, obținând astfel valoarea erorii.

3. *Propagarea înapoi* a modificărilor necesare în vederea minimizării erorii:

- Minimizarea erorii rețelei se realizează prin ajustarea parametrilor astfel încât să producă predicții cât mai precise. Pentru a determina ajustările necesare fiecărui neuron (adică a parametrilor acestuia), vom introduce termenul de **gradient al erorii** (”contribuția la eroare”), care este propagat din stratul de ieșire către straturile ascunse. Matematic vorbind este partea cea mai complexă.
- Se folosește **regula lanțului** și **derivatele**:



derivata măsoară cât de sensibilă este eroarea la un anumit parametru, de unde determinăm cu cât trebuie schimbat acel parametru pentru ca eroarea să scadă

$$\delta_l = (W_{l+1}^T \cdot \delta_{l+1}) \odot f'(Z_l)$$

Aceasta este formula folosită pentru a calcula **delta**, notație utilizată pentru a simplifica formula.

4. *Actualizarea parametrilor***:**

- Aceasta implică ajustarea fiecărui parametru în direcția opusă gradientului său, proporțional cu rata de învățare (”η”) care controlează mărimea pașilor de actualizare.

III. Setul de date și îmbunătățiri

Modified National Institute of Standards and Technology database (M.N.I.S.T):

- Pentru recunoașterea cifrelor scrise de mână, unul dintre cele mai utilizate seturi de date este M.N.I.S.T. . Acesta conține 60.000 de imagini de antrenament și 10.000 de imagini de test, fiecare imagine având 28x28 de pixeli.
- Pentru ca aceste date sa poată fii procesate de rețeaua mea neuronală acestea trebuie adaptate:
  - **Aplatizarea imaginilor**: fiecare imagine (28×28) este transformată într-un vector unidimensional de 784 de valori.
  - **Normalizare**: valorile pixelilor (inițial între 0 (negru) și 255 (alb)) sunt împărțite la 255, rezultând valori cuprinse între 0 și 1 ce denotă intensitatea pixelului.
  - **Etichetele**: pentru fiecare imagine, eticheta (cifra pe care o reprezintă imaginea, între 0 și 9) se păstrează pentru a permite calcularea erorii și ajustarea greutăților în timpul antrenării.
- Pentru a verifica performanța rețelei, se parcurg toate imaginile din setul de test, iar pentru fiecare imagine se face propagarea înainte și se determină predicția prin căutarea probabilității maxime, se acumulează erorile, iar la final se contorizează câte predicții sunt corecte și se calculează acuratețea si eroarea medie (număr de predicții corecte împărțit la total, și identic se procedează si pentru erorile acumulate).

Îmbunătățiri:

- În dezvoltarea proiectului, am implementat mai multe tehnici și optimizări pentru a îmbunătăți performanțele rețelei neuronale. Aceste îmbunătățiri se aplică atât la nivel de arhitectură a rețelei, cât și la procesul de antrenare, testare și utilizare practică. Mai jos sunt detaliate cele mai importante îmbunătățiri implementate:
  - **Cicluri de antrenament.** Pentru a obține o rețea performantă, este necesară antrenarea repetată pe datele de antrenament. Astfel, s-a stabilit un număr de cicluri de antrenament, în care rețeaua își ajustează parametrii pentru a produce răspunsuri cât mai corecte.
  - **Inițializarea parametrilor.** Pentru a evita problemele clasice de tip "*vanishing gradients*" sau "*exploding gradients*", am utilizat tehnica de **He Initialization**. Aceasta inițializează factorul de ajustare cu valoarea 0 și ponderile stratului cu valori aleatorii dintr-o distribuție gaussiană scalată : $\sqrt{\frac{2}{numărul\ de\ conexiuni\ de\ intrare}}$
  - **Funcții de activare adecvate.** Am utilizat **ReLU** (Rectified Linear Unit) în favoarea altor funcții precum Sigmoid sau Tanh, pentru toate straturile ascunse, deoarece aceasta ajută la spargerea non-liniarităților și permite propagarea gradientului în rețele mai adânci.
  - **Mini-batch training (Stochastic Gradient Descent).** Aceasta este o tehnică de optimizare utilizată pentru antrenarea rețelelor neuronale și a altor modele de machine learning. Este o variantă a algoritmului Gradient Descent care face actualizări mai frecvente, ceea ce poate accelera procesul de antrenare. În loc să actualizez parametrii după fiecare exemplu, am utilizat eșaloane de antrenament (**mini-batchs)**. Gradientul a fost calculat și acumulat pentru un eșalon de dimensiune prestabilită (de exemplu 10 imagini), iar parametrii au fost actualizate după fiecare lot. Această tehnică ajută la reducerea fluctuațiilor din gradient și la o antrenare mai stabilă.
  - **"Amestecarea" datelor la începutul fiecărui ciclu de antrenament.**
  - **Optimizarea cu rata de învățare adaptivă.** Am implementat o rată de învățare variabilă, care scade pe măsură ce ciclurile de antrenament avansează. Formula utilizată a fost

B. Componenta de interfață web

Aplicația web reprezintă modul principal prin care utilizatorii pot interacționa cu modelul neuronal (acesta este ”frontend-ul”). Am proiectat o interfață intuitivă cu ajutorul limbajului JavaScript și a librăriei extensie p5.js, care permite testarea în timp real a rețelei neuronale, oferind feedback instant utilizatorului. Interfața web de asemenea facilitează vizualizarea arhitecturii rețelei și explorarea capacității acesteia de a recunoaște cifre desenate de utilizator în timp real.

Codul HTML care cuprinde toate fișierele componente ce alcătuiesc aplicația web:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>RNC</title>
    <link rel="stylesheet" type="text/css" href="style.css">
    <script src="libraries/p5.min.js"></script>
    <script src="libraries/p5.sound.min.js"></script>
  </head>

  <body>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/mathjs/10.6.0/math.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/FileSaver.js/2.0.5/FileSaver.min.js"></script>
    <script src="obj_Library/neural.network.js"></script>
    <script src="obj_Library/layer.js"></script>
    <script src="obj_Library/utils.js"></script>
    <script src="obj_Library/cell.js"></script>
    <script src="obj_Library/grid.js"></script>
    <script src="main.js"></script>
  </body>
</html>
```

Secvență de cod din fișierul script ”main.js”:

```
function draw() {
    background(255);
    brushHardness = brushSlider.value();
    grid.show();
    if (training == false) {
        if (painting) { brush() }
        let input = flatten(grid.getGrid());
        NN.feedForward(input);
        NN.predict();
        PredictionText.html('Predicted: ' + NN.getPrediction());
        PredictionText.position(canvas.x + (imgWidth / 2) - 60, canvas.y + imgWidth);
        PredictionText.style('font-size', '32px');
    }
    drawNetwork();
}
```

Despre limbajul de programare si instrumente:

Pentru dezvoltarea și testarea aplicației, am utilizat Visual Studio Code, un editor modern și performant. Acesta mi-a oferit autocompletare inteligentă pentru limbaje pe care le-am folosit, precum HTML, CSS și JavaScript și **integrarea cu Git**, pentru gestionarea versiunilor codului.

O caracteristică distinctivă a proiectului este utilizarea **Programării Orientate pe Obiecte (OOP)**, care facilitează organizarea logicii și modularitatea codului. Clasele și obiectele ajută la modelarea entităților aplicației, cum ar fi rețeaua neuronală, straturile, grila pentru desen, celulele grilei ș.a.m.d. Exemplu:

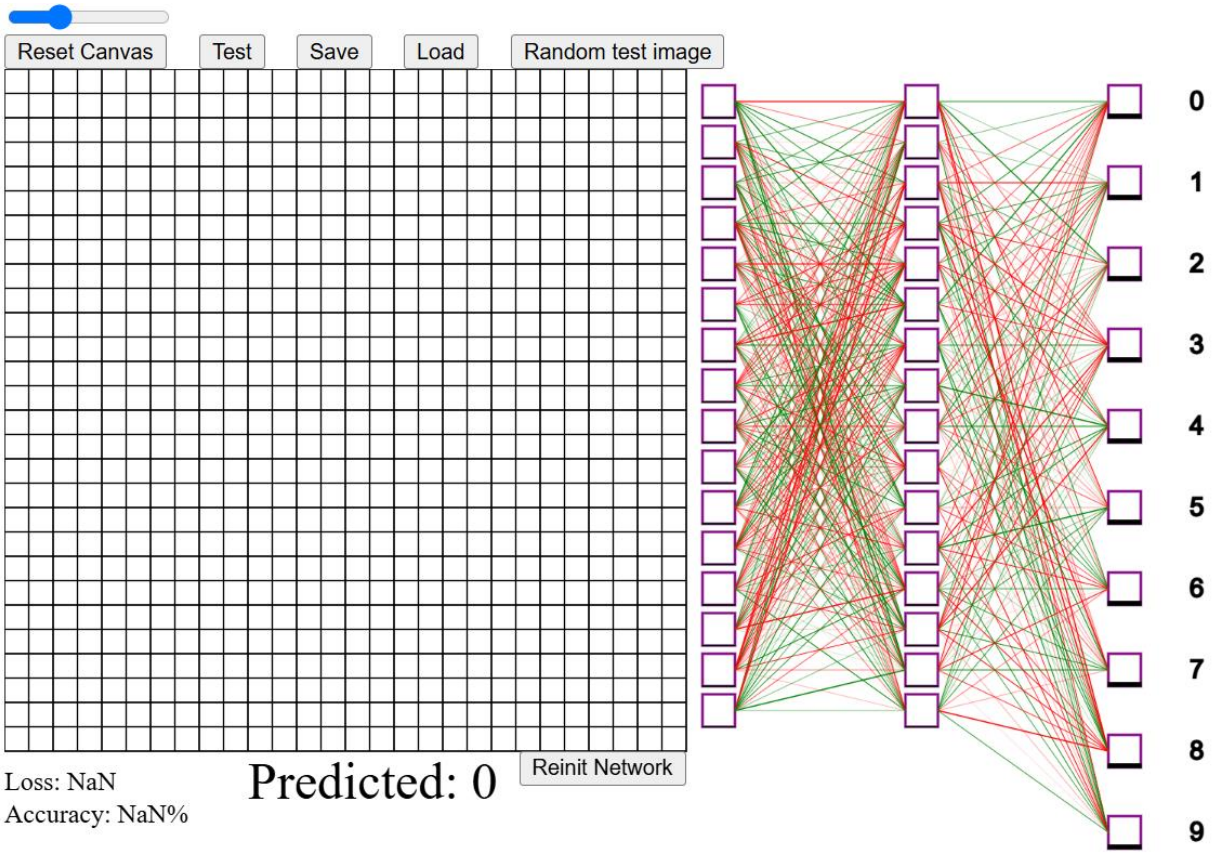
```
class Cell {
    constructor(x, y, width, val){
        this.x = x;
        this.y = y;
        this.cellWidth = width;
        this.val = val; // cuprinsă intre 0 și 1, indică activarea pixelului (min-max)
    }

    show() {
        fill((1 - this.val) * 255); // voi umple cu negru daca e activat, sau alb in caz contrat
        square(this.x * this.cellWidth, this.y * this.cellWidth, this.cellWidth);
    }

    // ------------------- Setters -------------------
    setVal(newVal) {
        this.val = newVal;
    }

    // ------------------- Getters -------------------
    getVal() {
        return this.val;
    }
}
```

HTML (HyperText Markup Language) este utilizat pentru a defini structura de bază a paginii web

CSS (Cascading Style Sheets) este utilizat pentru a defini stilul și aspectul interfeței aplicației, gestionând culorile, dimensiunile și poziționarea elementelor vizuale

JavaScript este limbajul principal de programare utilizat pentru funcționalitatea aplicației. Include scripturi pentru implementarea logicii neuronale (prin fișiere precum neural.network.js, layer.js și utils.js), pentru manipularea interactivă a interfeței și pentru procesarea datelor utilizatorilor. Biblioteci utilizate: **p5.js** utilizată pentru grafică, **math.js** facilitează operații matematice necesare pentru funcționarea rețelei neuronale și **FileSaver.js** ce permite descărcarea fișierelor, pentru a salva R.N.

Instrucțiuni interfață:



*Canvas-ul pentru desen (**stânga**)*:

- *Zona grilă*: Un pătrat format din 28×28 celule, unde utilizatorul poate **desena** cifre folosind mouse-ul și colorând celulele dorite prin click stânga.
- *Slider-ul pentru **grosimea pensulei**:* Deasupra canvas-ului se află un slider care ajustează duritatea pensulei. Trage slider-ul spre stânga pentru a reduce grosimea pensulei și spre dreapta pentru a o mări.

*Vizualizarea rețelei neuronale (**dreapta**)*:

- Straturile rețelei: Fiecare strat al rețelei este reprezentat prin **neuroni** (pătrate mici). **Neuronii** din stratul de ieșire sunt etichetați cu numerele 0–9, corespunzând claselor (cifrele recunoscute).
- Conexiunile dintre straturi: Liniile reprezintă ponderile dintre neuroni. Liniile verzi indică greutăți pozitive, iar cele roșii greutăți negative. Grosimea liniei reflectă modulul valorii ponderii.

*Zona de butoane (**sus**)*:

- Reset Canvas: Șterge tot desenul curent de pe canvas, permițând utilizatorului să înceapă din nou.
- Test: Evaluează rețeaua pe setul de date de test MNIST și actualizează afișările de sub canvas.
- Save: Salvează configurația rețelei (ponderile și factorul de ajustare) într-un fișier JSON.
- Load: Încarcă un model salvat anterior dintr-un fișier JSON.
- Random test image: Încarcă o imagine aleatorie din setul de test MNIST și o afișează pe grilă.

Butonul **Reinit Network**: Reinitializează rețeaua neuronală, ștergând toți parametrii curenți și creând o rețea nouă cu parametrii proaspăt inițializați.

NaN indică că nu au fost înregistrate valori pentru a putea fi afișate.

Bibliografie:

- Serie explicativă pe YouTube despre rețele neuronale
- https://www.3blue1brown.com/topics/neural-networks
- Datele MNIST folosite

➢ **GitHub** : *Profil GitHub*

Sfârșit!